The (a, b, r) class of discrete distributions with applications

by

Esther Yartey

A thesis submitted in partial fulfilment

of the requirements for the degree of

Master of Science (MSc) in Computational Sciences

The Faculty of Graduate Studies

Laurentian University

Sudbury, Ontario, Canada

THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE
Laurentian Université/Université Laurentienne
Faculty of Graduate Studies/Faculté des études supérieures

Title of Thesis Titre de la thèse	The (a; b; r) class of discrete distributions with applications	
Name of Candidate Nom du candidat	Yartey, Esther	
Degree Diplôme	Master of Science	
Department/Program Département/Programme	Computational Sciences	Date of Defence Date de la soutenance September 29, 2020

APPROVED/APPROUVÉ

Thesis Examiners/Examinateurs de thèse:

Dr. Peter Adamic
(Supervisor/Directeur de thèse)

Dr. Ratvinder Grewal
(Committee member/Membre du comité)

Dr. Abdellatif Serghini
(Committee member/Membre du comité)

Dr. Frankck Adekambi
(External Examiner/Examinateur externe)

Approved for the Faculty of Graduate Studies
Approuvé pour la Faculté des études supérieures
Dr. Serge Demers
Monsieur Serge Demers
Acting Dean, Faculty of Graduate Studies
Doyen intérimaire, Faculté des études supérieures

ACCESSIBILITY CLAUSE AND PERMISSION TO USE

Abstract

In the insurance field the number of events such as losses to the insured or claims to the insurance company are an important aspect of loss modeling. Understanding the size of claims in terms of numbers and amounts makes it possible to modify and address issues related to creating insurance contracts. In general, certain counting (or discrete) distributions are used to model the number and amount of claims. There are situations where the modelled probability of having no claim is high. Indeed this is a desirable case for the benefit of insurance companies. An approach in modeling the number of claims in this case is by using Panjer's $(a, b, 1)$ class of discrete distributions.

In this thesis, we look at a more general case of this class of distributions where there is an excess of claims at 0 to say r. We modify the existing $(a, b, 1)$ model by assigning values greater than 0 to p_0 (the probability of no claims) all the way up to p_r (the probability of r claims). We then analyze this new model in terms of goodness of fit to actual claim data and compare with the classical $(a, b, 1)$ and $(a, b, 0)$ class of discrete distributions. This is done by using the maximum likelihood estimate (MLE) in estimating the parameters of each distribution discussed. In addition, the Akaike information criterion (AIC) is used to choose between competing distributions. This new model will be called (a, b, r) class of distributions, where $r > 1$.

Keywords

Discrete distributions, claims, claim modeling, probability, $(a, b, 1)$, $(a, b, 0)$, goodness of fit, MLE, AIC, (a, b, r).

Acknowledgment

First and foremost, my thanks and praises goes to God, the Almighty, for successfully completing my research.

I would like to express my deepest and sincere gratitude to my research supervisor, Dr. Peter Adamic, for giving me the opportunity to do research and providing invaluable support and guidance throughout this research. I couldn't have chosen a better supervisor! I am also extending my heartfelt thanks to his beautiful wife, Dr. Mrs. Karen Adamic and their amazing children for their care and acceptance.

I would also like to thank the members of my supervisory committee; Dr. Abdelatif Serghini, Dr. Ratvinder Grewal and Dr. Franck Adekambi for their time spent and invaluable comments made on my thesis.

I am extremely grateful to my mum and sisters, Rosemond and Diana for their love, prayers, encouragement and support.

Special thanks goes to my amazing friend, Mr. David Wiredu, for being so supportive while I was working on my thesis and for his many constructive suggestions. Thanks for your emotional support, understanding and the keen interest shown to complete this thesis successfully and on time.

I am also grateful to my colleagues at Laurentian and also my friends in Sudbury for their encouragement and support especially Dr. Bright Afum and his family.

Contents

List of Tables

List of Figures

1 Introduction

In non-life insurance, possible profits are derived by a chain of income payments called premiums and outgoing payments called claims. Therefore estimating the accurate premium expenses is very crucial in general insurance. Setting the right premium amount of an insurance good can be a complex problem as compared to other areas such as manufacturing where the cost of a good is (relatively) known. In other financial service areas, market prices are available. In order to price premiums competitively and still hold adequate reserves for the following year, insurance companies must find models that can adequately predict future claims. In an attempt to estimate a good premium amount, the insurer starts with an expected cost and then add 'margins' to account for the product's riskiness, expenses incurred in servicing the product, and a profit/surplus allowance to the insurer. The expected cost for an insurance can be defined as the expected number of claims times the expected amount per claim, i.e., expected frequency times severity. The focus on claim count allows the insurer to consider those factors which directly affect the occurrence of a loss, thereby potentially generating a claim. The insurer can then model the frequency process to determine the likelihood that they will have to pay out a claim within a period. Ideally, it is every insurers dream to continually receive premiums for underwriting new insurance policies without ever having to pay a claim, but this is a very unlikely event.

The traditional loss distribution approach to modeling aggregate losses or total claims in a time period starts by separately fitting a frequency distribution to the number of

losses(claims) and a severity distribution to the size of losses. In modeling the number of losses to the insured or the number of claims to an insurer, one popular approach is to find appropriate counting distributions. Counting distributions are the discrete distributions that have positive probabilities only on the non-negative integers $0, 1, 2, 3, \ldots$. On the other hand, with severity distributions, claim sizes are usually modeled as continuous random variables and as such are modeled using continuous distributions. This thesis is concerned with finding a method for modeling the number of claims only (not size or cost of claim). We do this by looking at a class of discrete distributions which satisfies the recursive relation (also referred as the Panjer distribution). In this thesis, we look at the $(a, b, 0)$ and $(a, b, 1)$ class of distributions by looking at its mathematical properties and estimation procedures. We then use the above assumptions to make a more general classification which is (a, b, r) where $r > 1$. Again we derive its mathematical properties and its estimate in the four discrete distributions in this class. We then analyze this new model in terms of goodness of fit to actual claim data compared with the classical $(a, b, 1)$ and $(a, b, 0)$ class of discrete distributions.

The remainder of this thesis is organized as follows. The next section reviews existing literature surrounding insurance claim data modeling. Section 3 gives some important definitions and notations that are adapted in the rest of the thesis. Section 4 looks at frequency modeling using the Panjer's class of distributions. Section 5 estimates the parameters of the discrete distributions of interest. Section 6 extends the $(a, b, 1)$ class

of distributions and estimate its parameters using maximum likelihood estimate. Section 7 presents an analysis of the vehicle insurance dataset from Brazil. Section 8 states the results, section 9 concludes this research and section 10 suggests future areas to consider working on.

2 Literature Review

A *claim count distribution* is a sequence $S = \{s_n\}_{n\in\mathbb{N}_0}$ satisfying $s_n \geq 0$ for all $n \in \mathbb{N}_0 := \{0, 1, 2, \ldots\}$ and $\sum_{n=0}^{\infty} s_n = 1$. A claim count distribution is said to be *nondegenerate* if $s_n < 1$ holds for all $n \in \mathbb{N}_0$, see [22]. Modeling of insurance claim count data has been an active area of research for some decades. A general approach to claims data modeling is to consider separately claim count experience from the claim size experience. Also known as claim frequency and severity random variables, hence there is a risk that the future claims experience will deviate from past eventualities. It is therefore crucial to apply appropriate statistical distributions in modeling claim data variables. Although the empirical distributions are useful in analyzing claims processes, it is always suitable to model claims data by fitting a probability distribution with mathematically tractable features, [34].

