Loss modelling and reserving analytics in R

A hands-on workshop

Katrien Antonio & Jonas Crevecoeur

IA|BE workshop | June 3, 10 & 17, 2021

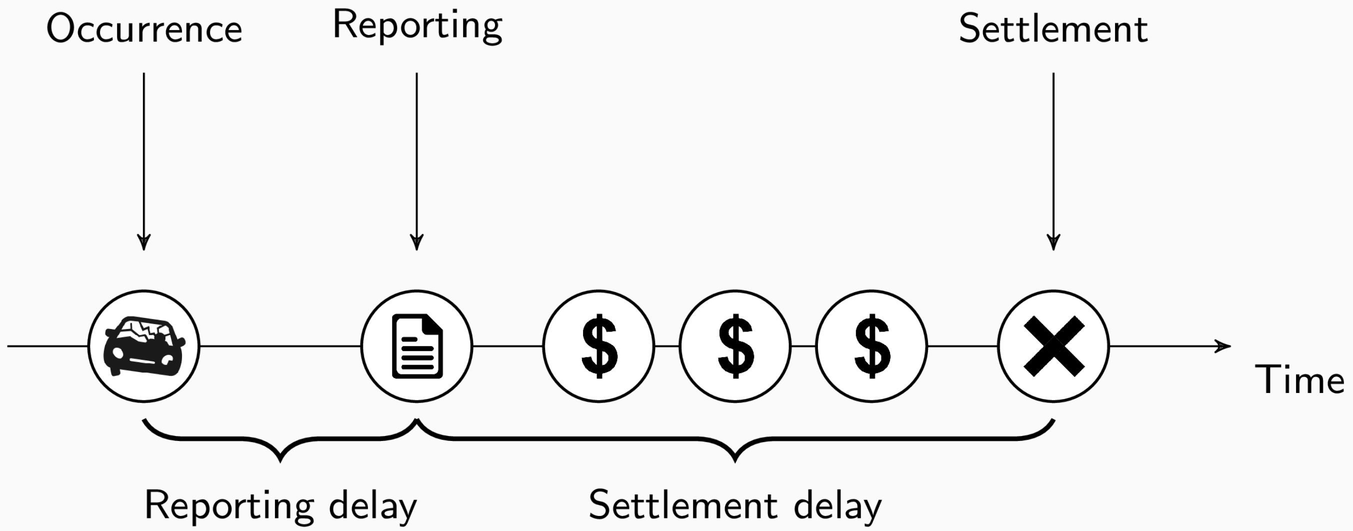
Today's Outline

- Motivation and strategies
- Reserving data structures
  - individual and aggregated reserving data
  - runoff triangles
- Claims reserving with triangles
  - chain ladder method
  - {ChainLadder} package
  - chain ladder implementation with GLMs
- When the chain ladder method fails
  - detection tools for triangle stability
  - creating homogeneous triangles
- Research outlook
  - granular reserving methods for predicting the IBNR reserve
  - individual reserving methods for predicting the RBNS reserve

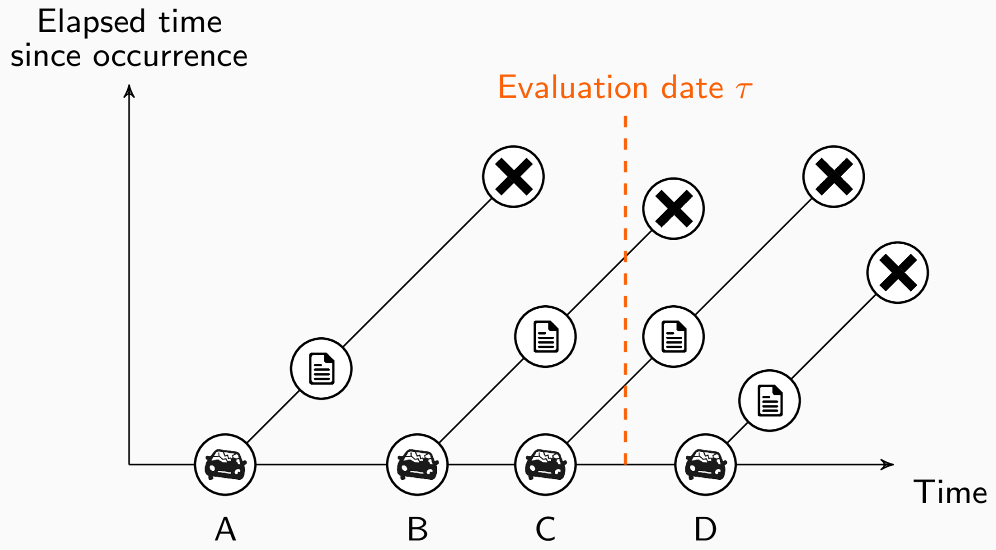
Motivation and strategies

Motivation

Insurers observe the **detailed evolution** of **individual claims** over time:



Motivation (continued)



Observed claims are **censored** due to **delays** (reporting, settlement) in the claim development process.

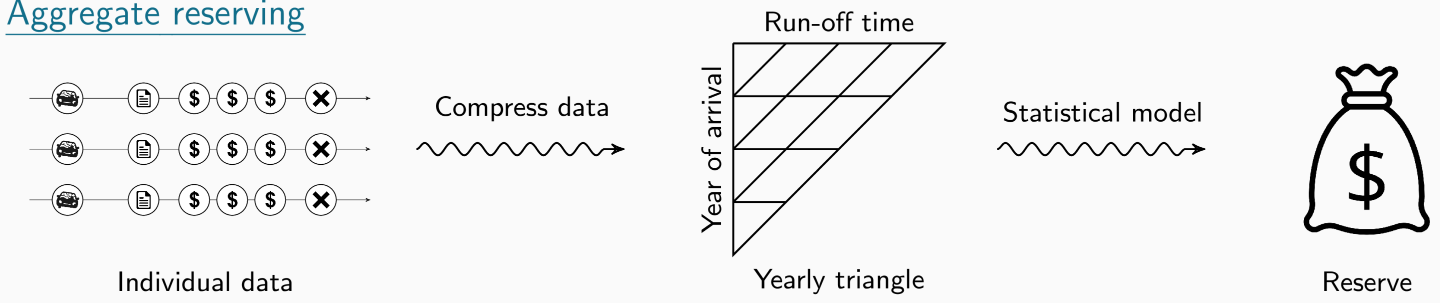
Reserve: future costs for claims that occurred in **past exposure periods** (claims B and C)

- RBNS: Reported, But Not yet Settled claims (claim B)
- IBNR: Incurred, But Not yet Reported claims (claim C).

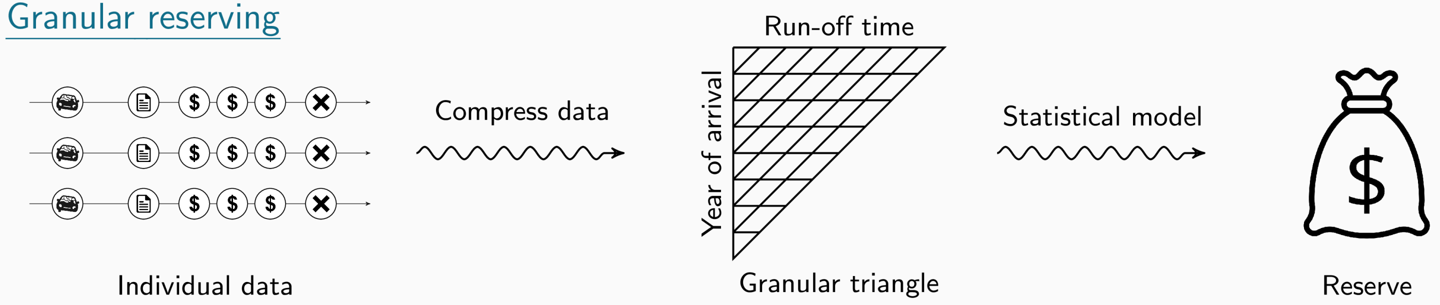
Pricing: all costs for claims that will occur in **future** insured **exposure periods** (claim D).

Three strategies for non-life reserving

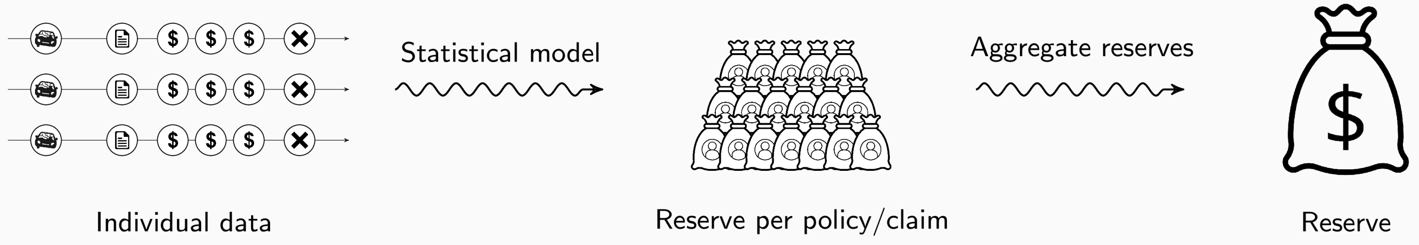
Aggregate reserving



Granular reserving



Individual reserving



Three strategies for non-life reserving (continued)

Aggregate reserving

First, we aggregate the past claim history into a **small number of summary statistics**, then we predict the total reserve based on these summary statistics.

Granular reserving

First, we aggregate the past claim history into a **large number of summary statistics**, then we predict the total reserve based on these summary statistics.

Individual reserving

First, we predict the future costs for **individual claims**, then we aggregate these individual predictions to predict the total reserve.

Data set used in this session

We illustrate reserving analytics based on a **simulated dataset** registering the **detailed development** of 30,000 claims occurring between January 1, 2010 and December 31, 2020. Data is available in a daily and yearly format.

Daily data:

- with claim events recorded at daily precision
- with records correspond to the reporting of a claim or a payment being made.

The data is stored in a .rds file in the `data` directory in the course material:

```
# install.packages("rstudioapi")
dir <- dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(dir)

reserving_daily <- readRDS("data/reserving_data_daily.rds")
```

```
##   accident_number accident_date reporting_date settlement_date reporting_delay settlement_delay payment_date payment_size
## 1               1    2012-12-02     2012-12-07      2013-03-10               5               98   2012-12-07       0.0000
## 2               1    2012-12-02     2012-12-07      2013-03-10               5               98   2012-12-13     446.2251
## 3               1    2012-12-02     2012-12-07      2013-03-10               5               98   2012-12-24     490.1792
## 4               1    2012-12-02     2012-12-07      2013-03-10               5               98   2013-01-06     587.7409
## 5               1    2012-12-02     2012-12-07      2013-03-10               5               98   2013-01-24    1726.0902
## 6               2    2014-02-04     2014-02-10      2014-08-21               6              198   2014-02-10       0.0000
```

Data set used in this session

We illustrate reserving analytics based on a **simulated dataset** registering the **detailed development** of 30,000 claims occurring between January 1, 2010 and December 31, 2020. Data is available in a daily and yearly format.

