

A Study on the Application of the Chain Ladder Model in Non-Life Insurance Loss Reserves: From a Python Implementation Perspective

Zixian Wang*

Bodwell High School, Vancouver, Canada

**Corresponding author: Zixian Wang.*

Abstract

This paper looks at how the Chain Ladder model can be used for non-life insurance loss reserves, with a focus on doing it in Python. It also compares this approach with the more traditional Excel way of working. In non-life insurance, companies have to set aside reserves for losses that have happened but not yet been fully paid. When there are many products and large amounts of data, Excel can be slow, error prone, and hard to keep track of. This paper explains the main ideas behind the Chain Ladder model, shows how Python can be used to implement it, and discusses both the good points and the problems. The Python chain-ladder library supports several common methods: the basic Chain Ladder, the Mack model, Bornhuetter–Ferguson, and Cape Cod. Python is better than Excel when it comes to automation, reproducibility, and handling many files at once. However, the Chain Ladder model still depends on the assumption that past claims behaviour will continue into the future. Also, using Python in insurance companies is not always easy because people need training and regulators have to approve the methods.

Keywords

chain ladder, loss reserving, python, non-life insurance, actuarial methods

1. Introduction

Non-life insurance includes auto insurance, liability insurance, accident insurance, and other types of coverage. Unlike life insurance, non-life claims often take a long time to settle. After an incident happens, it may take several years for all the payments to be made. So insurers have to set aside loss reserves in advance. This is a big part of an insurer's liabilities, and getting the number right matters for solvency, risk management, and pricing. For a long time, most domestic insurance companies used Excel for reserve work. Excel is easy to learn and works well for small or medium-sized datasets. But as business grows and claims data increases, Excel starts to cause problems. Manual entry leads to mistakes. Spreadsheet versions get mixed up. And it is hard to trace the analysis later. In recent years, Python has become more popular in finance and insurance. It is free, not too hard to learn, and has a wide range of functions. The chain-ladder library, which is built for loss reserve modeling, has also matured over time. Against this background, this paper looks at whether using

Python for the Chain Ladder model is feasible and useful. This study is useful in two ways. First, it provides actuarial students a path from theory to Python code. Second, it offers small and medium-sized insurers a reference for whether to adopt Python. The paper is organized as follows. Chapter 2 covers the Chain Ladder method and formulas. Chapter 3 explains the Python implementation. Chapter 4 looks at advantages, challenges, and use cases. The final chapter is the conclusion.

The Chain Ladder is one of the most widely used reserving models in non-life actuarial work. Mack [1] introduced a way to calculate standard errors, turning point estimates into interval estimates. England and Verrall [2] reviewed various stochastic reserving models and set up the modern framework. Richman [3] discussed trends in actuarial data analysis tools and argued that automation is the direction things are moving. However, most existing research focuses on model theory or on using R. There is not much systematic work on Python implementations, and not enough discussion about the learning costs, regulatory approval, and industry habits that companies face when adopting Python. To fill this gap, this paper uses public sample data (the RAA General Liability dataset) to illustrate the method. It tries to answer three questions. First, how do you run Chain Ladder calculations in Python? Second, what are the pros and cons compared to Excel? Third, what are the limits of the Chain Ladder model and how can you deal with them? This paper uses a literature review method. The sources come from Google Scholar, Web of Science, and Scopus. The search terms were “Chain Ladder reserving,” “loss reserving,” and “Python actuarial science.”

2. Overview of the Chain Ladder Method

2.1 Basic Principles and Core Formulas

The Chain Ladder is the most commonly used reserve model in non-life insurance. The core assumption is that past claims development patterns will continue in the future. In practice, claims data is arranged in a loss triangle. The rows are accident years. The columns are development stages, such as the first year after the accident, the second year, and so on. The model uses historical data to calculate a development factor for each stage. This factor shows the growth from stage j to stage $j+1$. These factors are then used to predict the missing values in the lower-right corner of the triangle. Here is a simple example. Suppose there are only two accident years. In 2023, claims paid were 1 million yuan. By the next development year, they grew to 1.5 million yuan. So the factor is 1.5. In 2024, claims paid so far are 1.2 million yuan. If we assume the same growth, the final amount would be $1.2 \times 1.5 = 1.8$ million yuan. The reserve is the difference: $1.8 - 1.2 = 0.6$ million yuan. This example is simple, but it shows the basic idea—using past growth to predict the future.

