Collective risk

Ignacio Gómez, Gonzalo Pato, Gonzalo Prats

April 2023

Contents

Statement

The clients of an insurance company generate claims according to independent (homogeneous) Poisson processes with common rate λ. The claims are independent and the amount of each of them follows a distribution with cdf $F(x) = 1 - (100/x)^{2.5}$ if $x > 100$. New clients enroll the company according to another (homogeneous) Poisson process with rate ν and the time they stay in the company follows an exponential distribution with rate μ. Each policyholder pays a fixed amount a every year to the company. If the policyholder stays in the company only during a fraction of the year, she will pay the corresponding fraction of a which is assumed to be paid on a continuous manner.

If the initial capital is c_0 and the initial number of clients is n_0, we want to compute the probability that the capital of the company remains positive during some given time t_l.

Introduction

Each policyholder has a fixed amount a to pay every year, so the capital of the company will increase by a times the number of clients every fraction of year.

On the other hand, they also hold the right to claim the premium on the policy, which happens according to a Poisson process with common rate λ. The amount of is this premium is distributed according to a Pareto distribution with parameters $\alpha = 2.5$ and $\beta = 100$.

The clients of the company can leave the company at any time, and the time they stay in the company follows an exponential distribution with rate μ.

Clients come according to an independent (homogeneous) Poisson processes with rate ν, and they stay according to an exponentially distributed time with rate μ.

So in general, the capital of the company at any time t will be:

$$C(t) = c_0 + at(n_0 + N_A(t) - N_D(t)) - \sum_{j=1}^{N_C(t)} X_j$$

where $N_A(t)$ is the number of clients that arrive by time t, $N_D(t)$ is the number of clients that leave by time t, $N_C(t)$ is the number of claims that arrive by time t, X_j is the amount of the j-th claim, and $n(t)$ is the number of clients at time t.

The project

Project development including the code and emphasizing the main difficulties and most important parts.

First, to simulate the amount of the claims we will define a function with the inverse of the cdf of the Pareto distribution. Mathematically, this function is:

$$F^{-1}(x) = \frac{\beta}{\sqrt[\alpha]{(1-x)}} = \frac{100}{\sqrt[2.5]{(1-x)}}$$

```
inverse.claim <- function(x) {
  return(max((100 / (1 - x)^(1 / 2.5)), 0))
}
```

The algorithm we will follow is, for each path:

1. Initialize variables $t = 0, n = n_0, c = c_0$.
2. Compute the time of the first event: $dt = rexp(1, \lambda * n + \mu * n + \nu)$
3. Update the time: $t = t + dt$
4. While $t < tl$:
   4.1. Update capital with the proportional part of the payments: $c = c + n * a * dt$
   4.2. Decide the type of event and update in each case:
      4.2.1. If $u < \lambda * n/(\lambda * n + \mu * n + \nu)$, claim event so update capital $= c - X_j$
      4.2.2. If $u < (\lambda * n + \mu * n)/(\lambda * n + \mu * n + \nu)$, departure event so $n = n - 1$
      4.2.3. Else, enrollment event, so $n = n + 1$
   4.3. Compute the time of the next event: $dt = rexp(1, \lambda * n + \mu * n + \nu)$
   4.4. Update the time: $t = t + dt$

Thus for one simulation we have:

```
simulate_one_insurance <- function(c0, n0, a, tl, lambda, mu, nu) {
  # Initialize variables
  t <- 0
  n <- n0
  c <- c0
  # Update capital with annuity payments
  dt <- rexp(1, lambda * n + mu * n + nu)
  t <- t + dt
  while (t < tl) {
    # Update the capital with the proportional part of the payments
```

```r
    c <- c + n * a * dt
    # Decide the type of event
    u <- runif(1)
    if (u < lambda * n / (lambda * n + mu * n + nu)) {
      # Claim event
      claim_amount <- inverse.claim(runif(1))
      c <- c - claim_amount
    } else if (u < (lambda * n + mu * n) / (lambda * n + mu * n + nu)) {
      # Departure event
      n <- max(n - 1, 0)
    } else {
      # Enrollment event
      n <- n + 1
    }
    # Stop if capital becomes negative
    if (c < 0) break
    # Time of next event
    dt <- rexp(1, lambda * n + mu * n + nu)
    t <- t + dt
  }
  return(c)
}
```

And for multiple:

```r
simulate_insurance <- function(c0, n0, a, tl, lambda, mu, nu, MC) {
  final_capitals <- vector("numeric", MC)
  # Simulate MC paths
  final_capitals <- replicate(MC, simulate_one_insurance(c0, n0, a, tl, lambda, mu, nu))
  # Compute and return outputs
  fraction <- sum(final_capitals > 0) / MC
  pos_fc <- final_capitals[final_capitals > 0]
  mean_final_capitals <- mean(pos_fc)
  sd_final_capitals <- sd(pos_fc)
  return(list(fraction = fraction, mean_final_capital = mean_final_capitals,
              sd_final_capital = sd_final_capitals, pos_fc = pos_fc))
}
```