Over the past years, there has been a considerable interest in count data models, particularly in the actuarial literature. Cameron and Trivedi [10], reached an important turning point in the development of models for count data by the emergence of Generalized Linear Models (GLMs). Poisson regression is a special case of GLMs that was first developed by

Nelder and Wedderbun, see [32] and was later detailed in papers of Gourieroux et al., see [17] [16]. Haussman et al. [20] also gave detailed insight of it in their work on longitudinal or panel count data models. In the non-life insurance context, McCullagh and Nelder [29], demonstrate that the use of GLMs techniques in order to estimate the frequency of claims has a priori Poisson structure. Also, Antonio et al. present the Poisson distribution as the ideal distribution for modeling claim frequency, see [3].

Despite the fact that it offers a suitable statistical support, Gourieroux and Jasiak highlights on the significant constraints that limit the use of the Poisson distribution, see [15]. The Poisson distribution suggests the equality of variance and mean, which is a property called equidispersion as shown in [11], is a particular form of unobserved heterogeneity. A common outcome of unobserved heterogeneity in count data is overdispersion i.e. the variance exceeds the mean. An alternate explanation is provided by Jong and Heller in [12], who termed the overdispersion as extra-Poisson variation since this type of data displays far greater variance than that predicted by the Poisson model.

The substitute distributions frequently used in rectifying this problem of overdispersion are known as compound or mixed distributions. A specific example of this class is the negative binomial distribution which comprises of simple and efficient techniques that manages the limits of the Poisson distribution. Amongst the relatively recent studies, Denuit et al. in [13], gives a thorough image concerning the mixed Poisson models and they highlight

that the negative binomial distribution is a sufficient alternative to Poisson distribution in order to estimate the claim frequency for an auto insurance portfolio.

Working with cross-sectional insurance data, Boucher et al. [8], sustains that the comparison of the log-likelihoods for the two distributions reveals the extra parameter of the negative binomial distribution improves the fit of data in comparison to the Poisson distribution. Also, Antonio and Valdez [2] emphasizes the practical use of the negative binomial models in modeling auto insurance claim frequency.

Bahnemann [6] argued that the number of claims arising in a portfolio is a discrete quantity which makes standard discrete distributions ideal since their probabilities are defined on only non-negative integers.

An important feature of insurance claim data is the large proportions of zeros. This may be due to unreported small claims by the policyholders, thus, the Poisson regression model may not give accurate result in this circumstance [36]. Moreover, the most important equidispersion property of Poisson distribution is violated. Though the negative binomial model was formulated as an alternative to Poisson model, this does not provide exact prediction in this case of excess zeros. A number of parametric zero-inflated count distributions have been presented by [40] to provide accommodation to the surplus zeros in insurance claim count data.

In order to overcome the restriction of these regression models, Lambert [25] introduced a more realistic way for modeling count data which has large counts and zeros, called zero-inflated Poisson (ZIP) regression models. A great number of modifications to the Poisson regression model have been presented by [18] as an extension to Lambert's ZIP regression model. Mouatassim and Ezzahid [30] fitted Poisson regression and ZIP regression to Moroccan private health insurance count data and the authors concluded that ZIP regression fits excess of zeros count data better than the standard Poisson regression. Also, Benlagha et al. [7] fitted the same models as Mouatassim to a Tunisia automobile insurance market and obtained same results in terms of the best model. If the data are overdispersed and exhibit an excess of zeros, the zero inflated negative binomial (ZINB) model is commonly used as an alternative to the ZIP model, see [39].

Another approach in dealing with extra zeros in claim counts is using the hurdle model. This model take account of all zeros in the right censored part and all positive counts in the left truncated-at-zero part.The hurdle model for count data was first discussed in [31]. In [21], this model was considered as a two part model. For handling the overdispersion and underdispersion present in the data, [19] initiated the generalized hurdle model. The difference between zero-inflated models and hurdle models have been re-examined in [26]. A new approach for modeling of zero-inflated count data, called dynamic hurdle model and giving new justifications to excessive zeros is discussed in [5]. They assumed that the counts are generated from the non-stationary stochastic process.

In 1981, Panjer considered a family of discrete random variables satisfying a recursive formula for calculating the compound distributions in cases where the corresponding claim number distribution belongs to that family, see [35]. This class of Panjer's distributions solves many problems in actuarial science and in particular risk theory. This has captivated the attention of many researchers. It is shown in [37] that the Panjer class of order 0 i.e. $(a, b, 0)$ is identical with the collection of all (nondegenerate) Poisson, negative binomial (including geometric) and binomial distributions. These are the only distributions in the Panjer class of order 0. All the distributions of the Panjer class of order 1 is identified in [38].

In this thesis, the properties of the $(a, b, 0)$ and $(a, b, 1)$ class of discrete distributions are discussed and also its importance in modeling claim count data. We extend this idea in the (a, b, r) class of distributions where $r > 1$. This is important in scenarios where there is not just an excess at zero, but in the first r counts.

3 Definitions and Notations

In this chapter, the discrete distributions of interest are defined and also some notations are adapted in this thesis.

3.1 Notations

We now adapt the following notations in modeling this discrete distributions.

Let the probability function (pf) p_k denote the probability that exactly k events (such as claims or losses) occur. Also, N is assumed to be a random variable representing the number of such events. Then

$$p_k = Pr(N = k), \qquad k = 0, 1, 2, \ldots.$$

Recall that the probability generating function (pgf) of a discrete random variable N with pf p_k is

$$P(z) = P_N(z) = E(z^N) = \sum_{k=0}^{\infty} p_k z^k.$$

3.2 Poisson Distribution

The pf for the Poisson distribution is

$$p_k = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \lambda > 0.$$

The mean and variance of the Poisson distribution are

$$E(N) = \lambda \quad \text{and} \quad Var(N) = \lambda.$$

The mean is equal to the variance for the Poisson distribution. The following is a useful property of the Poisson distribution in modeling insurance risks. Suppose for a complicated medical benefit, the number of claims follows a Poisson distribution. Also assume the types of claims to be the different medical procedures or benefits under the plan. If

one or more benefits of the plan is removed, the distribution of the number of claims under the revised plan again turns out to be a Poisson Distribution but with a new parameter λ.

3.3 Negative Binomial Distribution

The pf of the negative binomial distribution is given by

$$p_k = \binom{k+r-1}{k}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k, \quad k = 0, 1, 2, \ldots, r > 0, \beta > 0.$$

The pgf of the negative binomial is given by

$$P(z) = [1 - \beta(z-1)]^{-r}.$$

Hence the mean and variance of the negative binomial distribution are

$$E(N) = r\beta \text{ and } Var(N) = r\beta(1+\beta).$$

The variance is greater than the mean because β is positive. Thus, for a set of data where the observed variance is larger than the observed mean, then the negative binomial might be a better option than the Poisson distribution as a model to be used.