The formula for the development factor is:

$$f_j = \frac{\sum C_{i,j+1}}{\sum C_{i,j}} \quad (1)$$

where $C_{i,j}$ represents the cumulative claims for the i -th accident year in the j -th development year, and f_j represents the growth multiplier from the j -th development stage to the $(j+1)$ -th development stage.

The formula for predicting final claims is:

$$\hat{C}_{\text{hat}_{i,n}} = C_{i,j} \prod_k f_k \quad (2)$$

where $\hat{C}_{i,n}$ represents the final projected claims for the i -th accident year, $C_{i,j}$ is the currently observed cumulative claims, and $\prod_{k=j}^{n-1} f_k$ represents the product of all development factors from the current development stage to the final development stage.

The formula for calculating the reserve is:

$$\text{Reserve}_i = \hat{C}_{\text{hat}_{i,n}} - C_{i,\text{current}} \quad (3)$$

That is, the final claims amount minus the claims already paid; the difference represents the loss reserve that still needs to be set aside.

2.2 Stochastic Chain Ladder Model (Mack Model)

The basic Chain Ladder gives only a single number. It does not show how uncertain that number is. Mack [1] added a way to estimate the standard error and confidence interval. The key idea is to estimate the variance of each development factor. The Mack model's variance has three parts. One is process variance, which is the random variation among policies in the same year. Another is estimation variance, which comes from the fact that development factors are estimated from data. The third is the covariance between these two. By adding these up, you get the standard error for the final estimate. This helps management think about worst-case funding needs. It is also useful for solvency and risk capital calculations. The Mack model works when accident years are independent and when the variance of the development factors is related to the claims level. Usually, people assume the variance is proportional to the square root or to the claims themselves. These assumptions should be checked in practice, for example using residual plots.

2.3 Common Extended Models: B-F and Cape Cod

The basic model has some limitations, so two common extensions have been developed. The first is the Bornhuetter–Ferguson model (B-F). Its core idea is that the reserve equals the prior expected final loss multiplied by (1 minus the maturity ratio). The prior expected final loss usually comes from industry loss ratios or the company's own pricing assumptions. The maturity ratio is estimated from the historical triangle. The B-F model is good for new business or when data is scarce, because it does not fully depend on the early data in the triangle. The second extension is the Cape Cod model. Its main feature is credibility weighting within the B-F framework. When historical data is stable, it relies more on the data. When data is volatile, it relies more on prior assumptions. Cape Cod works well when business volumes fluctuate a lot. It is also less sensitive to outliers than the basic Chain Ladder. Although both B-F and Cape Cod are seen as Chain Ladder extensions, they are different in logic. B-F uses one external prior loss ratio for the whole triangle. Cape Cod adjusts the prior weight based on how credible each accident year's data is. Recently, Dal Moro [7] has unified the stochastic structures of these three models in an incremental triangle framework, offering a new way to compare them.

2.4 Model Limitations and Applicability Conditions

The Chain Ladder model has several limitations. First, it depends heavily on the assumption that claims patterns are stable. If the insurer changes its claims policy, modifies product terms, or if the legal environment changes, the historical patterns may no longer apply. Second, the model is very sensitive to outliers. If one year has an unusually large claim, the development factor for that year gets pulled up, which affects all the other years' estimates. Third, the model needs enough data. For new products or lines with very few claims, the triangle has only a few rows. In that case, Chain Ladder results can be very unstable and need to be checked against other methods like B-F. In practice, most companies use multiple models. They usually run several—Chain Ladder, B-F, and Cape Cod—and compare the results before choosing a final number.

3. Implementing the Chain Ladder in Python

3.1 Tool Setup and Data Loading

The chain-ladder library in Python is one of the main tools for non-life reserve modeling. It was originally based on an R package with the same name. You can install it using Python's package management tools. Its main class is Triangle, which is designed to store and handle loss triangle data. There are two ways to create a triangle. One is to read a CSV file and convert it into a Triangle object. The other is to build a two-dimensional array directly in Python. To help users get started, the library includes several built-in datasets. The most common one is RAA. This comes from a historical loss development study by the Reinsurance Association of America (1991). It is a 10×10 General Liability triangle, and it is the same data that Mack [1] used in his paper. Although it is for General Liability, the triangle structure is the same as for other non-life lines like auto insurance. So the methods described here apply to those as well. After loading the data, you get a Triangle object with accident year, development stage, and cumulative claims. The missing values in the lower-right corner are the ones the model needs to predict. The full workflow from raw data to final reserves can be summarized as follows: clean and organize the raw data; build a Triangle object; choose a model (deterministic

Chain Ladder, Mack, B-F, or Cape Cod); fit the model and predict; output the final claims and reserves; and run diagnostics like residual plots and factor trend charts.