Yearly data with one record per claim per development year:

- for each development year the data registers:
  - close is 1 when the claim closes in the current development year, 0 otherwise
  - payment is 1 when there is at least one payment, 0 otherwise
  - size: total amount paid for the claim in the current development year

The data is stored in a .rds file in the `data` directory in the course material:

```
reserving_yearly <- readRDS("data/reserving_data_yearly.rds")
```

```
## # A tibble: 6 x 10
##   accident_number accident_year reporting_year settlement_year development_year calendar_year  size payment close is_closed
##             <int>         <dbl>          <dbl>           <dbl>            <dbl>         <dbl> <dbl>   <dbl> <dbl>     <dbl>
## 1               1             2              2               3                1             2  936.       1     0         0
## 2               1             2              2               3                2             3 2314.       1     1         1
## 3               2             4              4               4                1             4 3537.       1     1         1
## 4               3             6              6               6                1             6 1975.       1     1         1
## 5               4             9             10              10                1             9    0        0     0         0
## 6               4             9             10              10                2            10 3107.       1     1         1
```

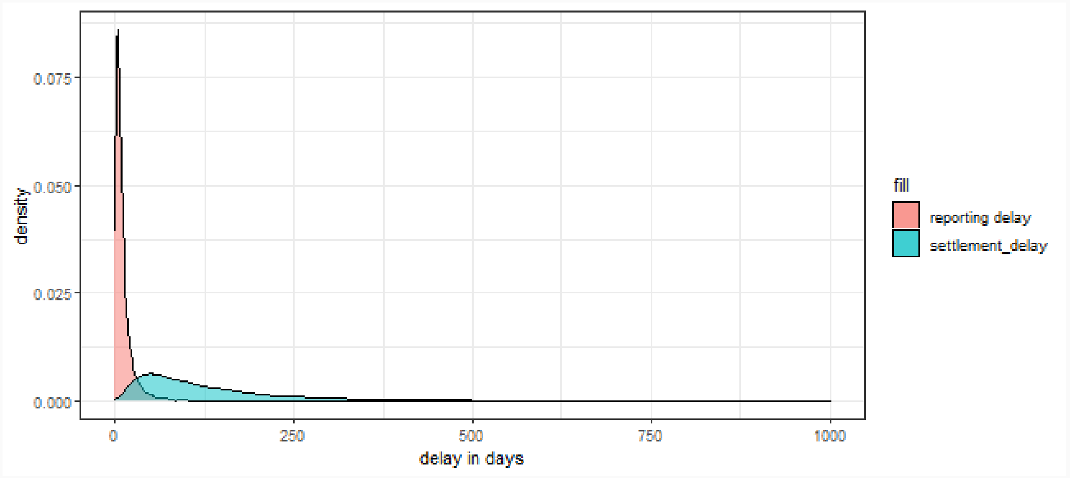
Your turn

In this warm up exercise you load the `reserving_daily` or `reserving_yearly` data and get some feel for the data. Choose for each task the format that is most convenient (daily or yearly).

1. Visualize the reporting and settlement delay of claims with a density plot.
2. When was the last payment registered in the data set?
   What do you conclude?
3. What is the average number of payments per claim?
4. Calculate the number of claims per accident year.

For **Q.1** we plot the density of reporting and settlement delay:

```
ggplot(data = reserving_daily) +
  theme_bw() +
  geom_density(aes(reporting_delay, fill = 'reporting delay'),
               alpha = .5) +
  geom_density(aes(settlement_delay, fill = 'settlement_delay')
               alpha = .5) +
  xlab('delay in days') +
  xlim(c(0, 1000))
```



For **Q.2** the last payment was in 2025, i.e. the simulated data is not yet censored!

```
max(reserving_daily$payment_date)
```

```
## [1] "2025-05-03"
```

For **Q.3** we average the number of payments per claim. Count only records with `payment_size > 0` to exclude the reporting event.

```
reserving_daily %>%
  group_by(accident_number) %>%
  summarise(payments = sum(payment_size > 0)) %>%
  ungroup() %>%
  summarise(average = mean(payments))
```

```
## # A tibble: 1 x 1
##   average
##     <dbl>
## 1    2.38
```

For **Q.4** we group the data by accident year and summarise the number of claims.

```
library(tidyverse)
reserving_yearly %>%
        filter(development_year == 1) %>%
        group_by(accident_year) %>%
        summarise(num_claims = n())
```

```
## # A tibble: 11 x 2
##    accident_year num_claims
##  *         <dbl>      <int>
##  1             0       2743
##  2             1       2725
##  3             2       2810
##  4             3       2756
##  5             4       2705
##  6             5       2706
##  7             6       2703
##  8             7       2708
##  9             8       2640
## 10             9       2751
## 11            10       2391
```

We apply the filter `development_year == 1` to count each claim only once. Claim frequency is slightly lower in accident year 10.

Censoring the data

The simulated data is not censored, i.e. we observe the full development of all claims that occur before the end of 2020. In practice the available data set will be censored and we will only observe:

- reported claims
- development information in past calendar years.

We censor the data assuming that we observe data until the end of 2020:

```r
observed_daily <- reserving_daily %>%
  filter(payment_date <= as.Date('2020-12-31'))

unobserved_daily <- reserving_daily %>%
  filter(payment_date > as.Date('2020-12-31'))

observed_yearly <- reserving_yearly %>%
  filter(calendar_year <= 10,
         reporting_year <= 10)

unobserved_yearly <- reserving_yearly %>%
  filter(calendar_year > 10 | reporting_year > 10)
```

IBNR and RBNS reserves

Reserving models predict the total payments in the unobserved part of the data:

```
reserve_actual <- sum(unobserved_yearly$size)
reserve_actual
```

```
## [1] 2149312
```

Some reserving methods split this reserve into

- IBNR: a reserve for Incurred, But Not (yet) Reported claims
- RBNS: a reserve for Reported, But Not (yet) Settled claims.

```
unobserved_yearly %>%
  mutate(reported = (reporting_year <= 10)) %>%
  group_by(reported) %>%
  summarise(reserve = sum(size))
```

```
## # A tibble: 2 x 2
##   reported  reserve
## * <lgl>       <dbl>
## 1 FALSE     134124.
## 2 TRUE     2015188.
```

Reserving data structures: individual and aggregated data

From individual to aggregated data

Individual triangle

The data set `reserving_yearly` describes the development of **individual claims discretised by calendar year**.

We can represent the claim characteristics in this data set (settlement, payment, payment size) with **two-dimensional tables** in which the rows represent **individual claims** and columns represent **development years**.

For payment size, this table consists of cells $U_{k,j}^{\texttt{size}}$, denoting the total amount paid for claim k in development year j.

Runoff triangle

Many reserving models go one step further and remove individual claim characteristics by aggregating claims by occurrence year. The data is then represented in a (small) two-dimensional table, the so-called **incremental runoff triangle**.

$$X_{i,j}^{\texttt{size}} = \sum_{\texttt{occ.year(k)}=i} U_{k,j}^{\texttt{size}}.$$

As a result of the Law of Large Numbers (LLN), aggregating data from many claims into runoff triangles **reduces the variance** when all claims are independent realizations from the same distribution.

From individual to aggregated data (continued)

We construct a runoff triangle by aggregating the individual claims by `accident_year`.

```
observed_yearly %>%
  group_by(accident_year, development_year) %>%
  summarise(value = sum(size)) %>%
  pivot_wider(values_from = value,
              names_from = development_year,
              names_prefix = 'DY.')
```

`pivot_wider` creates for each unique value of `names_from` a new column

```
## # A tibble: 11 x 10
## # Groups:   accident_year [11]
##    accident_year    DY.1    DY.2   DY.3  DY.4  DY.5  DY.6  DY.7  DY.8  DY.9
##            <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1             0 4075901. 1610076. 112557. 24668. 10680. 8840. 9479. 6217. 3468.
##  2             1 4037070. 1461939.  62622. 18518.     0    NA    NA    NA    NA
##  3             2 4322579. 1774121. 105850. 41429. 24099. 7031.    0    NA    NA
##  4             3 4162489. 1616297. 104487. 15982.    NA    NA    NA    NA    NA
##  5             4 3903230. 1605598. 160906. 24194. 11622.   NA    NA    NA    NA
##  6             5 4052230. 1623940. 131721. 26686. 14096. 6262.   NA    NA    NA
##  7             6 4087125. 1593968.  96484. 40310.  4930.   NA    NA    NA    NA
##  8             7 4059058. 1771141. 170375. 37317.    NA    NA    NA    NA    NA
##  9             8 4033678. 1309634. 140624.    NA    NA    NA    NA    NA    NA
## 10             9 4141278. 1615333.      NA    NA    NA    NA    NA    NA    NA
## 11            10 3443223.      NA      NA    NA    NA    NA    NA    NA    NA
```

From individual to aggregated data (continued)

A more sophisticated function for computing incremental runoff triangles is available in the R code.

```r
incremental_triangle(observed_yearly,
                     variable = 'payment',
                     lower_na = TRUE)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,] 2054  523   17    6    2    1    1    1    1     0     0
##  [2,] 2074  482   16    4    0    0    0    0    0     0    NA
##  [3,] 2106  549   25    5    4    2    0    0    0    NA    NA
##  [4,] 2033  532   24    5    0    0    0    0   NA    NA    NA
##  [5,] 2008  473   32    7    2    0    0   NA   NA    NA    NA
##  [6,] 2035  498   26    4    2    2   NA   NA   NA    NA    NA
##  [7,] 2028  487   17    5    1   NA   NA   NA   NA    NA    NA
##  [8,] 2030  528   34    9   NA   NA   NA   NA   NA    NA    NA
##  [9,] 2005  444   24   NA   NA   NA   NA   NA   NA    NA    NA
## [10,] 2100  507   NA   NA   NA   NA   NA   NA   NA    NA    NA
## [11,] 1784   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
```

Using this function you can easily inspect the reserving data with multiple triangles.

```r
incremental_triangle(observed_yearly,
                     variable = 'size',
                     lower_na = TRUE)
```

```
##          [,1]    [,2]      [,3]     [,4]      [,5]     [,6]    [,7]     [,8]     [,9]
##  [1,] 4075901 1610076 112556.58 24668.43 10679.546 8840.487 9478.77 6217.266 3468.364
##  [2,] 4037070 1461939  62622.23 18517.80     0.000    0.000    0.00    0.000    0.000
##  [3,] 4322579 1774121 105850.01 41429.20 24098.625 7030.788    0.00    0.000    0.000
##  [4,] 4162489 1616297 104486.62 15982.39     0.000    0.000    0.00    0.000       NA
##  [5,] 3903230 1605598 160905.57 24194.32 11622.131    0.000    0.00       NA       NA
##  [6,] 4052230 1623940 131721.42 26685.98 14095.805 6261.739      NA       NA       NA
##  [7,] 4087125 1593968  96483.79 40310.11  4930.009       NA      NA       NA       NA
##  [8,] 4059058 1771141 170374.72 37317.06        NA       NA      NA       NA       NA
##  [9,] 4033678 1309634 140624.40       NA        NA       NA      NA       NA       NA
## [10,] 4141278 1615333        NA       NA        NA       NA      NA       NA       NA
## [11,] 3443223      NA        NA       NA        NA       NA      NA       NA       NA
```

Cumulative runoff triangles

Cumulative runoff triangles are constructed by computing the cumulative row sum of an **incremental runoff triangle**, i.e.