3.4 Geometric Distribution

This is a special case of the negative binomial distribution when $r = 1$. It's pf is given by

$$p_k = \frac{\beta^k}{(1+\beta)^{k+1}}.$$

Hence, the mean and variance of the geometric distribution are

$$E(N) = \beta \text{ and } Var(N) = \beta(1+\beta).$$

3.5 Binomial Distribution

The pf of the binomial distribution is

$$p_k = \binom{m}{k} q^k (1-q)^{m-k} \quad \text{where } k = 0, 1, \ldots, m.$$

The probability generating function is

$$[1 + q(z-1)]^m, \quad 0 < q < 1.$$

The mean and variance of the binomial distribution are given by

$$E(N) = mq, \; Var(N) = mq(1-q).$$

A useful attribute of the binomial distribution is that it has finite support; i.e. the range of values for which there exists positive probabilities has finite length. For instance, when one is modeling the number of accidents per automobile during a period of one year, it is very unlikely for there to be more than 12 claims since it might take some time to repair the automobile between accidents.

4 Frequency Modeling

In this section, the $(a, b, 0)$ class of discrete distributions is defined. We then look at the importance of the recursive relationship underpinning this class of distributions. Also, the conditions under which this general class reduces to each of the Poisson, geometric, negative binomial and binomial distributions are identified. The $(a, b, 1)$ class is also defined. However, the $(a, b, 1)$ class does not only contain the members of $(a, b, 0)$, we will

look at two additional distributions; the so-called extended truncated negative binomial distribution(ETNB) and the logarithmic distribution.

4.1 The (a, b, 0) Class

Definition 1. *Let p_k be the pf of a discrete random variable. It is a member of the $(a, b, 0)$ class of distributions provided that there exists constants a and b such that*

$$\frac{p_k}{p_{k-1}} = a + \frac{b}{k}, \quad k = 1, 2, 3, \ldots. \tag{4.1}$$

The recursion describes the relative size of successive probabilities in the counting distribution. Note that the recursive relation (4.1) generates all the probabilities p_k for all integers starting from 1 but does not account for the probability at zero, p_0. Does that mean the initial probability p_0 can be any arbitrary probability value? Looking at the recursive relation in (4.1), p_0 is the starting value and each p_k can ultimately be expressed in terms of p_0 as shown below, see [27].

$$\begin{aligned}
p_0 &= p_0 \\
p_1 &= (a+b)p_0 \\
p_2 &= \left(a + \frac{b}{2}\right)(a+b)p_0 \\
&\vdots \\
p_k &= \left(a + \frac{b}{k}\right)\left(a + \frac{b}{k-1}\right)\ldots\left(a + \frac{b}{2}\right)(a+b)p_0
\end{aligned}$$

When a and b are fixed, the value of p_0 is also fixed since the probabilities must sum to

1. Thus a member of $(a, b, 0)$ class has two parameters being a and b which completely determines the distribution.

4.1.1 Only Three Members in the (a, b, 0) Class

The $(a, b, 0)$ class has only three members, namely the Poisson, binomial and negative binomial distributions, with each distribution represented by a different sign of the parameter a. The following table below lists the parameters a, b for each distribution (including the geometric distribution which is a one parameter special case of the negative binomial when $r = 1$) together with its probability function at 0, see [24, p.87].

Table 4.1: Members of the (a, b, 0) class

Distribution	$\mathbf{a}$	$\mathbf{b}$	$\mathbf{p_0}$
Poisson	0	λ	$e^{-\lambda}$
Negative binomial	$\frac{\beta}{1+\beta}$	$(r-1)\frac{\beta}{1+\beta}$	$(1+\beta)^{-r}$
Geometric	$\frac{\beta}{1+\beta}$	0	$(1+\beta)^{-1}$
Binomial	$\frac{-q}{1-q}$	$(m+1)\frac{q}{1-q}$	$(1-q)^m$

We show how the following values for a, b and p_0 were derived in the proof below.

Proof.

- **Poisson**

$$\begin{aligned}
\frac{p_k}{p_{k-1}} &= \frac{e^{-\lambda}\lambda^k}{k!} \div \frac{e^{-\lambda}\lambda^{k-1}}{(k-1)!} \\
&= \frac{e^{-\lambda}\lambda^k}{k(k-1)!} \times \frac{(k-1)!}{e^{-\lambda}\lambda^{k-1}} \\
&= \frac{\lambda^{k-k+1}}{k} \\
&= \frac{\lambda}{k}
\end{aligned}$$

Therefore, $a = 0$ and $b = \lambda$.

Now, the probability function at 0 is

$$p_0 = \frac{e^{-\lambda}\lambda^0}{0!} = e^{-\lambda}$$

- **Negative binomial**

$$\begin{aligned}
\frac{p_k}{p_{k-1}} &= \binom{k+r-1}{k}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k \div \binom{(k-1)+r-1}{k-1}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^{k-1} \\
&= \frac{(k+r-1)!}{k!(k+r-1-k)!}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k \\
&\div \frac{(k+r-2)!}{(k-1)!(k+r-2-k+1)!}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^{k-1} \\
&= \frac{(k+r-1)\cancel{(k+r-2)!}}{k\,\cancel{(k-1)!}\cancel{(r-1)!}}\cancel{\left(\frac{1}{1+\beta}\right)^r}\left(\frac{\beta}{1+\beta}\right)^k \times \frac{\cancel{(k-1)!}\cancel{(r-1)!}}{\cancel{(k+r-2)!}\cancel{\left(\frac{1}{1+\beta}\right)^r}\left(\frac{\beta}{1+\beta}\right)^{k-1}} \\
&= \frac{(k+r-1)\left(\frac{\beta}{1+\beta}\right)^{k-k+1}}{k} \\
&= \frac{k\left(\frac{\beta}{1+\beta}\right)}{k} + \frac{(r-1)\left(\frac{\beta}{1+\beta}\right)}{k}
\end{aligned}$$

This implies $a = \dfrac{\beta}{1+\beta}$ and $b = (r-1)\left(\dfrac{\beta}{1+\beta}\right)$.

Now, the probability function at 0 is

$$\begin{aligned} p_0 &= \binom{0+r-1}{0}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^0 \\ &= \binom{r-1}{0}\left(\frac{1}{1+\beta}\right)^r \\ &= (1+\beta)^{-r} \end{aligned}$$

- **Geometric**

$$\begin{aligned} \frac{p_k}{p_{k-1}} &= \frac{\left(\frac{\beta}{1+\beta}\right)^{k-1}\cancel{\left(\frac{1}{1+\beta}\right)}}{\left(\frac{\beta}{1+\beta}\right)^{k-1-1}\cancel{\left(\frac{1}{1+\beta}\right)}^{k-1-k+2}} \\ &= \left(\frac{\beta}{1+\beta}\right) \\ &= \frac{\beta}{1+\beta} \end{aligned}$$

This implies $a = \dfrac{\beta}{1+\beta}$ and $b = 0$. The probability function at 0 is $p_0 = \left(\dfrac{\beta}{1+\beta}\right)^{0-1}\left(\dfrac{1}{1+\beta}\right) = \dfrac{1}{1+\beta} = (1+\beta)^{-1}$.