3.2 Implementation of the Deterministic Chain Ladder and Mack Models

For the deterministic Chain Ladder, you create a ChainLadder model object and then call the fit and predict methods. The Mack model works the same way—just use the Mack object instead. Both models output the predicted final claims for each accident year. The Mack model also gives standard errors. To compare the deterministic and stochastic models, you can put the results in a table. Here is an example using the first three accident years of the RAA data. The numbers are only for illustration:

Table 1: The differences between deterministic models and stochastic models

Accident Year	CL Final Claims (10,000 yuan)	Mack Final Claims (10,000 yuan)	Standard Error (10,000 yuan)	95% Confidence Interval (10,000 yuan)
1981	18,500	18,500	620	[17,285, 19,715]
1982	17,800	17,800	540	[16,742, 18,858]
1983	19,200	19,200	780	[17,671, 20,729]

As the table shows, the deterministic model gives a single number. The Mack model gives extra information about how reliable that number is. This is the main value of stochastic modeling in risk management—it answers not just “how much do we need,” but “how much could we need in a negative case.”

3.3 Multi-Model Extension of Modeling Logic

The B-F and Cape Cod models are used in the same way—just swap the model object. The B-F model needs a prior expected loss ratio, which is usually based on industry data or company experience. Cape Cod estimates a credibility-weighted prior on its own, so you do not need to input anything extra. The difference between these two extensions is in how they weight data and prior information. B-F uses one prior for the whole triangle. Cape Cod adjusts the prior weight based on the credibility of each year's data. Besides single business lines, the library also supports batch processing of multiple triangles. For example, a company might have separate triangles for auto, liability, and accident insurance. You can group them in a higher-dimensional Triangle and run the same model on all of them at once. This is very different from Excel. In Excel, if you have 20 lines, you have to copy the worksheet 20 times and adjust the formulas manually. In Python, you run the code once and everything is generated automatically.

3.4 Model Diagnostics and Visualization System

The chain-ladder library also comes with tools for diagnostics and visualization. These help you check whether the model assumptions are reasonable and whether the results can be trusted. For diagnostics, residual plots are commonly used. If you plot the residuals for different accident years and development stages, you can see whether the Mack model's variance assumption holds. If the residuals become larger as claims get larger, the variance structure may need adjusting. For development factors, the library can produce trend charts. Actuaries can look at these to spot any sudden jumps in the factors. A sharp jump might mean there was an unusual claim that year, or that claims policy changed. If that happens, the data should be reviewed. These visualization tools turn numbers into pictures, which makes it easier to spot problems and also helps when presenting results to regulators.

4. Advantages, Challenges, and Application Scenarios of Python Implementation

4.1 Advantages of Python over Excel

From a practical point of view, Python has several advantages over Excel. The first is reproducibility. In Excel, if the data for a month changes, you have to re-enter formulas and manually adjust the ranges. It is easy to miss a row or change the wrong column. In Python, you just run the saved code again. All results update automatically, and there is no manual entry. For example, a company might need to update 50 auto insurance triangles every month. In Excel, this takes about 3 hours of repetitive work. In Python, it takes about 5 minutes, and there is no need to check whether formulas were missed. Second, error rates are lower. When you drag formulas in Excel, you can easily reference the wrong row or column. These mistakes are hard to find later.

Python's automation removes this problem. Third, batch processing is much faster. When you have hundreds of triangles, Excel slows down or freezes. Python can handle the same work in seconds. Fourth, auditability is better. In Excel, it is often hard to explain where a number came from. In Python, every step is recorded in the code. With Git for version control, you can go back and see exactly what was done at any point in time. Of course, Excel is still easier to learn and more intuitive. For small companies with only a few products, Excel is still a viable choice. In the short term, the two tools complement each other.

4.2 Practical Challenges in Promoting Python in the Insurance Industry

Even though Python has many technical benefits, there are still real challenges to using it in insurance companies in China. The first is the learning curve. Most people working in non-life actuarial departments come from finance, statistics, or economics backgrounds. They are used to Excel, and many have never programmed before. Asking someone with ten years of experience to learn Python from scratch is challenging. It takes a lot of training and motivation. The second issue is regulatory. The CBIRC's reporting system is still built around Excel. Many templates ask for Excel files directly. If a company uses Python internally, it still has to copy the results back into Excel for reporting. This reduces some of the benefit of automation. Also, regulators have not clearly said whether they accept Python-based results for solvency audits. Companies have to check case by case. The third issue is industry inertia. R has been used by actuaries for a long time. The ChainLadder package in R was already mature around 2005, while Python's chain-ladder library only started to grow quickly around 2018. Many companies already have their processes and people trained in R. Switching to Python takes time and effort. So even though Python is becoming more popular in data analysis overall, it will take a while to become the standard in actuarial work.