$$C_{i,j}^{\texttt{size}} = \sum_{l=1}^{j} X_{i,l}^{\texttt{size}}.$$

When using a new reserving method, check whether the incremental or cumulative runoff triangle is required!

```r
incremental_triangle(observed_yearly,
                     variable = 'size',
                     lower_na = TRUE)
```

```
##          [,1]    [,2]      [,3]     [,4]      [,5]     [,6]    [,7]     [,8]     [,9]
##  [1,] 4075901 1610076 112556.58 24668.43 10679.546 8840.487 9478.77 6217.266 3468.364
##  [2,] 4037070 1461939  62622.23 18517.80     0.000    0.000    0.00    0.000    0.000
##  [3,] 4322579 1774121 105850.01 41429.20 24098.625 7030.788    0.00    0.000    0.000
##  [4,] 4162489 1616297 104486.62 15982.39     0.000    0.000    0.00    0.000       NA
##  [5,] 3903230 1605598 160905.57 24194.32 11622.131    0.000    0.00       NA       NA
##  [6,] 4052230 1623940 131721.42 26685.98 14095.805 6261.739      NA       NA       NA
##  [7,] 4087125 1593968  96483.79 40310.11  4930.009       NA      NA       NA       NA
##  [8,] 4059058 1771141 170374.72 37317.06        NA       NA      NA       NA       NA
##  [9,] 4033678 1309634 140624.40       NA        NA       NA      NA       NA       NA
## [10,] 4141278 1615333        NA       NA        NA       NA      NA       NA       NA
## [11,] 3443223      NA        NA       NA        NA       NA      NA       NA       NA
```

```r
cumulative_triangle(observed_yearly,
                    variable = 'size',
                    lower_na = TRUE)
```

```
##          [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]    [,9]   [,10]
##  [1,] 4075901 5685977 5798533 5823202 5833881 5842722 5852200 5858418 5861886 5861886
##  [2,] 4037070 5499009 5561632 5580150 5580150 5580150 5580150 5580150 5580150 5580150
##  [3,] 4322579 6096700 6202550 6243979 6268078 6275109 6275109 6275109 6275109      NA
##  [4,] 4162489 5778787 5883273 5899256 5899256 5899256 5899256 5899256      NA      NA
##  [5,] 3903230 5508828 5669733 5693928 5705550 5705550 5705550      NA      NA      NA
##  [6,] 4052230 5676170 5807891 5834577 5848673 5854935      NA      NA      NA      NA
##  [7,] 4087125 5681093 5777577 5817887 5822817      NA      NA      NA      NA      NA
##  [8,] 4059058 5830199 6000574 6037891      NA      NA      NA      NA      NA      NA
##  [9,] 4033678 5343311 5483936      NA      NA      NA      NA      NA      NA      NA
## [10,] 4141278 5756610      NA      NA      NA      NA      NA      NA      NA      NA
## [11,] 3443223      NA      NA      NA      NA      NA      NA      NA      NA      NA
```

Claims reserving with triangles

Mack chain ladder method

Assumptions underneath Mack's (1993) **chain ladder method**:

1. There exist development factors f_j such that

$$E(C_{i,j+1} \mid C_{i,1}, \dots, C_{i,j}) = C_{i,j} \cdot f_j \quad \text{for} \quad 1 \leq i \leq l, 1 \leq j \leq l-1,$$

   with l the dimension of the runoff triangle.

2. There exist parameters σ_j such that

$$Var(C_{i,j+1} \mid C_{i,1}, \dots, C_{i,j}) = C_{i,j} \cdot \sigma_j^2 \quad \text{for} \quad 1 \leq i \leq l, 1 \leq j \leq l-1.$$

3. Occurrence years are independent.

Remarks:

- Method based on the **cumulative triangle**!
- Assumption 1. and 2. add a **Markov property**. Future evolutions of the reserve depend only on the last known situation!

Mack chain ladder method (continued)

Estimating the **development factors** $\hat{f}_j$:

$$\hat{f}_j = \frac{\sum_{i=1}^{l-j} C_{i,j+1}}{\sum_{i=1}^{l-j} C_{i,j}}.$$

```r
triangle <- cumulative_triangle(observed_yearly, variable = 'size')
l <- nrow(triangle)
f <- rep(0, l-1)

for(j in 1:(l-1)) {
  f[j] <- sum(triangle[1:(l-j), j+1]) / sum(triangle[1:(l-j), j])
}

f
```

```
##  [1] 1.391002 1.021245 1.004906 1.001600 1.000630 1.000323 1.000263 1.000196 1.000000 1.000000
```

Development factors should converge to 1 for high development years.

Mack chain ladder method (continued)

Use the development factors to estimate cells in the lower triangle ($i + j > l + 1$):

$$\hat{C}_{i,j} = \hat{C}_{i,j-1} \cdot \hat{f}_{j-1} \quad \text{for} \quad i + j > l + 1.$$

```r
triangle_completed <- triangle
for(j in 2:l) {
  triangle_completed[l:(l-j+2), j] <- triangle_completed[l:(l-j+2), j-1] * f[j-1]
}
```

Completed cumulative triangle:

```r
triangle_completed
```

```
##          [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]    [,9]   [,10]
##  [1,] 4075901 5685977 5798533 5823202 5833881 5842722 5852200 5858418 5861886 5861886
##  [2,] 4037070 5499009 5561632 5580150 5580150 5580150 5580150 5580150 5580150 5580150
##  [3,] 4322579 6096700 6202550 6243979 6268078 6275109 6275109 6275109 6275109 6275109
##  [4,] 4162489 5778787 5883273 5899256 5899256 5899256 5899256 5899256 5900411 5900411
##  [5,] 3903230 5508828 5669733 5693928 5705550 5705550 5705550 5707052 5708170 5708170
##  [6,] 4052230 5676170 5807891 5834577 5848673 5854935 5856829 5858371 5859518 5859518
##  [7,] 4087125 5681093 5777577 5817887 5822817 5826485 5828370 5829905 5831046 5831046
##  [8,] 4059058 5830199 6000574 6037891 6047551 6051361 6053318 6054913 6056098 6056098
##  [9,] 4033678 5343311 5483936 5510838 5519655 5523132 5524919 5526374 5527456 5527456
## [10,] 4141278 5756610 5878910 5907750 5917202 5920930 5922845 5924405 5925565 5925565
## [11,] 3443223 4789529 4891282 4915278 4923142 4926243 4927837 4929134 4930100 4930100
```

Completed incremental triangle:

```r
require(ChainLadder)
cum2incr(triangle_completed)
```

```
##          [,1]    [,2]      [,3]     [,4]      [,5]     [,6]     [,7]     [,8]      [,
##  [1,] 4075901 1610076 112556.58 24668.43 10679.546 8840.487 9478.770 6217.266 3468.36
##  [2,] 4037070 1461939  62622.23 18517.80     0.000    0.000    0.000    0.000    0.00
##  [3,] 4322579 1774121 105850.01 41429.20 24098.625 7030.788    0.000    0.000    0.00
##  [4,] 4162489 1616297 104486.62 15982.39     0.000    0.000    0.000    0.000 1155.08
##  [5,] 3903230 1605598 160905.57 24194.32 11622.131    0.000    0.000 1502.662 1117.44
##  [6,] 4052230 1623940 131721.42 26685.98 14095.805 6261.739 1893.935 1542.505 1147.07
##  [7,] 4087125 1593968  96483.79 40310.11  4930.009 3667.976 1884.732 1535.009 1141.50
##  [8,] 4059058 1771141 170374.72 37317.06  9660.235 3809.544 1957.475 1594.254 1185.56
##  [9,] 4033678 1309634 140624.40 26902.60  8816.985 3477.005 1786.605 1455.090 1082.07
## [10,] 4141278 1615333 122299.67 28840.23  9452.018 3727.432 1915.283 1559.891 1160.00
## [11,] 3443223 1346306 101753.93 23995.21  7864.126 3101.242 1593.525 1297.837  965.13
```

Mack chain ladder method (continued)

The estimated reserve is the sum of the estimates in the **lower half** of the **incremental runoff triangle**.

```
triangle_completed_incr <- cum2incr(triangle_completed)
lower_triangle <- row(triangle_completed_incr) + col(triangle_completed_incr) > l+1

reserve_cl <- sum(triangle_completed_incr[lower_triangle])

data.frame(reserve_cl = reserve_cl,
           reserve_actual = reserve_actual,
           difference = reserve_cl - reserve_actual,
           relative_difference_pct = (reserve_cl - reserve_actual) / reserve_actual * 100)
```

```
##   reserve_cl reserve_actual difference relative_difference_pct
## 1    1734146        2149312  -415165.1               -19.31619
```

Mack chain ladder method (continued)

These calculations are already implemented in the {ChainLadder} package.

```r
require(ChainLadder)
triangle <- cumulative_triangle(observed_yearly, variable = 'size')
MackChainLadder(triangle)
```

```
## MackChainLadder(Triangle = triangle)
##
##       Latest Dev.To.Date  Ultimate     IBNR Mack.S.E CV(IBNR)
## 1  5,861,886       1.000 5,861,886 0.00e+00        0      NaN
## 2  5,580,150       1.000 5,580,150 0.00e+00      635      Inf
## 3  6,275,109       1.000 6,275,109 9.31e-10      693 7.44e+11
## 4  5,899,256       1.000 5,900,411 1.16e+03    2,417 2.09e+00
## 5  5,705,550       1.000 5,708,170 2.62e+03    4,164 1.59e+00
## 6  5,854,935       0.999 5,859,518 4.58e+03    6,288 1.37e+00
## 7  5,822,817       0.999 5,831,046 8.23e+03    7,643 9.29e-01
## 8  6,037,891       0.997 6,056,098 1.82e+04   11,916 6.54e-01
## 9  5,483,936       0.992 5,527,456 4.35e+04   14,755 3.39e-01
## 10 5,756,610       0.971 5,925,565 1.69e+05   40,015 2.37e-01
## 11 3,443,223       0.698 4,930,100 1.49e+06  126,084 8.48e-02
##
##                  Totals
## Latest:   61,721,361.87
## Dev:               0.97
## Ultimate: 63,455,508.27
## IBNR:      1,734,146.40
## Mack.S.E     136,785.03
## CV(IBNR):          0.08
```

`Latest`: amount already paid

`Ultimate`: estimated total amount (= amount paid + reserve)

`IBNR`: estimated total reserve, IBNR + RBNS!

`Mack.S.E`: estimated standard deviation of the reserve.