- **Binomial**

$$\begin{aligned} \frac{p_k}{p_k - 1} &= \binom{m}{k}q^k(1-q)^{m-k} \div \binom{m}{k-1}q^{k-1}(1-q)^{m-(k-1)} \\ &= \frac{m!}{k!(m-k)!}q^k(1-q)^{m-k} \div \frac{m!}{(k-1)!(m-k+1)!}q^{k-1}(1-q)^{m-k+1} \\ &= \frac{\cancel{m!}q^k(1-q)^{m-k}}{k\cancel{(k-1)!}\cancel{(m-k)!}} \times \frac{\cancel{(k-1)!}(m-k+1)\cancel{(m-k)!}}{\cancel{m!}q^{k-1}(1-q)^{m-k+1}} \\ &= \frac{q^{k-k+1}(1-q)^{m-k-m+k-1}(m-k+1)}{k} \\ &= \frac{q(1-q)^{-1}}{k}(m-k+1) \\ &= -\frac{q\cancel{k}}{(1-q)\cancel{k}} + (m+1)\frac{q}{(1-q)k} \end{aligned}$$

This implies $a = -\frac{q}{1-q}$ and $b = (m+1)\frac{q}{1-q}$. The probability function at 0 is

$$\begin{aligned} p_0 &= \binom{m}{0} q^0 (1-q)^{m-0} \\ &= (1-q)^m. \end{aligned}$$

□

Note that not all combinations of a and b will make a probability distribution under the recursive relation (4.1). For example, when both a and b are negative constants, the resulting probabilities p_k are negative for odd k. When $a+b<0$, the resulting probabilities p_k cannot be reliably positive in all instances. When $a+b=0$, $p_0=1$, i.e. the distribution is a point mass at 0. So we would like to restrict our attention to the case where $a+b>0$. To reiterate the point made above, when $a+b>0$ and when the recursive relation (4.1) produces a viable probability distribution, it must be one of the four distributions listed in Table 4.1 above.

4.1.2 Fitting (a, b, 0) distributions

If the $(a,b,0)$ recursive formula in (4.1) generates no new distributions, why study $(a,b,0)$ class and not just focus on Poisson, binomial, geometric and negative binomial distribution individually? One reason for studying this class is that, it gives a graphical way to choose an appropriate member of the $(a,b,0)$ class. To see this, rewrite (4.1) as follows:

$$k\frac{p_k}{p_{k-1}} = ak + b, \qquad k = 1, 2, 3, \ldots \tag{4.2}$$

Note that the expression on the left-hand side of (4.2) is a linear function of k. If we plot the left-hand side of (4.2) with k on the x-axis, the plot should be a linear one with the slope being the parameter a and the y-intercept being the parameter b.

The relation (4.2) is a way to quickly determine whether a given sample is taken from a member of the $(a, b, 0)$ class. To do this, calculate ratio such as the following for values of k:

$$k\frac{\hat{p}_k}{\hat{p}_k - 1} = k\frac{n_k}{n_k - 1}, \qquad k = 1, 2, 3, \ldots$$

where n_k is the observed frequency for the category k. The observed values should form approximately a straight line if one of these models is to be selected. The slope gives an indication of which $(a, b, 0)$ member to use. If the plot is approximately horizontal, then the Poisson model is appropriate.If the plot is a line with a negative slope, then the binomial model is more appropriate. If the plot is approximately a line with positive slope then the negative binomial model including the geometric is more appropriate.

Note that this approach will not work if any of the n_k is zero. Hence this procedure is less useful for a small number of observations.

4.1.3 Examples

We now present a few examples to illustrate how the $(a, b, 0)$ recursive relation works, for more see [14, chapter 28].

1. Let N be a member of $(a, b, 0)$ satisfying the recursive probabilities

$$k\frac{p_k}{p_{k-1}} = \frac{3}{4}k + 3.$$

Identify the distribution N.

Solution:

We can see that $a > 0$ and $b > 0$ thus N must be a negative binomial distribution. We have $\frac{\beta}{1+\beta} = \frac{3}{4} \implies 4\beta = 3 + 3\beta \implies \beta = 3$. Also, we have $(r-1)\frac{\beta}{1+\beta} = 3 \implies r - 1 = \frac{3(1+\beta)}{\beta} \implies r = 1 + \frac{3(1+3)}{3} \implies r = 5$.

2. The distribution of accidents for 84 randomly selected policies is as follows:

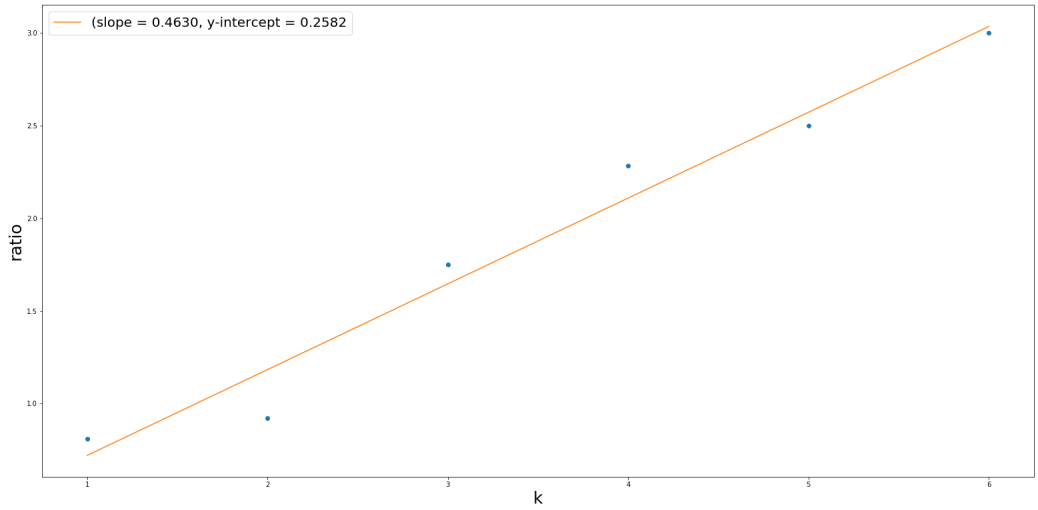
Number of Accidents, k	0	1	2	3	4	5	6
Number of Policies, n_k	32	26	12	7	4	2	1

Identify the frequency model that best represents these data.

Solution:

Number of Accidents, $\mathbf{k}$	**Number of Policies,** $\mathbf{n_k}$	$\mathbf{k\frac{n_k}{n_{k-1}}}$
0	32	N/A
1	26	0.8125
2	12	0.9231
3	7	1.75
4	4	2.2857
5	2	2.5
6	1	3

Figure 4.1: Plot of the ratio $\mathbf{kn_k/n_{k-1}}$ against $\mathbf{k}$



From figure 4.1 above, we see that the negative binomial distribution is the best model from $(a, b, 0)$ to use since the slope $a = 0.4630$ is positive.

3. The number of dental claims in a year follows a Poisson distribution with a mean of λ. The probability of exactly 6 claims during a year is 40% of the probability that there will be 5 claims. Determine the probability that there will be 4 claims.

Solution:

Let N be the number of dental claims in a year. Since N is a member of $(a, b, 0)$ we can write

$$\frac{p_k}{p_{k-1}} = a + \frac{b}{k} = \frac{\lambda}{k}, \qquad k = 1, 2, \ldots$$

We are told from the question example above that $\frac{p_6}{p_5} = \frac{40}{100}$.

$\implies \frac{\lambda}{6} = 0.4 \implies \lambda = 2.4$. Thus $p_4 = \frac{e^{-\lambda}\lambda^4}{4!} = \frac{e^{-2.4}(2.4)^4}{4!} = 0.1254$.