4.3 Typical Application Scenarios and Practical Value

Python is already being used in several areas of non-life reserving. The first is routine reserve calculation. Take a medium-sized company with eight lines of business. Each line needs monthly updates for 12 development years. In the Excel days, this took two people a full week. With Python, the same work is automated and done in about 20 minutes. The actuaries only need to check the unusual results. The second area is solvency and risk management. Under the second phase of China Solvency II, companies have to do uncertainty analysis on reserves and calculate adverse deviation factors. The standard errors from the Mack model fit this purpose well. For example, Python can batch-calculate the 75th percentile reserve for each line and show which ones use up too much capital. The third area is reinsurance pricing. Reinsurers look at five to ten years of claims history for each ceding company. Python can group and analyze this data in minutes. It can spot lines with unusual claims patterns and help adjust pricing. The fourth area is audit and regulation. When a provincial regulator asked a mid-sized company for reserve working papers, the company using Python was able to submit a complete, step-by-step document. The regulator could run the code and reproduce the results. This kind of transparency is increasingly valued as regulation becomes more data-driven.

5. Conclusion

This paper has looked at the Chain Ladder model for non-life loss reserves, especially its implementation in Python. It has also compared Python with the traditional Excel approach. The chain-ladder library supports the main methods used in practice: deterministic Chain Ladder, Mack, Bornhuetter–Ferguson, and Cape Cod. Compared to Excel, Python is better for automation, reproducibility, batch processing, and auditability. But Python does not fix the model's own limitations. Chain Ladder still assumes that past patterns will continue. If the business changes or there are large claims, the results can be unreliable. Also, Python faces practical problems in insurance companies, including staff training, regulatory uncertainty, and competition from R, which is already well established.

This paper has some limitations. It is based on literature and public data, not on actual company data. It focuses on Python and does not compare it with R in detail. The RAA data is from the US, so the results may not fit Chinese products directly. Future work could use real data from a Chinese company to compare Python and Excel on the same dataset. It could also look at combining traditional reserving models with machine learning for better stability. And it would help to develop clearer regulatory guidance on validating Python-based models. Overall, Python offers a more flexible and automated approach to loss reserving. It can help

modernize actuarial departments. For students, learning Python alongside traditional methods is a beneficial way to prepare for changes in the industry. Even if they end up using Excel in their first jobs, the habit of thinking systematically and writing reproducible code is valuable.

References

- [1] Mack, T. (1993). Distribution-free calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin: The Journal of the IAA*, 23(2), 213–225. <https://www.cambridge.org/core/journals/astin-bulletin-journal-of-the-iaa/article/distributionfree-calculation-of-the-standard-error-of-chain-ladder-reserve-estimates/E8D207F9A4DCE30300A76780FE510437>
- [2] England, P. D., & Verrall, R. J. (2002). Stochastic claims reserving in general insurance. *British Actuarial Journal*, 8(3), 443–518. <https://doi.org/10.1017/S1357321700003809>
- [3] Richman, R. (2020). AI in actuarial science – a review of recent advances – part 2. *Annals of Actuarial Science*, 14(2), 258–295. <https://doi.org/10.1017/S1748499520000126>
- [4] Bogaardt, J., van der Veen, R., et al. (2024). chainladder: Property and casualty loss reserving in Python (Version 0.8.13) [Computer software]. <https://pypi.org/project/chainladder/>
- [5] Siegenthaler, F. (2023). Unbiased estimator for the ultimate claim prediction error in the chain-ladder model of Mack. *Annals of Actuarial Science*, 17(1), 118–144. <https://doi.org/10.1017/S1748499522000157>
- [6] Gesmann, M., & Murphy, D. (2022). ChainLadder: Statistical methods for loss reserving in R (Version 0.2.18) [Computer software]. <https://CRAN.R-project.org/package=ChainLadder>
- [7] Dal Moro, E. (2024). Unification of stochastic reserving models using individual claims information. *Variance*, 17(2). <https://variancejournal.org/article/116646-unification-of-stochastic-reserving-models-using-individual-claims-information>

Funding

This research received no external funding.

Conflicts of Interest

The authors declare no conflict of interest.

Acknowledgment

This paper is an output of the science project.

Open Access

This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