Antithetic variates

We try a variance reduction technique called antithetic variates. The idea is to simulate pairs of antithetic paths, that is, paths with the same initial conditions but with the sign of the increments changed. This way, the variance of the final capital is reduced by a factor of 2. The algorithm is the same as before, but we will simulate $MC/2$ pairs of antithetic paths. The final capital of the i-th path will be the sum of the final capital of the i-th antithetic path and the final capital of the $(MC/2 + i)$-th antithetic path.

```r
simulate_one_insurance_antithetic <- function(c0, n0, a, tl, lambda, mu, nu) {
  # Initialize variables
  t <- 0
  n <- n0
  c <- c0
  c_anti <- c0
  # Update capital with annuity payments
  dt <- rexp(1, lambda * n + mu * n + nu)
```

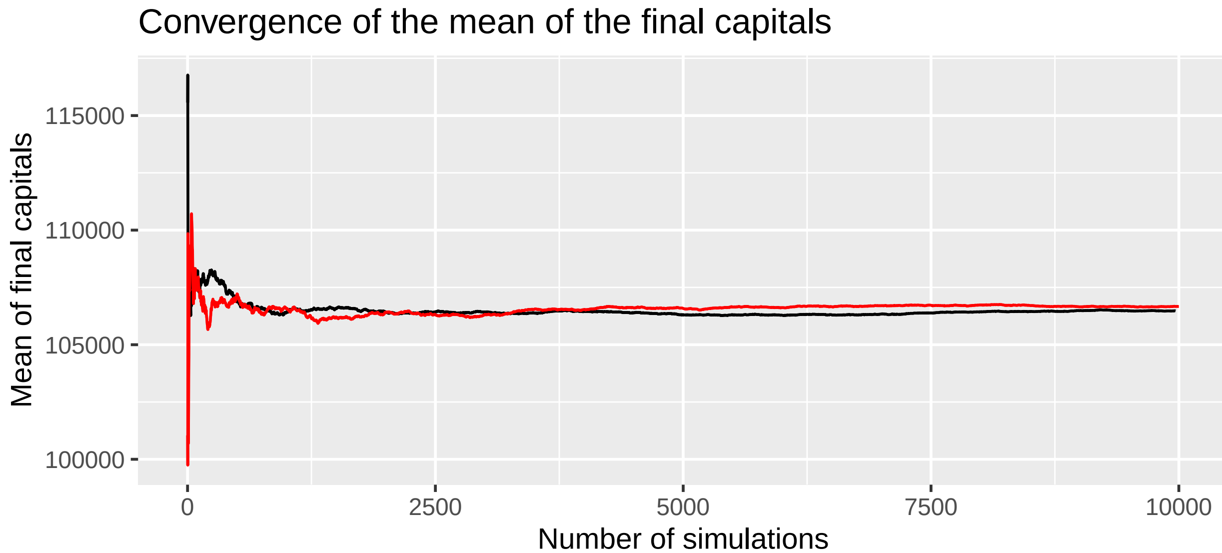
```r
  t <- t + dt
  while (t < tl) {
    # Update the capital with the proportional part of the payments
    c <- c + n * a * dt
    c_anti <- c_anti + n * a * dt
    # Decide the type of event
    u <- runif(1)
    if (u < lambda * n / (lambda * n + mu * n + nu)) {
      # Claim event
      u <- runif(1)
      if (c > 0) { # If the capital is positive, we can perform a claim
        claim_amount <- inverse.claim(u)
        c <- c - claim_amount
      }
      if (c_anti > 0) {
        # If the capital is positive for antithetic,
        # we can perform a claim else keep it negative
        claim_anti <- inverse.claim(1-u)
        c_anti <- c_anti - claim_anti
      }
    } else if (u < (lambda * n + mu * n) / (lambda * n + mu * n + nu)) {
      # Departure event
      n <- max(n - 1, 0)
    } else {
      # Enrollment event
      n <- n + 1
    }
    # Stop if both capitals becomes negative, if only one does,
    # we keep going with the other
    if (c < 0 && c_anti < 0) break
    # Time of next event
    dt <- rexp(1, lambda * n + mu * n + nu)
    t <- t + dt
  }
  return(c(c, c_anti))
}
```