GLM chain ladder method

The chain ladder reserve estimate can also be obtained by assuming a Poisson distribution for the incremental runoff triangle with multiplicative mean, i.e.

$$X_{i,j} \sim \texttt{POI}(\alpha_i \cdot \beta_j), \quad 1 \leq i, j \leq l.$$

```r
triangle <- incremental_triangle(observed_yearly,
                                 variable = 'size')

triangle_long <- data.frame(
  occ.year = as.numeric(row(triangle)),
  dev.year = as.numeric(col(triangle)),
  size = as.numeric(triangle))
```

```r
head(triangle_long)
```

```
##   occ.year dev.year    size
## 1        1        1 4075901
## 2        2        1 4037070
## 3        3        1 4322579
## 4        4        1 4162489
## 5        5        1 3903230
## 6        6        1 4052230
```

GLM chain ladder method (continued)

The chain ladder reserve estimate can also be obtained by assuming a Poisson distribution for the incremental runoff triangle with multiplicative mean, i.e.

$$X_{i,j} \sim \texttt{POI}(\alpha_i \cdot \beta_j), \quad 1 \leq i,j \leq l.$$

This Poisson model can be estimated using the `glm` (Generalized Linear Model) routine in R:

$$\log(E(X_{i,j})) = \log(\alpha_i) + \log(\beta_j).$$

```
fit <- glm(size ~ factor(occ.year) + factor(dev.year),
           data = triangle_long,
           family = poisson(link = log))
```

`factor`: Treat the input as a categorical instead of a numeric variable.

```
summary(fit)
```

```
##
## Call:
## glm(formula = size ~ factor(occ.year) + factor(dev.year), family = poisson(link = log
##     data = triangle_long)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -169.664   -55.719    -1.113    42.518   123.909
##
## Coefficients:
##                      Estimate Std. Error   z value Pr(>|z|)
## (Intercept)         1.523e+01  4.220e-04 36079.449   <2e-16 ***
## factor(occ.year)2  -4.926e-02  5.914e-04   -83.281   <2e-16 ***
## factor(occ.year)3   6.812e-02  5.744e-04   118.589   <2e-16 ***
## factor(occ.year)4   6.550e-03  5.832e-04    11.232   <2e-16 ***
## factor(occ.year)5  -2.657e-02  5.881e-04   -45.183   <2e-16 ***
## factor(occ.year)6  -4.040e-04  5.843e-04    -0.691    0.489
## factor(occ.year)7  -5.275e-03  5.851e-04    -9.015   <2e-16 ***
## factor(occ.year)8   3.259e-02  5.799e-04    56.205   <2e-16 ***
## factor(occ.year)9  -5.874e-02  5.943e-04   -98.853   <2e-16 ***
## factor(occ.year)10  1.080e-02  5.873e-04    18.397   <2e-16 ***
## factor(occ.year)11 -1.731e-01  6.845e-04  -252.915   <2e-16 ***
## factor(dev.year)2  -9.390e-01  2.950e-04 -3183.012   <2e-16 ***
## factor(dev.year)3  -3.522e+00  9.734e-04 -3617.713   <2e-16 ***
## factor(dev.year)4  -4.966e+00  2.096e-03 -2369.345   <2e-16 ***
```

GLM chain ladder method

Predict the cells in the lower triangle

```r
lower_triangle <- triangle_long$occ.year +
  triangle_long$dev.year > l + 1

triangle_long$size[lower_triangle] <-
  predict(fit, newdata = triangle_long[lower_triangle, ], type

triangle_long %>%
  pivot_wider(values_from = size,
              names_from = dev.year,
              names_prefix = 'DY.')
```

```
## # A tibble: 11 x 12
##    occ.year    DY.1    DY.2   DY.3  DY.4  DY.5  DY.6  DY.7  DY.8  DY.9     DY.10
##       <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
##  1        1 4075901. 1610076. 112557. 24668. 10680. 8840. 9479. 6217. 3468. 0
##  2        2 4037070. 1461939.  62622. 18518.     0     0     0     0     0  0
##  3        3 4322579. 1774121. 105850. 41429. 24099. 7031.     0     0     0  0.000100
##  4        4 4162489. 1616297. 104487. 15982.     0     0     0     0  1155. 0.0000941
##  5        5 3903230. 1605598. 160906. 24194. 11622.     0     0  1503. 1117. 0.0000910
##  6        6 4052230. 1623940. 131721. 26686. 14096. 6262. 1894. 1543. 1147. 0.0000934
##  7        7 4087125. 1593968.  96484. 40310.  4930. 3668. 1885. 1535. 1142. 0.0000930
##  8        8 4059058. 1771141. 170375. 37317.  9660. 3810. 1957. 1594. 1186. 0.0000965
##  9        9 4033678. 1309634. 140624. 26903.  8817. 3477. 1787. 1455. 1082. 0.0000881
## 10       10 4141278. 1615333. 122300. 28840.  9452. 3727. 1915. 1560. 1160. 0.0000945
## 11       11 3443223. 1346306. 101754. 23995.  7864. 3101. 1594. 1298.  965. 0.0000786
```

Compute the reserve

```r
reserve_glm <- sum(triangle_long$size[lower_triangle])
reserve_glm
```

```
## [1] 1734146
```

Your turn

We have demonstrated the estimation of the reserve (future amount paid). In this exercise you will use the same techniques to estimate the number of future development years with at least one payment. Use the yearly data set `reserving_yearly` for this exercise.

1. Compute the actual number of future payments from the unobserved data set.
2. Create a cumulative triangle containing the number of payments per accident and development year.
3. Estimate the future number of payments with the chain ladder method implemented in the {ChainLadder} package.
4. Compute the difference between the estimated and actual number of payments. Express this error in terms of standard deviations.

For **Q.1** we compute the actual number of future payments from `unobserved_yearly`.

```
payment_actual <- sum(unobserved_yearly$payment)
payment_actual
```

```
## [1] 617
```

For **Q.2** we use `cumulative_triangle` with `variable = 'payment'` to create a payment triangle .

```
triangle <- cumulative_triangle(observed_yearly,
                                variable = 'payment')
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,] 2054 2577 2594 2600 2602 2603 2604 2605 2606  2606  2606
##  [2,] 2074 2556 2572 2576 2576 2576 2576 2576 2576  2576    NA
##  [3,] 2106 2655 2680 2685 2689 2691 2691 2691 2691    NA    NA
##  [4,] 2033 2565 2589 2594 2594 2594 2594 2594   NA    NA    NA
##  [5,] 2008 2481 2513 2520 2522 2522 2522   NA   NA    NA    NA
##  [6,] 2035 2533 2559 2563 2565 2567   NA   NA   NA    NA    NA
##  [7,] 2028 2515 2532 2537 2538   NA   NA   NA   NA    NA    NA
##  [8,] 2030 2558 2592 2601   NA   NA   NA   NA   NA    NA    NA
##  [9,] 2005 2449 2473   NA   NA   NA   NA   NA   NA    NA    NA
## [10,] 2100 2607   NA   NA   NA   NA   NA   NA   NA    NA    NA
## [11,] 1784   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
```

For **Q.3** we appy the chain ladder method to this payment triangle.

```
require(ChainLadder)
cl <- MackChainLadder(triangle)
cl
```

```
## MackChainLadder(Triangle = triangle)
##
##    Latest Dev.To.Date Ultimate    IBNR Mack.S.E CV(IBNR)
## 1   2,606       1.000    2,606   0.000    0.000      NaN
## 2   2,576       1.000    2,576   0.000    0.121      Inf
## 3   2,691       1.000    2,691   0.000    0.125      Inf
## 4   2,594       1.000    2,594   0.330    0.677   2.0532
## 5   2,522       1.000    2,523   0.561    0.862   1.5354
## 6   2,567       1.000    2,568   0.769    0.998   1.2973
## 7   2,538       0.999    2,540   1.577    1.435   0.9101
## 8   2,601       0.999    2,604   3.200    2.069   0.6464
## 9   2,473       0.997    2,481   8.443    2.673   0.3166
## 10  2,607       0.987    2,640  33.472    7.535   0.2251
## 11  1,784       0.793    2,250 466.225   28.387   0.0609
##
##                Totals
## Latest:   27,559.00
## Dev:           0.98
## Ultimate: 28,073.58
## IBNR:        514.58
## Mack.S.E      30.08
## CV(IBNR):      0.06
```

For **Q.4** we compute the prediction error of the chain ladder method.

```r
ultimate <- sum(cum2incr(cl$FullTriangle))
already_paid <- sum(cum2incr(cl$Triangle), na.rm = TRUE)

payment_cl <-  ultimate - already_paid
sigma_cl <- as.numeric(cl$Total.Mack.S.E)

error = payment_actual - payment_cl

round(c(error = error,
        pct_error = error / payment_actual * 100,
        std.dev = error / sigma_cl),2)
```

```
##      error pct_error   std.dev
##     102.42     16.60      3.41
```

For our data set, the chain ladder method significantly underestimates the number of future payments.

Actual triangle:

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,] 2054  523   17    6    2    1    1    1    1     0     0
##  [2,] 2074  482   16    4    0    0    0    0    0     0     0
##  [3,] 2106  549   25    5    4    2    0    0    0     0     0
##  [4,] 2033  532   24    5    0    0    0    0    0     0     0
##  [5,] 2008  473   32    7    2    0    0    0    0     0     0
##  [6,] 2035  498   26    4    2    2    1    0    0     0     0
##  [7,] 2028  487   17    5    1    1    0    0    0     0     0
##  [8,] 2030  528   34    9    3    1    0    0    0     0     0
##  [9,] 2005  444   24    6    2    0    0    0    0     0     0
## [10,] 2100  507   19    5    3    1    0    0    0     0     0
## [11,] 1784  524   39    8    3    1    0    0    0     0     0
```

Predicted triangle:

```
##       dev
## origin    1     2        3        4        5         6         7         8         9
##      1  2054 523.0 17.00000 6.000000 2.000000 1.0000000 1.0000000 1.0000000 1.0000000
##      2  2074 482.0 16.00000 4.000000 0.000000 0.0000000 0.0000000 0.0000000 0.0000000
##      3  2106 549.0 25.00000 5.000000 4.000000 2.0000000 0.0000000 0.0000000 0.0000000
##      4  2033 532.0 24.00000 5.000000 0.000000 0.0000000 0.0000000 0.0000000 0.3295224
##      5  2008 473.0 32.00000 7.000000 2.000000 0.0000000 0.0000000 0.2409938 0.3204066
##      6  2035 498.0 26.00000 4.000000 2.000000 2.0000000 0.1976744 0.2453127 0.3261488
##      7  2028 487.0 17.00000 5.000000 1.000000 0.8161821 0.1955041 0.2426194 0.3225679
##      8  2030 528.0 34.00000 9.000000 1.582905 0.8369510 0.2004790 0.2487932 0.3307761
##      9  2005 444.0 24.00000 5.394067 1.508290 0.7974988 0.1910288 0.2370655 0.3151839
##      10 2100 507.0 24.48796 5.739759 1.604952 0.8486084 0.2032713 0.2522584 0.3353832
##      11 1784 437.7 20.86878 4.891454 1.367749 0.7231888 0.1732290 0.2149761 0.2858155
```

When the chain ladder method fails

Detecting when the chain ladder method fails

The chain ladder method assumes that claims from all occurrence years follow the **same development pattern**.
We construct various triangles to assess this assumption.

Number of open claims at the start of the year:

```r
triangle_open <- incremental_triangle(
  observed_yearly %>%
    mutate(open = calendar_year <= settlement_year),
  variable = 'open')
triangle_open
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,] 2743  722   28    6    2    1    1    1    1     0     0
##  [2,] 2725  698   24    4    1    0    0    0    0     0    NA
##  [3,] 2810  789   35    5    5    2    1    0    0    NA    NA
##  [4,] 2756  743   33    6    0    0    0    0   NA    NA    NA
##  [5,] 2705  677   46    9    3    0    0   NA   NA    NA    NA
##  [6,] 2706  689   34    8    2    2   NA   NA   NA    NA    NA
##  [7,] 2703  706   27    7    1   NA   NA   NA   NA    NA    NA
##  [8,] 2708  730   45   10   NA   NA   NA   NA   NA    NA    NA
##  [9,] 2640  647   27   NA   NA   NA   NA   NA   NA    NA    NA
## [10,] 2751  711   NA   NA   NA   NA   NA   NA   NA    NA    NA
## [11,] 2328   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
```

Number of open claims at the end of the year:

```r
triangle_open_end <- incremental_triangle(
  observed_yearly %>%
    mutate(open_end = (calendar_year < settlement_year)),
  variable = 'open_end')
triangle_open_end
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,]  722   28    6    2    1    1    1    1    0     0     0
##  [2,]  698   24    4    1    0    0    0    0    0     0    NA
##  [3,]  789   35    5    5    2    1    0    0    0    NA    NA
##  [4,]  743   33    6    0    0    0    0    0   NA    NA    NA
##  [5,]  677   46    9    3    0    0    0   NA   NA    NA    NA
##  [6,]  689   34    8    2    2    1   NA   NA   NA    NA    NA
##  [7,]  706   27    7    1    1   NA   NA   NA   NA    NA    NA
##  [8,]  730   45   10    4   NA   NA   NA   NA   NA    NA    NA
##  [9,]  647   27    7   NA   NA   NA   NA   NA   NA    NA    NA
## [10,]  711   30   NA   NA   NA   NA   NA   NA   NA    NA    NA
## [11,]  644   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
```

Less claims occurred during the last accident year, but of those that occurred many are still open.

Detecting when the chain ladder method fails

The chain ladder method assumes that claims from all occurrence years follow the **same development pattern**.
We construct various triangles to assess this assumption:

Percentage of claims open at the start of the year for which a payment is made:

```
triangle_payment <- incremental_triangle(
  observed_yearly,
  variable = 'payment')

triangle_payment / triangle_open
```

```
##             [,1]      [,2]      [,3]      [,4]      [,5] [,6] [,7] [,8] [,9] [,10] [,1
##  [1,] 0.7488152 0.7243767 0.6071429 1.0000000 1.0000000    1    1    1    1   NaN   N
##  [2,] 0.7611009 0.6905444 0.6666667 1.0000000 0.0000000  NaN  NaN  NaN  NaN   NaN
##  [3,] 0.7494662 0.6958175 0.7142857 1.0000000 0.8000000    1    0  NaN  NaN    NA
##  [4,] 0.7376633 0.7160162 0.7272727 0.8333333       NaN  NaN  NaN  NaN   NA    NA
##  [5,] 0.7423290 0.6986706 0.6956522 0.7777778 0.6666667  NaN  NaN   NA   NA    NA
##  [6,] 0.7520325 0.7227866 0.7647059 0.5000000 1.0000000    1   NA   NA   NA    NA
##  [7,] 0.7502775 0.6898017 0.6296296 0.7142857 1.0000000   NA   NA   NA   NA    NA
##  [8,] 0.7496307 0.7232877 0.7555556 0.9000000        NA   NA   NA   NA   NA    NA
##  [9,] 0.7594697 0.6862442 0.8888889        NA        NA   NA   NA   NA   NA    NA
## [10,] 0.7633588 0.7130802        NA        NA        NA   NA   NA   NA   NA    NA
## [11,] 0.7663230        NA        NA        NA        NA   NA   NA   NA   NA    NA
```

Average amount paid given a payment is made:

```
triangle_size <- incremental_triangle(
  observed_yearly,
  variable = 'size')

triangle_size / triangle_payment
```

```
##            [,1]     [,2]     [,3]     [,4]     [,5]     [,6]    [,7]     [,8]     [,9]
##  [1,] 1984.372 3078.539 6620.975 4111.405 5339.773 8840.487 9478.77 6217.266 3468.364
##  [2,] 1946.514 3033.069 3913.889 4629.451      NaN      NaN     NaN      NaN      NaN
##  [3,] 2052.507 3231.551 4234.001 8285.841 6024.656 3515.394     NaN      NaN      NaN
##  [4,] 2047.462 3038.153 4353.609 3196.478      NaN      NaN     NaN      NaN       NA
##  [5,] 1943.840 3394.499 5028.299 3456.332 5811.065      NaN     NaN       NA       NA
##  [6,] 1991.268 3260.923 5066.209 6671.495 7047.903 3130.870      NA       NA       NA
##  [7,] 2015.347 3273.035 5675.517 8062.022 4930.009       NA      NA       NA       NA
##  [8,] 1999.536 3354.434 5011.021 4146.339       NA       NA      NA       NA       NA
##  [9,] 2011.809 2949.626 5859.350       NA       NA       NA      NA       NA       NA
## [10,] 1972.037 3186.061       NA       NA       NA       NA      NA       NA       NA
## [11,] 1930.058       NA       NA       NA       NA       NA      NA       NA       NA
```

Detecting when the chain ladder method fails

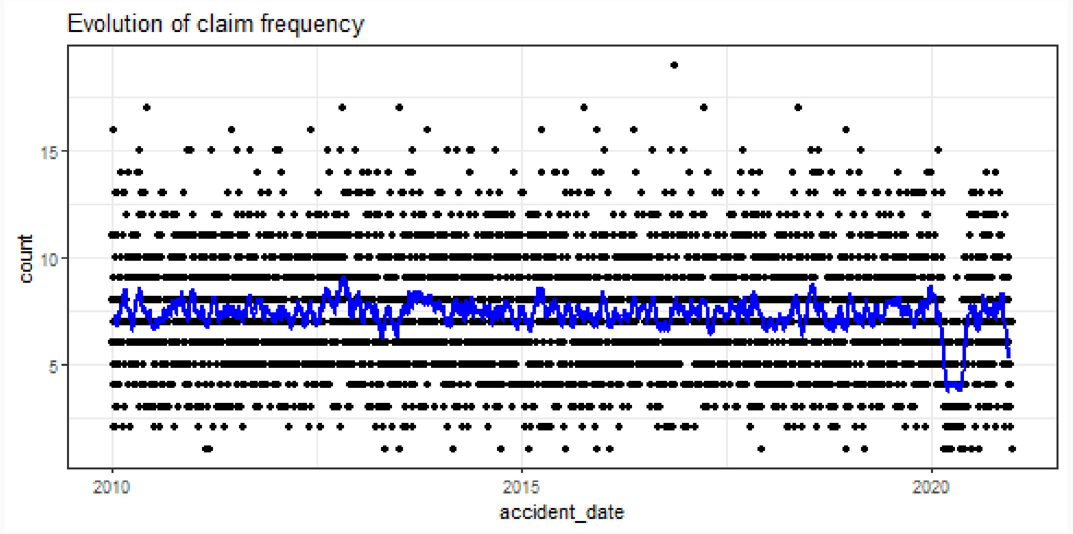
The chain ladder method assumes that claims from all occurrence years follow the **same development pattern**.
We inspect the (granular) daily claims data:

```r
# Select one observation per claim
claims <- observed_daily %>%
  group_by(accident_number) %>%
  slice(1) %>%
  ungroup()


occ_intensity <- claims %>%
  group_by(accident_date) %>%
  summarise(count = n())

require(zoo)
occ_intensity$moving_average <-
  rollmean(occ_intensity$count, 30, na.pad = TRUE)

ggplot(occ_intensity) +
  theme_bw() +
  geom_point(aes(x = accident_date, y = count)) +
  geom_line(aes(x = accident_date, y = moving_average),
            size = 1, color = 'blue') +
  ggtitle('Evolution of claim frequency')
```



The moving average indicates a period with decreased claim occurrence intensity in 2020.

Detecting when the chain ladder method fails

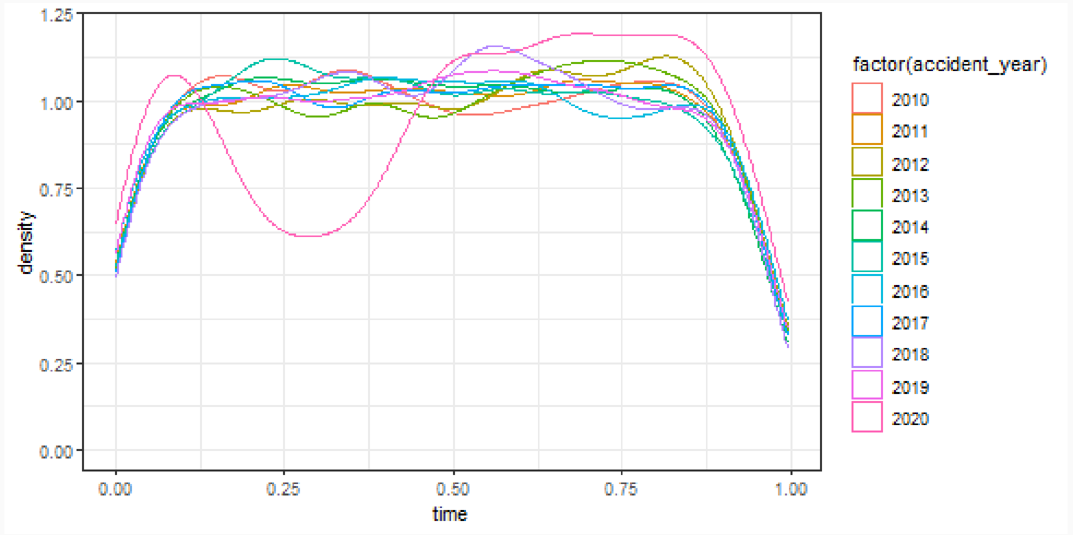
The chain ladder method assumes that claims from all occurrence years follow the **same development pattern**.
We inspect the (granular) daily claims data:

```
require(lubridate)
claims <- claims %>%
  mutate(start_year = floor_date(accident_date, unit = 'year'),
         time = as.numeric(accident_date - start_year) / 366,
         accident_year = year(accident_date),
         reporting_year = year(reporting_date)) %>%
  filter(accident_year == reporting_year)