4. X is a discrete random variable with a probability function which is a member of the $(a, b, 0)$ class of distributions. You are given:

   (i) $Pr(X = 0) = Pr(X = 1) = 0.25$

   (ii) $Pr(X = 2) = 0.1875$

   Calculate $Pr(X = 3)$.

Solution:

Let N represent the distribution above. Since N is a member of the $(a, b, 0)$ class, we have the recursive equation

$$p_k = \left(a + \frac{b}{k}\right) p_{k-1}, \quad k = 1, 2, \ldots$$

From (i) we have $0.25 = (a + b)(0.25)$

$$\implies a + b = 1 \tag{4.3}$$

From (ii) we have $0.1875 = \left(a + \frac{b}{2}\right)(0.25) \implies \frac{0.1875}{0.25} = a + \frac{b}{2}$

$$\implies a + \frac{b}{2} = 0.75 \tag{4.4}$$

Equation (4.3) - (4.4): $b - \frac{b}{2} = 1 - 0.75 \implies \frac{b}{2} = 0.25 \implies b = 0.5$

Substitute b into (4.3): $a = 1 - b \implies a = 1 - 0.5 \implies a = 0.5$

$\therefore P(X = 3) = \left(a + \frac{b}{k}\right) p_{k-1} = \left(0.5 + \frac{0.5}{3}\right) \times 0.1875 = 0.125.$

5. Let N be a counting distribution in $(a, b, 0)$ satisfying

   - $p_0 = \frac{1}{4^5}$
   - $\frac{p_k}{p_{k-1}} = c(0.25 + \frac{1}{k}),\ k = 1, 2, 3, \ldots.$

   Determine the value of c.

   Solution:

   The class $(a, b, 0)$ satisfies the recursive equation

   $$\frac{p_k}{p_{k-1}} = a + \frac{b}{k}.$$

   This means that the given distribution above can be written as

   $$\frac{p_k}{p_{k-1}} = 0.25c + \frac{c}{k} \implies a = 0.25c,\ b = c.$$

   We would go through each distribution in the class and see which one fits the condition of the problem.

- If the distribution is Poisson, then we have

$$a = 0 \implies c = \frac{a}{0.25} = 0 \implies b = c = 0.$$

Thus $p_k = 0$ for $k = 1, 2, \ldots$.

Hence $\sum_{k=0}^{\infty} p_k = p_0 = \frac{1}{4^5} \neq 1$.

Therefore this distribution cannot be the answer.

- If the distribution is geometric then $b = 0 \implies c = 0 \implies a = 0$ and as in the previous case, the distribution cannot be geometric.

- The distribution cannot be binomial since $a = 0.25c = 0.25b$. This implies $\frac{a}{b} = 0.25$. This shows that a and b are of the same signs but we know for a binomial distribution, a and b must be of opposite signs.

- The only possibility remaining is that the distribution is negative binomial. In this case $a = \frac{\beta}{1+\beta}$ and $b = (r-1)\frac{\beta}{1+\beta}$.

We have $\frac{a}{b} = \frac{1}{(r-1)} = 0.25 \implies r = 1 + \frac{1}{0.25} \implies r = 5$.

Also,

$$\begin{aligned} p_0 = \frac{1}{4^5} &= (1+\beta)^{-r} \\ \implies \frac{1}{(1+\beta)^r} &= \frac{1}{4^5} \\ \implies (1+\beta)^r &= 4^5 \quad \text{but } r = 5 \\ \implies 1+\beta &= \sqrt[5]{4^5} \\ \implies \beta &= 4 - 1 = 3. \end{aligned}$$

Finally,

$$\begin{aligned} c &= b = (r-1)\frac{\beta}{1+\beta} \\ &= (5-1)\frac{3}{1+3} \\ &= 3. \end{aligned}$$

6. Suppose that the number of claims N has a Poisson distribution with mean $\lambda = 3$. Calculate $\frac{p_{255}}{p_{254}}$.

Solution:

N belongs to the $(a, b, 0)$ class, so it satisfies the recursive equation

$$\frac{p_k}{p_{k-1}} = a + \frac{b}{k} \text{ where } a = 0 \text{ and } b = \lambda.$$

Therefore

$$\begin{aligned} \frac{p_{255}}{p_{254}} &= 0 + \frac{\lambda}{255} \quad \text{but } \lambda = 3 \\ \implies \frac{p_{255}}{p_{254}} &= \frac{3}{255} = 0.0118. \end{aligned}$$

4.2 The (a, b, 1) Class of Discrete Distributions

We notice from the $(a, b, 0)$ class that, all of the members have a fixed positive probability at 0. Sometimes they do not appropriately describe the characteristics of some data sets encountered in practice. In this section, the problem of a poor fit at the left-hand end of the distribution, particularly the probability at zero is addressed.

4.2.1 Truncation or Modification at zero

In an insurance study, if we consider a case where the number of claims under study is only for those losses that have resulted in a claim, then the minimum observed value is 1 and $p_0 = 0$. For example if one is counting the number of claims from accidents resulting in a claim, the minimum observed value is 1. This distribution can be created from $(a, b, 0)$ by assigning $p_0 = 0$ and dividing the remaining probabilities by $1-p_0$. This new distribution is called a zero-truncated distribution.

Similarly, there can also be an instance where the probability of having no claim is high. Consider the case of a group dental insurance in an household. Suppose both husband and wife have coverage with their employee-sponsored plans and both group insurance contracts provide coverage for all family members, then claims will be made to the insurer of the plan with better benefits and no claims may be made under the other contract. Then in conducting studies for a particular insurer, one may find a higher than expected number of individuals who made no claim. Another example is in auto insurance, policy holders (or the insured) may not want to report the first claim because of fear that it may increase future insurance rates. This behavior will inflate the proportion of zero counts. Hence we modify the model by assigning a large value to p_0 and then modify the other values of p_k so as to have a probability distribution. This is called a zero-modified distribution.

We now look at a formal definition of the $(a, b, 1)$ class.

Definition 2. *Let p_k be the pf of a discrete random variable. It is a member of the class $(a, b, 1)$ of discrete distributions provided that there exists constants a and b such that*

$$\frac{p_k}{p_{k-1}} = a + \frac{b}{k}, \quad k = 2, 3, 4, \ldots \tag{4.5}$$

The only difference from the class $(a, b, 0)$ is that the recursion begins at p_1 rather than p_0.

Let N be a discrete non-negative random variable with pf p_k, $k = 0, 1, 2, \ldots$.

We can create a new random variable N^* with pf p_k^* such that p_0^* has a preassigned value in the interval $[0, 1)$. The pf of N^* is given by

$$p_k^* = \frac{1 - p_0^*}{1 - p_0} p_k, \quad k = 1, 2, 3, \ldots.$$

Note that

$$\sum_{k=0}^{\infty} p_k^* = p_0^* + \sum_{k=1}^{\infty} p_k^* = p_0^* + \frac{1 - p_0^*}{1 - p_0} \sum_{k=1}^{\infty} p_k = p_0^* + \frac{1 - p_0^*}{1 - p_0}(1 - p_0) = 1.$$

We say that N^* has a **zero-modified** distribution.

In the special case when $p_0^* = 0$, we say that N^* has a **zero-truncated** distribution. Also note that the zero-truncated distribution is a special case of the zero-modified distribution.