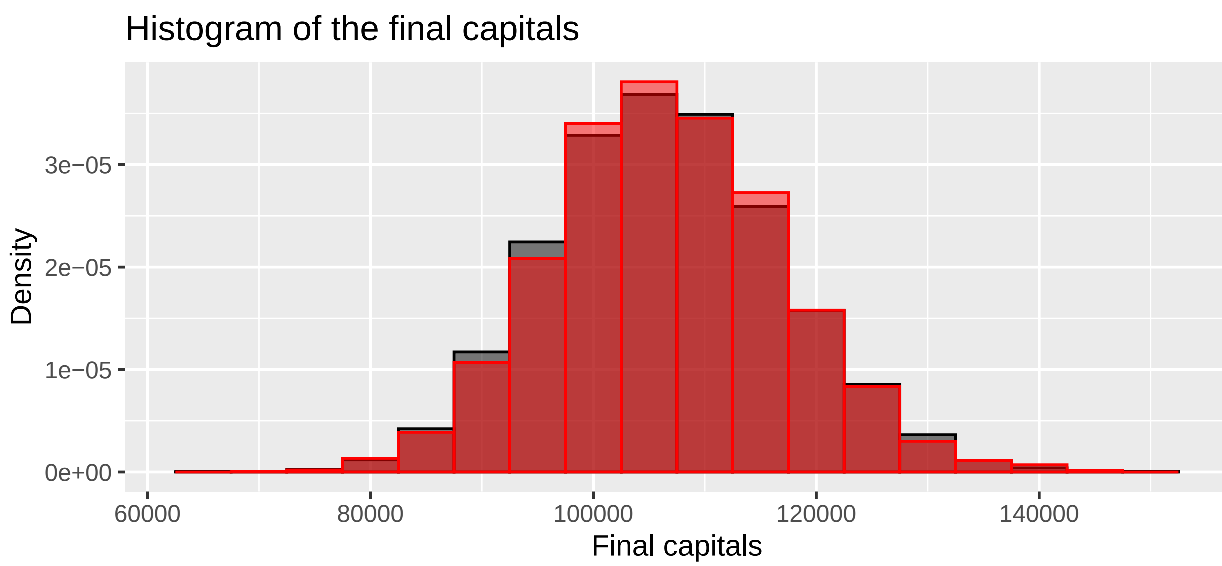
```r
simulate_insurance_antithetic <- function(c0, n0, a, tl, lambda, mu, nu, MC) {
  antithetic_final_capitals <- replicate(MC, simulate_one_insurance_antithetic(
                                         c0, n0, a, tl, lambda, mu, nu))
  fraction <- sum(antithetic_final_capitals > 0) /(MC*2)
  pos_fc <- antithetic_final_capitals[antithetic_final_capitals > 0]
  mean_final_capitals <- mean(pos_fc)
  sd_final_capitals <- sd(pos_fc)
  return(list(fraction = fraction, mean_final_capital = mean_final_capitals,
              sd_final_capital = sd_final_capitals, pos_fc = pos_fc))
}
```

Results

Table 1: Results of the simulation with and without antithetic variates

	fraction	mean_final_capital	sd_final_capital	time elapsed
normal	0.9967	106475.971675903	10516.1123920357	17.0029999999997
antithetic	1	106666.637870849	10361.4480989	10.4160000000002





Numerical results, quality assessment of the approximations, and time efficiency of the algorithms.

Conclusions

The results clearly show that the antithetic approach improves the first solution by far in terms of computing time. However, the variance reduction is not significantly lower compared to our model, as one would expect. This is because our approach is somewhat a simplification of this algorithm, as we are adapting our code to obtain negative correlated outputs instead of . . . Throughout the project we had to deal with several difficulties:

- When developing our own algorithm we first decided to simulate the type of event and modify the variables accordingly, but that was not accurate since, before doing that, we should update the proportional part of the previous payments so that they would be taken into account in the next event
- We also had trouble coming up with the total number of events that were going to happen beforehand. We later discovered that it was not possible to do this since we were dealing with a Poisson process, and it fluctuates. We were just supposed to simulate the time until the next event, so that we could afterwards simulate which event would that be
- The main difficulty we encountered was in the antithetic approach. Our first attempt was to change the sign, which we did twice and hence, we obtained the same results in both cases. After observing that, we changed ...

Overall, this project provided us with a fine representation of how insurance policies are designed and what variables come into play when it comes to making money out of them.

References

Cascos, Ignacio. 2023a. "Lecture Notes 2. Simulating Random Variables and Vectors." Aula Global UC3M.

———. 2023b. "Lecture Notes 4. Efficiency Improvement Techniques." Aula Global UC3M.

"Collective Risk Models." 2008. In *Modern Actuarial Risk Theory: Using r*, 41–86. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-70998-5_3.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2023. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wilke, Claus O. 2020. *Cowplot: Streamlined Plot Theme and Plot Annotations for Ggplot2*. https://wilkelab.org/cowplot/.

Xie, Yihui. 2014. "Knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

———. 2023. *Knitr: A General-Purpose Package for Dynamic Report Generation in r*. https://yihui.org/knitr/.