ggplot(claims) +
  theme_bw() +
  geom_density(aes(x = time,
                   group = factor(accident_year),
                   color = factor(accident_year)))
```

We apply `filter(accident_year == reporting_year)` such that the same amount of information is available for all occurrence years.

Density showing claim occurrences within each accident year. The chain ladder method works best when the densities look similar across accident years.



The occurrence intensity in accident year 2020 deviates from the other accident years!

Detecting when the chain ladder method fails

The data contains a drop in claim frequency during March, April, May 2020, with **less claims occurring** due to the first **COVID-19 lockdown**. Such changes in the claim occurrence and development pattern can severly impact chain ladder performance.

Other situations that might impact the performance of the chain ladder are:

- inflation:
  - to be detected via the triangle `size`
- changing product mix:
  - to be detected via the triangles `close`, `payment` and `size`
- internal changes in claim handling:
  - to be detected via the triangles `close` and `payment`
- extreme weather events:
  - to be detected via the yearly distribution of occurrence dates
- an extremely large claim:
  - to be detected via the triangle `size`.

Fixing the chain ladder method

Making the **data more homogeneous** using:

- **group** claims by:
  - claim characteristics, e.g. a separate triangle for fire and water related claims in home insurance.
  - occurrence time within a calendar year
- **omit**:
  - deviating years/cells, especially if they occurred long ago
- **adjust** for:
  - inflation.

The appropriate method(s) depend on the data inspection from the previous slides.

Applied to the **COVID-19 reserving data set**:

- claim frequency has changed in the months March, April, May
- only 2020, i.e. accident year 10, is affected.

As 2020 is a recent calendar year, we cannot omit accident year 10 from the data.

We illustrate two approaches to improve the homogeneity across the accident years:

- applying the chain ladder method to more granular data
- grouping claims by occurrence month.

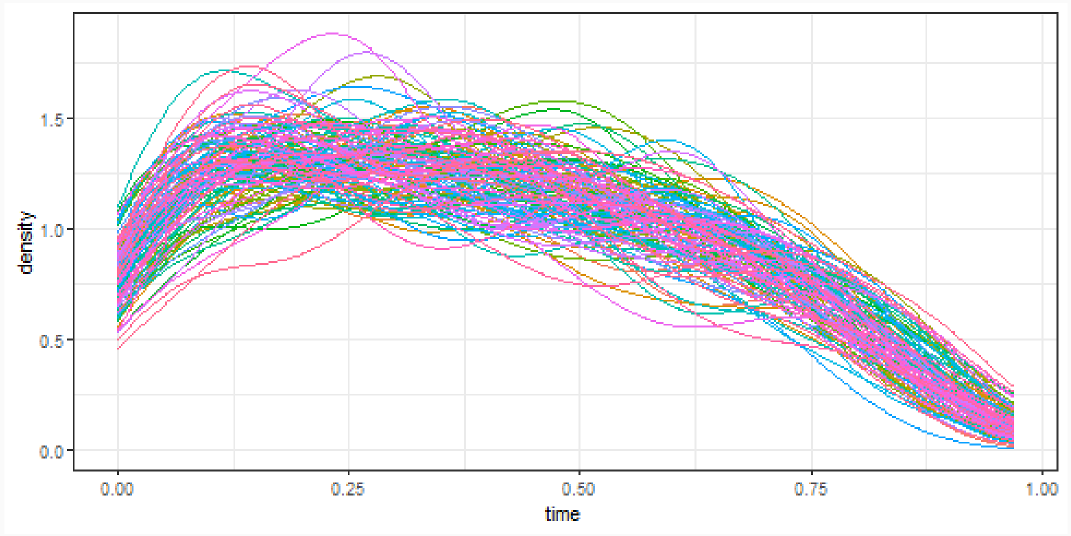
Monthly chain ladder

A more **granular chain ladder** offers more flexibility/parameters to capture fluctuations in the **fluctuations in the occurrence process**.

```r
require(lubridate)
claims <- observed_daily %>%
  group_by(accident_number) %>%
  slice(1) %>%
  ungroup() %>%
  mutate(start_month = floor_date(accident_date, unit = 'month
         time = as.numeric(accident_date - start_month) / 31,
         accident_month = format(accident_date, '%Y%m'),
         reporting_month = format(reporting_date, '%Y%m')) %>%
  filter(accident_month == reporting_month)

ggplot(claims) +
  theme_bw() +
  geom_density(aes(x = time,
                   group = factor(accident_month),
                   color = factor(accident_month))) +
  theme(legend.position = 'none')
```

We compare the distribution of claim occurrences within each calendar month.



In real (non simulated) data, these monthly densities might be disturbed by seasonal effects, weekends and holidays.

Monthly chain ladder

Example code to construct a **monthly triangle** using the observed daily data:

```r
triangle_month <- observed_daily %>%
  mutate(accident_month = year(accident_date)*12 + month(accident_date) - 2010*12,
         development_month = year(payment_date)*12 + month(payment_date) - 2010*12 - accident_month) %>%
  group_by(accident_month, development_month) %>%
  summarise(size = sum(payment_size)) %>%
  ungroup() %>%
  complete(expand.grid(accident_month = 1:132, development_month = 0:131), fill = list(size = 0)) %>%
  mutate(size = ifelse(accident_month + development_month > 132, NA, size)) %>%
  arrange(development_month) %>%
  pivot_wider(names_from = development_month, values_from = size) %>%
  arrange(accident_month)

triangle_month <- as.matrix(triangle_month[, 2:132])
```

Monthly chain ladder

```
cl <- MackChainLadder(incr2cum(triangle_month))
summary(cl)$Totals
```

```
##                  Totals
## Latest:    6.172136e+07
## Dev:       9.697826e-01
## Ultimate:  6.364453e+07
## IBNR:      1.923173e+06
## Mack S.E.: 1.035489e+05
## CV(IBNR):  5.384277e-02
```

Comparing the yearly and monthly chain ladder:

```
##                 method cells reserve.estimate     error pct.error
## 1  Yearly chain ladder    66          1734146 -415165.1 -19.31619
## 2 Monthly chain ladder  8778          1923173 -226139.0 -10.52146
```

The monthly chain ladder predicts the reserve more accurately on this data set.

Grouping claims by occurrence month

We group claims based on the occurrence month. For each of these groups we construct a **separate runoff triangle** and estimate the reserve using a **yearly chain ladder**.

```r
require(lubridate)
claims <- observed_daily %>%
  group_by(accident_number) %>%
  slice(1) %>%
  ungroup() %>%
  mutate(start_month = floor_date(accident_date, unit = 'month
         time = as.numeric(accident_date - start_month) / 31,
         accident_year = format(accident_date, '%Y'),
         reporting_year = format(reporting_date, '%Y'),
         month = format(accident_date, '%B')) %>%
  filter(accident_year == reporting_year)

ggplot(claims) +
  facet_wrap( ~ month, ncol = 3) +
  theme_bw() +
  geom_density(aes(x = time,
                   group = factor(accident_year),
                   color = factor(accident_year)))
```

We are comparing occurrences in the same month accross different years. We are no longer comparing different

Grouping claims by occurrence month

Add the accident date to the data set `reserving_yearly`

```
reserving_yearly <- reserving_yearly %>%
  left_join(reserving_daily %>%
              group_by(accident_number) %>%
              slice(1) %>%
              ungroup() %>%
              select(accident_number, accident_date))

reserving_yearly <- reserving_yearly %>%
  mutate(accident_month = format(accident_date, '%B'))
```

Create a runoff triangle for each accident month

```
triangles <- reserving_yearly %>%
  group_by(accident_month, accident_year, development_year) %>
  summarise(size = sum(size)) %>%
  ungroup() %>%
  complete(expand.grid(accident_month = unique(accident_month)
                       accident_year = 0:10, development_year
           fill = list(size = 0)) %>%
  mutate(size = ifelse(accident_year + development_year > 11,
```

```
triangles %>%
  filter(accident_month == 'april') %>%
  arrange(development_year) %>%
  pivot_wider(names_from = development_year, values_from = siz
```

```
## # A tibble: 11 x 13
##    accident_month accident_year    DY.1   DY.2   DY.3  DY.4  DY.5  DY.6  DY.7  DY.8
##    <chr>                  <dbl>   <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <
##  1 april                      0 406625. 70060.  8968. 4503. 2032.     0     0     0
##  2 april                      1 392370. 39237.     0     0     0      0     0     0
##  3 april                      2 490615. 62149.     0     0     0      0     0     0
##  4 april                      3 392533. 30396.  3977.    0     0      0     0     0
##  5 april                      4 426086. 40524. 19473. 5444. 5613.     0     0    NA
##  6 april                      5 423923. 43166.  3460.    0     0      0    NA    NA
##  7 april                      6 411784. 23560.  9287.    0     0     NA    NA    NA
##  8 april                      7 393665. 36667.     0     0    NA     NA    NA    NA
##  9 april                      8 394125. 30951.     0    NA    NA     NA    NA    NA
## 10 april                      9 388587. 24324.    NA    NA    NA     NA    NA    NA
## 11 april                     10 202582.    NA     NA    NA    NA     NA    NA    NA
```

Grouping claims by occurrence month (continued)

We use the `glm` routine to apply the chain ladder method to each triangle:

`factor(development_year) * accident_month` estimates a different effect `factor(development)` for each `accident_month`

```
fit <- glm(size ~ factor(development_year) * accident_month  + factor(accident_year) * accident_month,
           data = triangles,
           family = poisson(link = 'log'))

reserve_group <- sum(predict(fit, newdata = triangles %>% filter(is.na(size)), type = 'response'))
reserve_group
```

```
## [1] 1928050
```

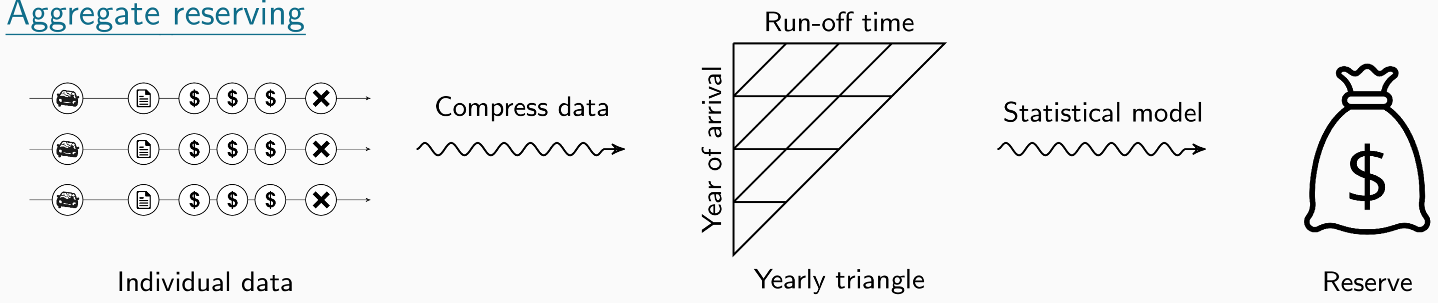
We obtain a reserve estimate similar to the one obtained with the monthly chain ladder