The probabilities of a zero-modified distribution are denoted by p_k^M where

$$p_k^M = \frac{1 - p_0^M}{1 - p_0} p_k, \quad k = 1, 2, \ldots$$

and p_0^M is an arbitrary number in $[0, 1)$.

The probabilities of a zero-truncated distribution are denoted by

$$p_k^T = \frac{1}{1 - p_0} p_k, \quad k = 1, 2, 3, \ldots$$

and $p_0^T = 0$.

Theorem 1. *Let N be in $(a, b, 0)$ with corresponding moment generating function $M_N(t)$.*

Then the moment generating function of N^M is

$$M_N^M(t) = \frac{p_0^M - p_0}{1 - p_0} + \left(\frac{1 - p_0^M}{1 - p_0}\right) M_N(t).$$

Proof. We have

$$\begin{aligned}
M_N^M(t) &= E(e^{tN}) = \sum_{n=0}^{\infty} e^{tN} p_n^M \\
&= p_0^M + \left(\frac{1 - p_0^M}{1 - p_0}\right) \sum_{n=1}^{\infty} e^{tN} p_n \\
&= p_0^M + \left(\frac{1 - p_0^M}{1 - p_0}\right) \left[\sum_{n=0}^{\infty} e^{tN} p_n - p_0\right] \\
&= p_0^M + \left(\frac{1 - p_0^M}{1 - p_0}\right) [M_N(t) - p_0] \\
&= p_0^M - p_0 \left(\frac{1 - p_0^M}{1 - p_0}\right) + \left(\frac{1 - p_0^M}{1 - p_0}\right) M_N(t) \\
&= \frac{p_0^M(1 - p_0) - p_0(1 - p_0^M)}{1 - p_0} + \left(\frac{1 - p_0^M}{1 - p_0}\right) M_N(t) \\
&= \frac{p_0^M - p_0^M p_0 - p_0 + p_0 p_0^M}{1 - p_0} + \left(\frac{1 - p_0^M}{1 - p_0}\right) M_N(t) \\
&= \frac{p_0^M - p_0}{1 - p_0} + \left(\frac{1 - p_0^M}{1 - p_0}\right) M_N(t)
\end{aligned}$$

□

Corollary 1. *Let N be in $(a, b, 0)$ with corresponding probability generating function $P_N(t)$.*

Then the probability generating function of N^M is

$$P_N^M(t) = \frac{p_0^M - p_0}{1 - p_0} + \left(\frac{1 - p_0^M}{1 - p_0}\right) P_N(t).$$

Proof.

$$\begin{aligned} P_N^M(t) &= M_N^M(\ln t) \\ &= \frac{p_0^M - p_0}{1-p_0} + \left(\frac{1-p_0^M}{1-p_0}\right) M_N(\ln t) \\ &= \frac{p_0^M - p_0}{1-p_0} + \left(\frac{1-p_0^M}{1-p_0}\right) P_N(t). \end{aligned}$$

□

Remarks 1. *Note that*

$$P_N^M(t) = \left(1 - \frac{1-p_0^M}{1-p_0}\right) \cdot 1 + \left(\frac{1-p_0^M}{1-p_0}\right) P_N(t).$$

Thus, N^M *is a mixture of a degenerate distribution (i.e a distribution with support consisting of a single point) with mixing weight* $\left(\frac{p_0^M - p_0}{1-p_0}\right)$ *and the corresponding member of* $(a, b, 0)$ *with mixing weight* $\left(\frac{1-p_0^M}{1-p_0}\right)$*.*

4.2.2 Examples

1. Show that $p_k^M = (1 - p_0^M)p_k^T, k = 1, 2, \ldots$.

   Solution:

   We know that

$$\begin{aligned} p_k^M &= \left(\frac{1-p_0^M}{1-p_0}\right) p_k \\ &= (1-p_0^M)\left(\frac{1}{1-p_0}\right) p_k \\ &= (1-p_0^M)p_k^T, \qquad \text{where } k = 1, 2, \ldots. \end{aligned}$$

2. Show that $E(N^M) = \left(\frac{1-p_0^M}{1-p_0}\right) E(N)$.

Solution:

$$\begin{aligned} E(N^M) &= \sum_{n=0}^{\infty} n P_n^M \\ &= \sum_{n=0}^{\infty} n \left(\frac{1-p_0^M}{1-p_0}\right) P_n \\ &= \left(\frac{1-p_0^M}{1-p_0}\right) \sum_{n=0}^{\infty} n P_n \\ &= \left(\frac{1-p_0^M}{1-p_0}\right) E(N). \end{aligned}$$

3. Let N^M be the zero-modified distribution associated to N. Find $E[N^M(N^M-1)]$ and $Var(N^M)$.

Solution:

$$\begin{aligned} E[N^M(N^M-1)] &= \left(\frac{1-p_0^M}{1-p_0}\right) E[N(N-1)] \\ Var(N^M) &= E[N^M(N^M-1)] + E(N^M) - [E(N^M)]^2 \\ &= \left(\frac{1-p_0^M}{1-p_0}\right) E[N(N-1)] + \left(\frac{1-p_0^M}{1-p_0}\right) E(N) - \left[\left(\frac{1-p_0^M}{1-p_0}\right) E(N)\right]^2. \end{aligned}$$

4. Let N be a Poisson distribution with mean 1.

   (a) Determine p_0, p_1 and p_2.

   (b) Find the mean and the variance of $E(N^T)$.

Solution:

We have a Poisson distribution with mean $\lambda = 1$.

(a) $p_0 = e^{-\lambda} = e^{-1} = \frac{1}{e} = 0.367879$

We have $p_k = \left(\frac{b}{k}\right) p_{k-1}$, $\qquad k = 1, 2, \ldots$.

But we know $b = \lambda = 1$, therefore

$$\begin{aligned} p_1 &= \frac{1}{1}(p_0) = p_0 = 0.367879 \\ p_2 &= \frac{1}{2}(p_1) = \frac{1}{2}\left(\frac{1}{e}\right) = 0.183940. \end{aligned}$$

(b) We know $E(N) = \lambda = 1$.

$$\begin{aligned} E(N^T) &= \left(\frac{1}{1-p_0}\right) E(N) \\ &= \left(\frac{1}{1-\frac{1}{e}}\right) \cdot 1 \\ &= 1.581977. \end{aligned}$$

$$\begin{aligned} Var(N^T) &= E[N^T(N^T-1)] + E(N^T) - [E(N^T)]^2 \\ &= \left(\frac{1}{1-p_0}\right) E[N(N-1)] + E(N^T) - [E(N^T)]^2 \\ &= \left(\frac{1}{1-\frac{1}{e}}\right) \cdot 1 + 1.581977 - (1.581977)^2 \\ &\quad \text{since } E[N(N-1)] = \lambda^2 = 1^2 = 1 \end{aligned}$$

$$\therefore Var(N^T) = 0.661302.$$

4.2.3 The Extended Truncated Negative Binomial model (ETNB)

The $(a, b, 1)$ class not only contains members of $(a, b, 0)$ but also two additional distributions. The (a, b) parameter space can be expanded to admit an extension of the negative binomial distribution to include cases where $-1 < r < 0$. For the $(a, b, 0)$ class, $r > 0$ is required. By adding the additional region to the sample space, the ETNB distribution has parameter restrictions $\beta > 0$, $r > -1$, $r \neq 0$.