```
##                method cells reserve.estimate     error pct.error
## 1  Yearly chain ladder    66          1734146 -415165.1 -19.31619
## 2 Monthly chain ladder  8778          1923173 -226139.0 -10.52146
## 3 Grouped chain ladder   792          1928050 -221261.9 -10.29455
```

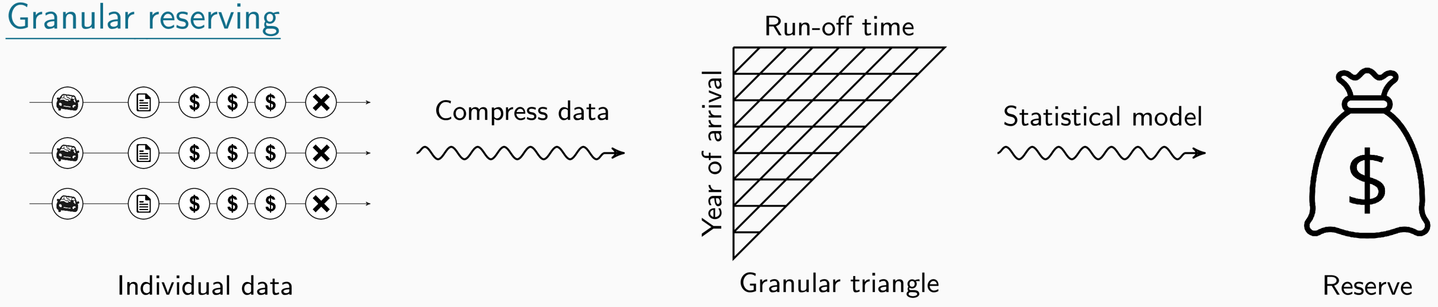
Research outlook

Research outlook

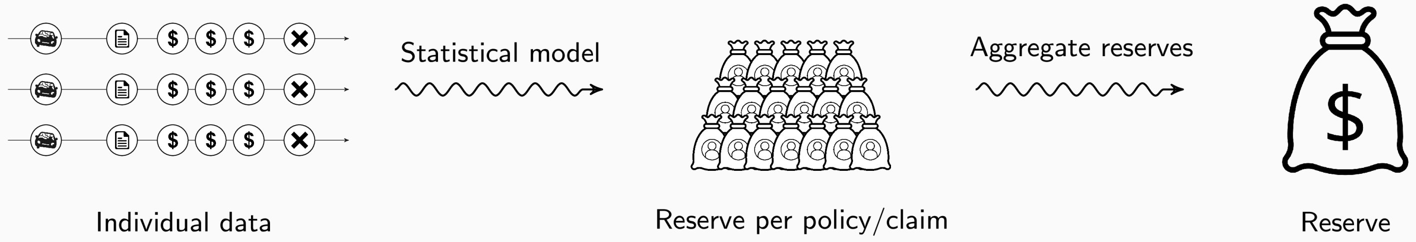
Aggregate reserving



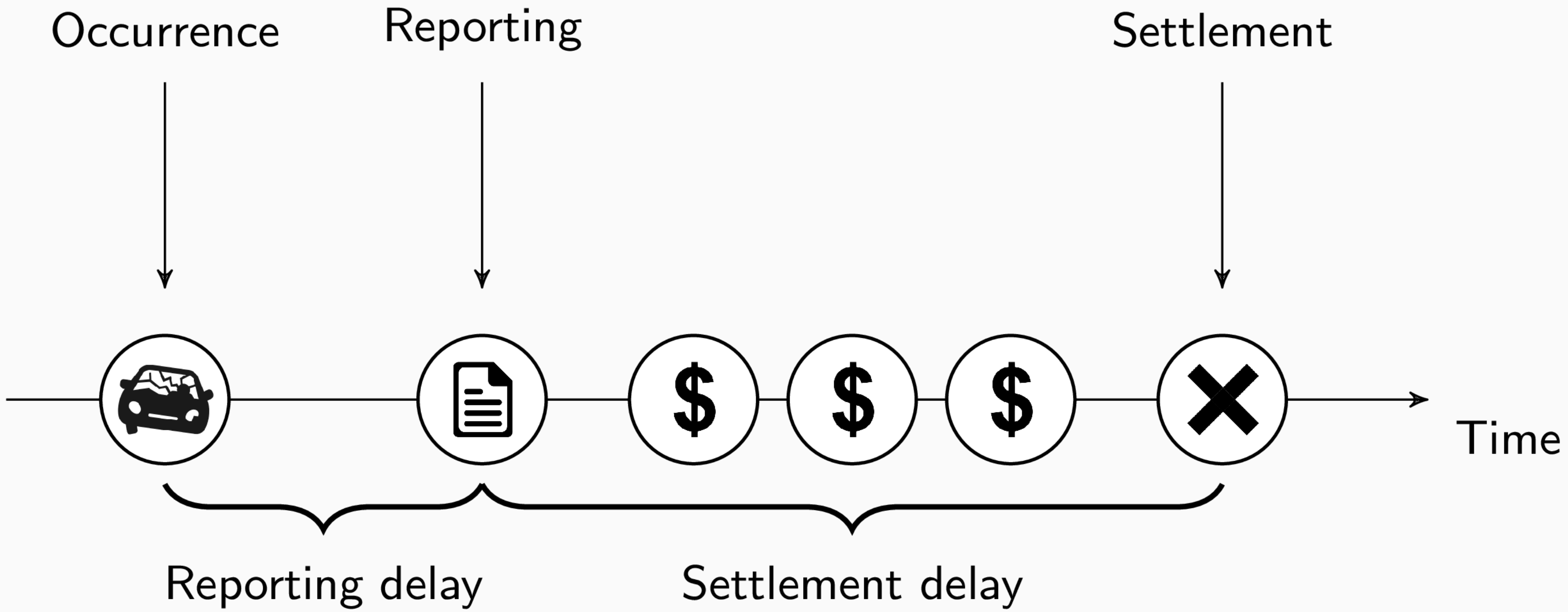
Granular reserving



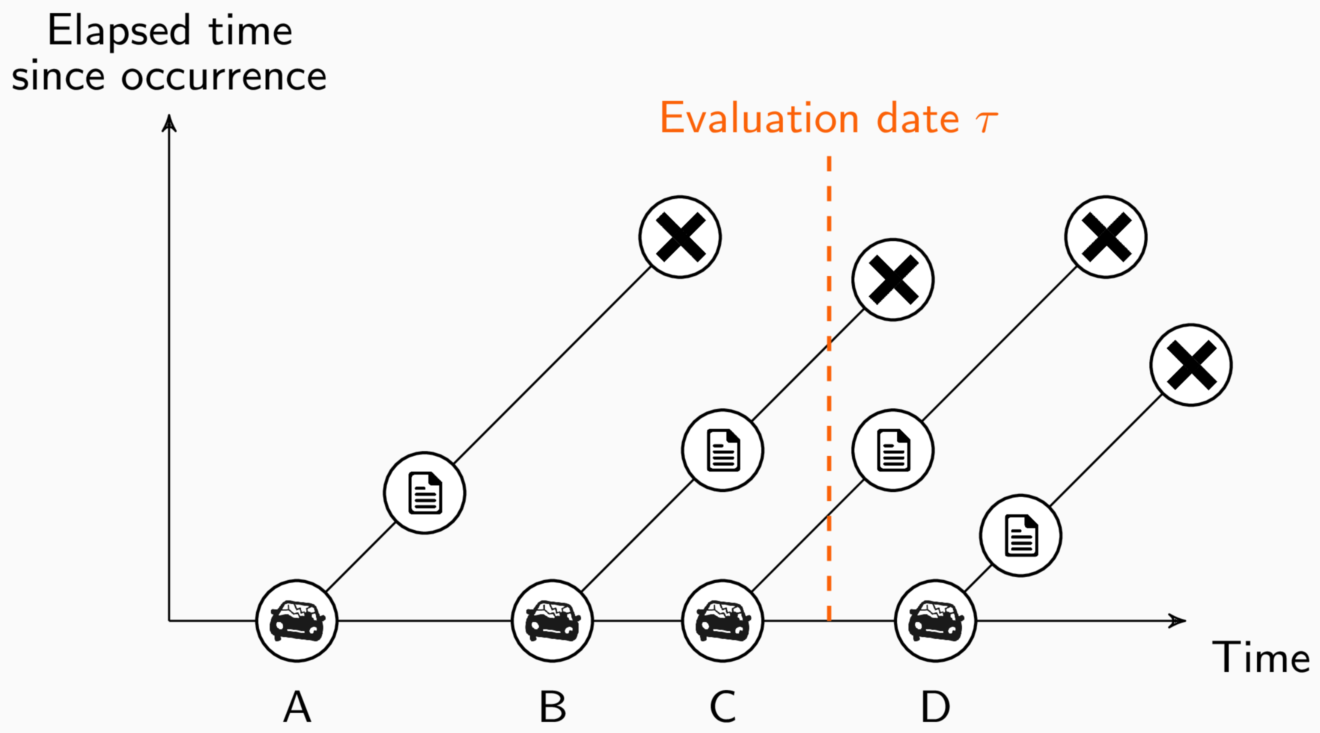
Individual reserving



Development of an individual claim



Development of an individual claim (continued)



Observed claims are **censored** due to **delays** (reporting, settlement) in the claim development process.

IBNR: future costs for claims that occurred, but are not yet reported (claim B)

RBNS: future costs for claims that are reported, but are not yet settled (claim C)

Pricing: all costs for claims that will occur in **future** insured **exposure periods** (claim D).

The IBNR reserve

Following a frequency-severity decomposition, the IBNR reserve is the product of the expected **number of unreported claims** times the **expected severity per claim**. Hereby:

- insurance pricing covers the estimation of claim severity for new claims
- reserving methods focus on predicting the **number of unreported claims**.

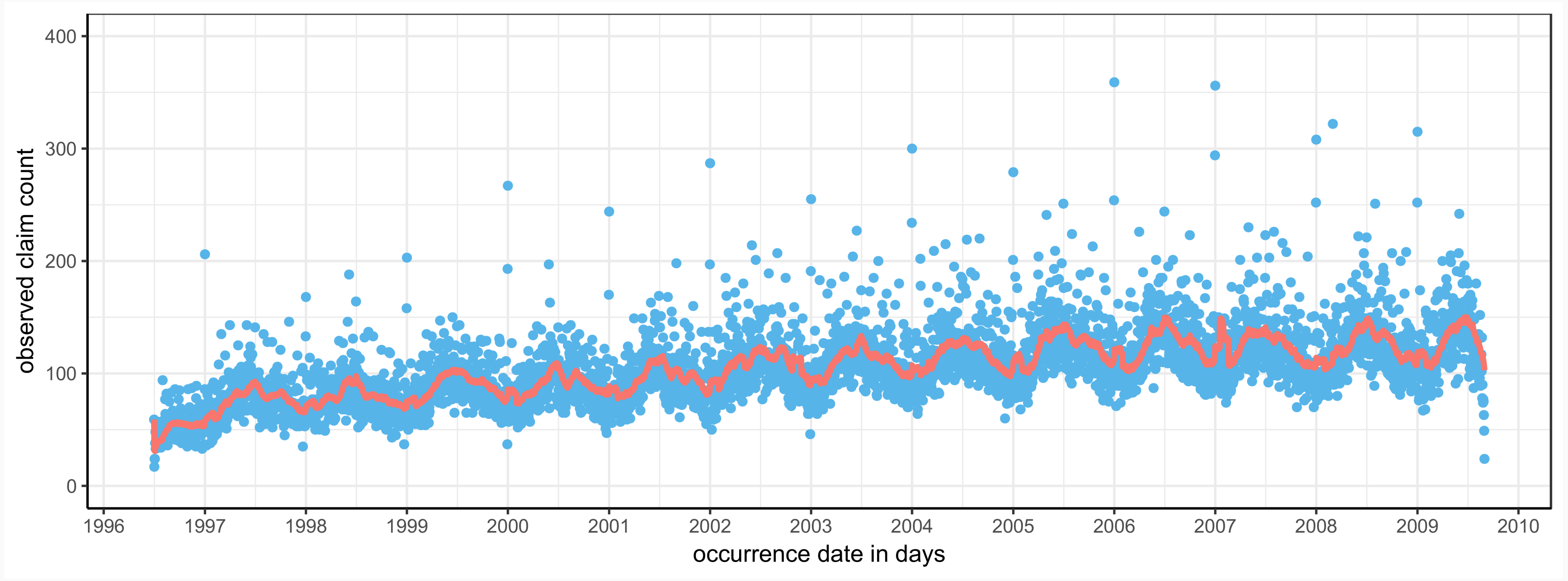
Our work on the topic:

- "Modeling the number of hidden events subject to observation delay". Jonas Crevecoeur, Katrien Antonio and Roel Verbelen. (2019). European Journal of Operational Research.

- "Modeling the occurrence of events subject to a reporting delay via an EM algorithm". Roel Verbelen, Katrien Antonio, Gerda Claeskens and Jonas Crevecoeur. (2021). Statistical Science.

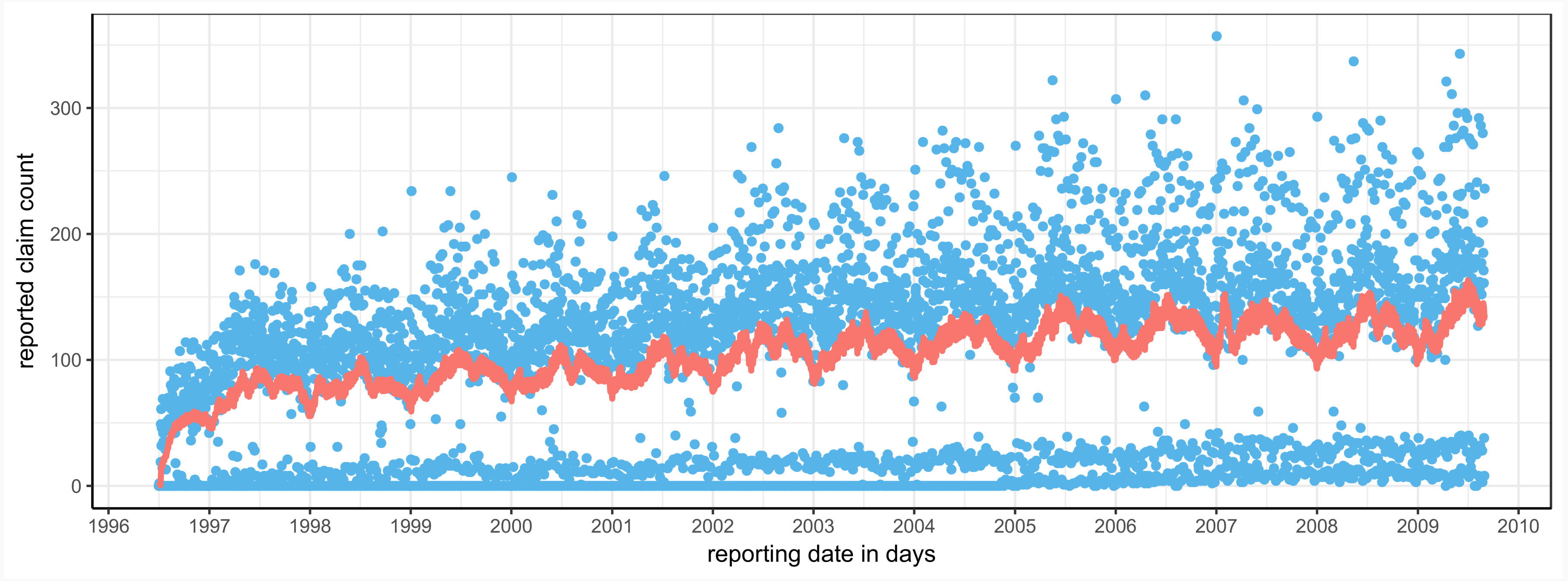
The IBNR reserve (continued)

Number of observed claim occurrences per day



The IBNR reserve (continued)

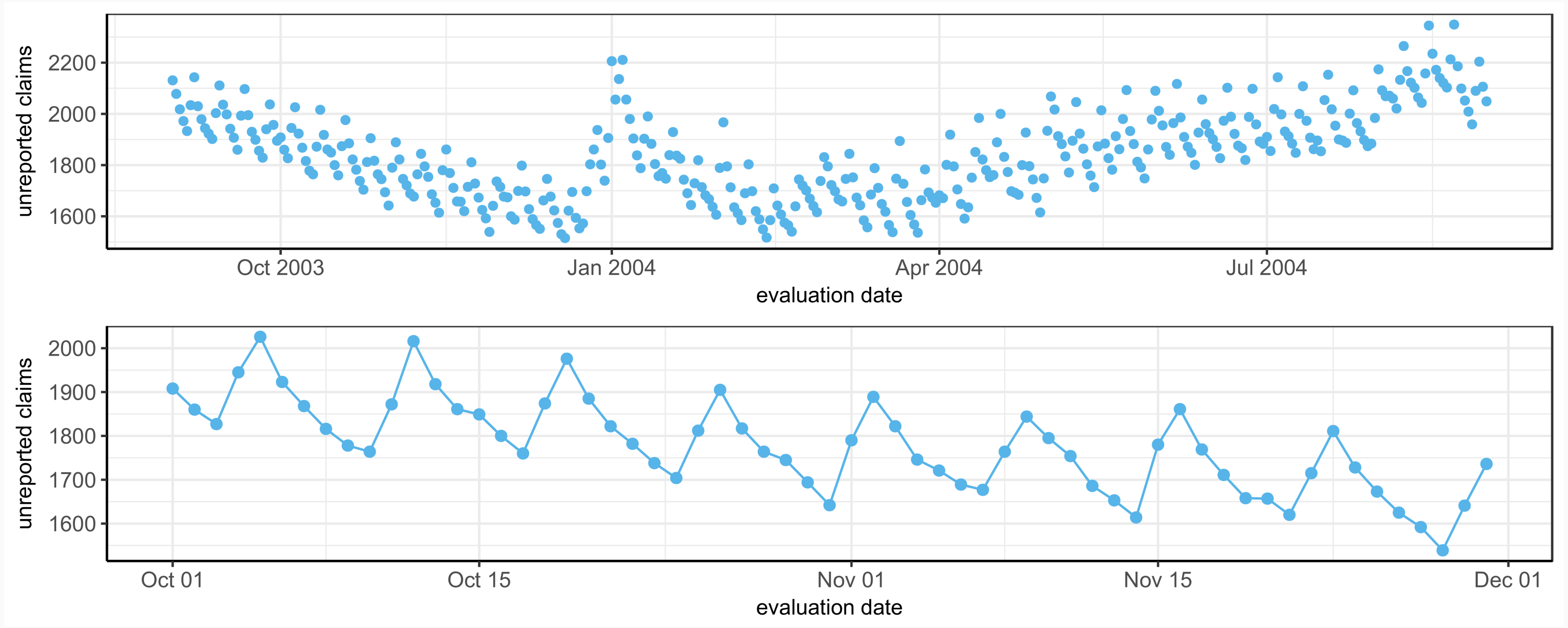
Number of reported claims per day



Almost no claims are reported on Saturdays, Sundays and holidays.

The IBNR reserve (continued)

Total number of unreported claims on each day, i.e. the number of claims that occur before the evaluation date, but are reported afterwards.



The number of unreported claims increases during the weekend (+10% on sunday) and around the end of the year (+30%).

The RBNS reserve

Predict the future costs of individual, reported claims.