The probability function of the ETNB is defined recursively as

$$\begin{aligned} p_0 &= 0 \\ \frac{p_k}{p_{k-1}} &= a + \frac{b}{k}, \quad k = 2, 3, \ldots \end{aligned}$$

where

$$a = \frac{\beta}{1+\beta}, \ \beta > 0 \ \text{ and } b = (r-1)\frac{\beta}{1+\beta}, \ r > -1, r \neq 0.$$

When $r \to 0$, the limiting case of the ETNB is the logarithmic distribution with

$$p_k^T = \left(\frac{\beta}{1+\beta}\right)^k \frac{1}{k \ln 1 + \beta}, \quad k = 1, 2, 3, \ldots. \tag{4.6}$$

The zero-modified logarithmic distribution is created by assigning an arbitrary probability at zero and reducing the remaining probabilities, see [28].

There are no other members of the $(a, b, 1)$ class beyond the ones discussed above. The following table gives a summary of the members in $(a, b, 1)$, see [24, page 94].

Table 4.2: Members of (a, b, 1) class

Distribution	$\mathbf{p_0}$	$\mathbf{a}$	$\mathbf{b}$	**Parameter space**
Poisson	$e^{-\lambda}$	0	λ	$\lambda > 0$
ZT Poisson	0	0	λ	$\lambda > 0$
ZM Poisson	Arbitrary	0	λ	$\lambda > 0$
Binomial	$(1-q)^m$	$-\frac{q}{1-q}$	$(m+1)\frac{q}{1-q}$	$0 < q < 1$
ZT binomial	0	$-\frac{q}{1-q}$	$(m+1)\frac{q}{1-q}$	$0 < q < 1$
ZM binomial	Arbitrary	$-\frac{q}{1-q}$	$(m+1)\frac{q}{1-q}$	$0 < q < 1$
Geometric	$(1+\beta)^{-1}$	$\frac{\beta}{1+\beta}$	0	$\beta > 0$
ZT geometric	0	$\frac{\beta}{1+\beta}$	0	$\beta > 0$
ZM geometric	Arbitrary	$\frac{\beta}{1+\beta}$	0	$\beta > 0$
Negative binomial	$(1+\beta)^{-r}$	$\frac{\beta}{1+\beta}$	$(r-1)\frac{\beta}{1+\beta}$	$r > 0, \beta > 0$
ETNB	0	$\frac{\beta}{1+\beta}$	$(r-1)\frac{\beta}{1+\beta}$	$-1 < r < 0, \beta > 0$
ZM ETNB	Arbitrary	$\frac{\beta}{1+\beta}$	$(r-1)\frac{\beta}{1+\beta}$	$-1 < r < 0, \beta > 0$
Logarithmic	0	$\frac{\beta}{1+\beta}$	$-\frac{\beta}{1+\beta}$	$\beta > 0$
ZM logarithmic	Arbitrary	$\frac{\beta}{1+\beta}$	$-\frac{\beta}{1+\beta}$	$\beta > 0$

4.3 The (a, b, r) Class of Discrete Distributions

Let N be a discrete non-negative random variable with pf p_k, $k = 0, 1, 2, \ldots$. We can create a new random variable such that p_0^M, $p_1^M, p_2^M, \ldots, p_{r-1}^M$ has a preassigned value in the interval $[0, 1)$. The pf of N^M is given by

$$\begin{aligned} p_k^M &= \frac{1 - (p_0^M + p_1^M + \ldots + p_{r-1}^M)}{1 - (p_0 + p_1 + \ldots + p_{r-1})} p_k, \quad k = r, r+1, \ldots \\ &= \frac{1 - \sum_{i=0}^{r-1} p_i^M}{1 - \sum_{i=0}^{r-1} p_i} \sum_{k=r}^{\infty} p_k \end{aligned}$$

Corollary 2. *Let N be in (a, b, r) with corresponding moment generating function $M_N(t)$. Then the moment generating function of N^M is*

$$M_N^M(t) = \sum_{n=0}^{r-1} e^{tn} p_n^M + \left(\frac{1 - \sum_{n=0}^{r-1} p_n^M}{1 - \sum_{n=0}^{r-1} p_n} \right) \left[M_N(t) - \sum_{n=0}^{r-1} e^{tn} p_n \right] \tag{4.7}$$

Proof.

$$\begin{aligned}
M_N^M(t) &= E(e^{tN}) = \sum_{n=0}^{\infty} e^{tn} p_n^M \\
&= e^0 p_0^M + e^t p_1^M + e^{2t} p_2^M + \ldots + e^{(r-1)t} p_{r-1}^M + \sum_{n=r}^{\infty} e^{tn} p_n^M \\
&= e^0 p_0^M + e^t p_1^M + e^{2t} p_2^M + \ldots + e^{(r-1)t} p_{r-1}^M + \sum_{n=r}^{\infty} e^{tn} \left(\frac{1 - \sum_{k=0}^{r-1} p_k^M}{1 - \sum_{k=0}^{r-1} p_k} \right) p_n \\
&= \sum_{n=0}^{r-1} e^{tn} p_n^M + \left(\frac{1 - \sum_{n=0}^{r-1} p_n^M}{1 - \sum_{n=0}^{r-1} p_n} \right) \left[\sum_{n=0}^{\infty} e^{tn} p_n - \sum_{n=0}^{r-1} e^{tn} p_n \right] \\
&= \sum_{n=0}^{r-1} e^{tn} p_n^M + \left(\frac{1 - \sum_{n=0}^{r-1} p_n^M}{1 - \sum_{n=0}^{r-1} p_n} \right) \left[M_N(t) - \sum_{n=0}^{r-1} e^{tn} p_n \right].
\end{aligned}$$

□

5 Parameter Estimation for Discrete distributions

Estimation by the method of moments seems easy to do, but they tend to perform poorly mainly because they use few features of the data, rather than the entire set of observations. It is particularly important to use as much information as possible especially when the population has a heavy tail. Thus in this section, we define the maximum likelihood estimate and then calculate the maximum likelihood estimator for the Poisson, geometric, binomial and negative binomial distributions. The special considerations that apply when

estimating these parameters are also discussed.

Definition 3. *The **likelihood function** is*

$$L(\theta) = \prod_{j=1}^{n} Pr(X_j \in A_j|\theta) \tag{5.1}$$

*and the **maximum likelihood estimate** of θ is the vector that maximizes the likelihood function.*

The symbol "|" is used to indicate that the distribution also depends on a paramter θ, where θ could be a real-valued unknown parameter or a vector of parameters.

5.1 The (a, b, 0) Class Estimation

Let N be a random variable in the class of $(a, b, 0)$ with probability function $p_k = P_r(N = k)$, $k = 0, 1, 2, \ldots$. Let n_k be the number of observations for which $N = k$ and n be the total number of observations or the sample size. Note that $n = \sum_{k=0}^{\infty} n_k$.