Our work on the topic:

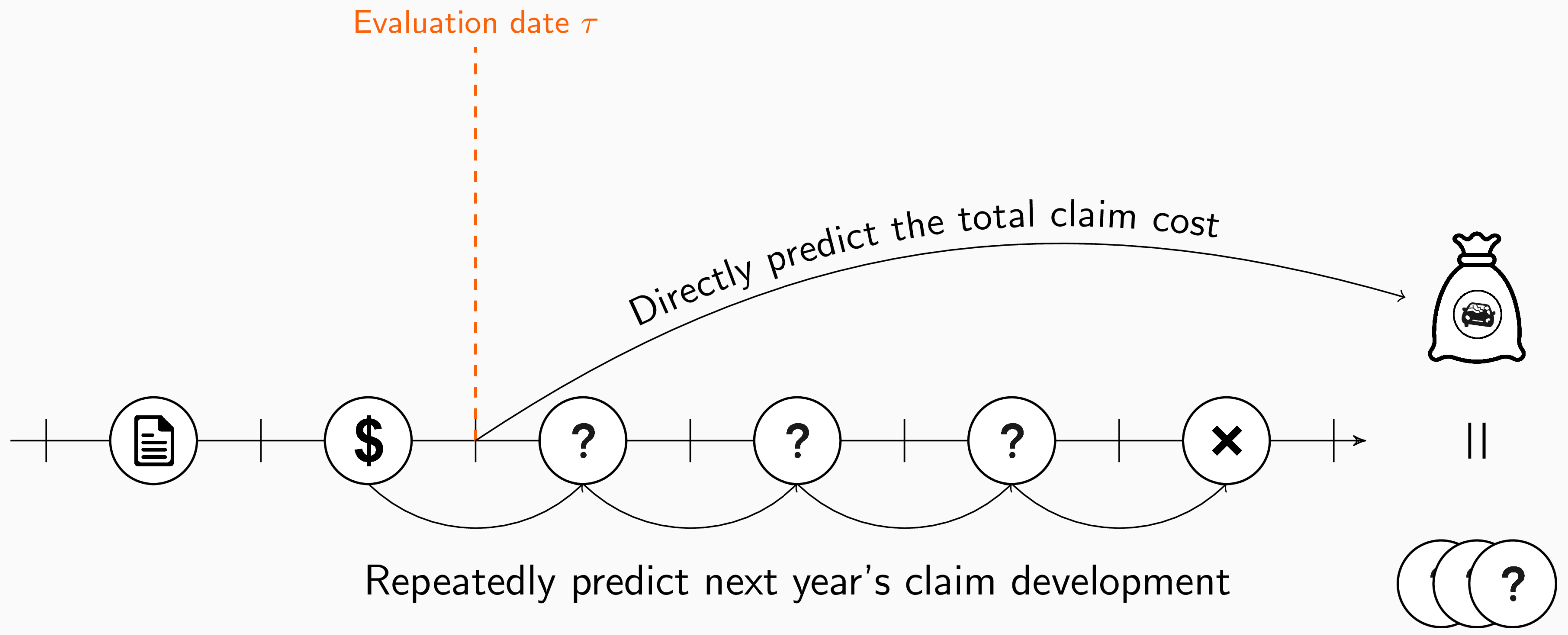
- "A hierarchical reserving model for non-life insurance claims". Jonas Crevecoeur and Katrien Antonio. (2021). Working paper, under review.
- "Bridging the gap between pricing and reserving with an occurrence and development model for non-life insurance claims". Katrien Antonio and Jonas Crevecoeur. (2021). Working paper.

Methods implemented in the R package {hirem}, covering tools for hierarchical reserving models, available on GitHub.

The RBNS reserve (continued)

The hierarchical model is based on **two key ideas**:

1. Model the development of indiviudal claim in discrete time (steps of one year) using the observed development in previous years.


Evaluation date τ
Directly predict the total claim cost
$
?
?
?
=
?
Repeatedly predict next year's claim development

The RBNS reserve (continued)

The hierarchical model is based on **two key ideas**:

1. Model the development of indiviudal claim in discrete time (steps of one year) using the observed development in previous years.
2. Focus on all events registered over the lifetime of a claim

Common events registered over the lifetime of a claim:

- settlement
- payments
- initial incurred / changes in the incurred
- involvement lawyer.

These events are **dependent**:

- if a claim is **settled**, there will be no **payments** in the future
- large **payments** are more likely when the outstanding **reserve** is large.

Implementation in the {hirem} package

Implementation readily available from the {hirem} package on GitHub.
Events are added to the model as **layers**.

```r
require(hirem)

individual_data <- individual_data %>%
  mutate(calendar_year = accident_year + development_year - 1,
         close = (settlement_year == calendar_year))

model <- hirem(individual_data %>% filter(calendar_year <= settlement_year)) %>%
  layer_glm('close', binomial(link = logit)) %>%
  layer_glm('payment', binomial(link = logit)) %>%
  layer_glm('size', Gamma(link = log),
            filter = function(data){data$payment == 1})

model <- fit(model,
             close = 'close ~ factor(development_year)',
             payment = 'payment ~ close + factor(development_year)',
             size = 'size ~ close + factor(development_year)')
```

Thanks!

Slides created with the R package xaringan.

Course material available via

https://github.com/katrienantonio/workshop-loss-reserv-fraud