5.1.1 Poisson Distribution

The Poisson distribution has only one parameter i.e λ, and its pf is defined as

$$p_k = \frac{e^{-\lambda}\lambda^k}{k!}$$

The likelihood function is

$$L(\lambda) = \prod_{k=0}^{\infty} \left(\frac{e^{-\lambda}\lambda^k}{k!}\right)^{n_k}.$$

Then the log likelihood function is

$$\begin{aligned}\ell(\lambda) = \ln(L(\lambda)) &= \sum_{k=0}^{\infty} n_k \ln\left(\frac{e^{-\lambda}\lambda^k}{k!}\right) \\ &= \sum_{k=0}^{\infty} n_k \left[-\lambda \ln(e) + k\ln(\lambda) - \ln(k!)\right]\end{aligned}$$

Let the derivative of $\ell(\lambda)$ w.r.t. λ be zero, i.e

$$\begin{aligned}&\ell'(\lambda) = -n + \frac{1}{\lambda}\sum_{k=0}^{\infty} k n_k = 0 \\ \implies &\sum_{k=0}^{\infty} k n_k = \lambda n \\ \implies &\hat{\lambda} = \frac{\sum_{k=0}^{\infty} k n_k}{n} = \overline{x} \quad \text{(sample mean).}\end{aligned}$$

5.1.2 Geometric Distribution

The geometric distribution also has one parameter i.e. β. Recall that its pf is

$$p_k = \left(\frac{1}{1+\beta}\right)\left(\frac{\beta}{1+\beta}\right)^k = \frac{\beta^k}{(1+\beta)^{k+1}}$$

The Likelihood function is

$$L(\beta) = \prod_{k=0}^{\infty}\left[\frac{\beta^k}{(1+\beta)^{k+1}}\right]^{n_k} = \prod_{k=0}^{\infty}\frac{\beta^{k n_k}}{(1+\beta)^{n_k(k+1)}}$$

Taking the log of the likelihood function above, we get

$$\begin{aligned}
\ell(\beta) &= \ln(L(\beta)) \\
&= \sum_{k=0}^{\infty} [kn_k \ln(\beta) - n_k(k+1)\ln(1+\beta)] \\
&= \ln(\beta) \sum_{k=0}^{\infty} kn_k - \ln(1+\beta) \sum_{k=0}^{\infty} n_k(k+1)
\end{aligned}$$

Taking the derivative and setting to zero, we have

$$\begin{aligned}
&\ell'(\hat{\beta}) = \frac{1}{\hat{\beta}} \sum_{k=0}^{\infty} kn_k - \frac{1}{1+\hat{\beta}} \sum_{k=0}^{\infty} n_k(k+1) = 0 \\
&\implies \frac{(1+\hat{\beta}) \sum_{k=0}^{\infty} kn_k - \hat{\beta} \sum_{k=0}^{\infty} n_k(k+1)}{\hat{\beta}(1+\hat{\beta})} = 0 \\
&\implies \sum_{k=0}^{\infty} kn_k + \hat{\beta} \sum_{k=0}^{\infty} kn_k - \hat{\beta} \sum_{k=0}^{\infty} kn_k - \hat{\beta} \sum_{k=0}^{\infty} n_k = 0 \\
&\implies \hat{\beta} \sum_{k=0}^{\infty} n_k = \sum_{k=0}^{\infty} kn_k \\
&\implies \hat{\beta} = \frac{\sum_{k=0}^{\infty} kn_k}{n} = \overline{x} \quad \text{(sample mean)}.
\end{aligned}$$

5.1.3 Binomial Distribution

The binomial distribution has two parameters m and q. In most cases, the value of m is known, thus only one parameter q needs to be estimated. Also, q is usually interpreted as the probability of some event such as death or disability in most insurance situations. Recall that $p_k = \binom{m}{k} q^k (1-q)^{m-k}$ where $k = 0, 1, 2, \ldots, m$.

Then, the likelihood function is

$$\begin{aligned} L(m,q) &= \prod_{k=0}^{\infty}\left[\binom{m}{k}q^k(1-q)^{m-k}\right]^{n_k} \\ &= \prod_{k=0}^{\infty}\binom{m}{k}^{n_k}q^{kn_k}(1-q)^{n_k(m-k)}. \end{aligned}$$

Taking the log of the likelihood function above, we have

$$\begin{aligned} \ell(m,q) &= \ln[L(m,q)] \\ &= \sum_{k=0}^{\infty}\left[n_k\ln\binom{m}{k} + kn_k\ln q + n_k(m-k)\ln(1-q)\right]. \end{aligned} \tag{5.2}$$

In most cases the value of m is known and fixed, hence only one parameter q needs to be estimated in such circumstance.

Taking the derivative of ℓ w.r.t to q

$$\ell'_q(m,q) = \sum_{k=0}^{\infty} kn_k \cdot \frac{1}{\hat{q}} + \sum_{k=0}^{\infty} n_k(m-k)\cdot\frac{1}{1-\hat{q}}\cdot(-1)$$

Setting $\ell'_q(m,q)$ to zero we have,

$$
\begin{aligned}
& \frac{1}{\hat{q}}\sum_{k=0}^{\infty} kn_k - \frac{1}{1-\hat{q}}\sum_{k=0}^{\infty} n_k(m-k) = 0 \\
\Longrightarrow \quad & \frac{(1-\hat{q})\sum_{k=0}^{\infty} kn_k - \hat{q}\sum_{k=0}^{\infty} n_k(m-k)}{\hat{q}(1-\hat{q})} = 0 \\
\Longrightarrow \quad & \sum_{k=0}^{\infty} kn_k - \hat{q}\cancel{\sum_{k=0}^{\infty} kn_k} - \hat{q}m\sum_{k=0}^{\infty} n_k + \hat{q}\cancel{\sum_{k=0}^{\infty} kn_k} = 0 \\
\Longrightarrow \quad & \hat{q}m\sum_{k=0}^{\infty} n_k = \sum_{k=0}^{\infty} kn_k \\
\Longrightarrow \quad & \hat{q} = \frac{\sum_{k=0}^{\infty} kn_k}{m\sum_{k=0}^{\infty} n_k} \\
& = \frac{\sum_{k=0}^{\infty} kn_k}{mn}
\end{aligned}
$$

When m is also unknown, the MLE of q is

$$\hat{q} = \frac{\sum_{k=0}^{\infty} kn_k}{\hat{m}n} = \frac{\overline{x}}{\hat{m}} \tag{5.3}$$

where $\hat{m}$ is the MLE of m.

Differentiating ℓ w.r.t m in (5.2) we have

$$\begin{aligned}
\ell'_m(m,q) &= \sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln\left(\frac{m!}{k!(m-k)!}\right) + \sum_{k=0}^{\infty} n_k \ln(1-q) \\
&= \sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln \frac{m(m-1)(m-2)\ldots(m-k+1)\cancel{(m-k)!}}{k!\cancel{(m-k)!}} + n\ln(1-q) \\
&= \sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln \prod_{r=0}^{k-1}(m-r) + n\ln(1-q) \\
&= \sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \sum_{r=0}^{k-1} \ln(m-r) + n\ln(1-q) \\
&= \sum_{k=1}^{\infty} n_k \sum_{r=0}^{k-1} \frac{1}{\hat{m}-r} \cdot (1) + n\ln(1-q)
\end{aligned}$$

Setting the above to zero yields

$$\sum_{k=1}^{\infty} n_k \sum_{r=0}^{k-1} \frac{1}{\hat{m}-r} = -n\ln(1-q)$$

Substituting $\hat{q}$ from (5.3) into the above, we get

$$\sum_{k=1}^{\infty} n_k \left(\sum_{r=0}^{k-1} \frac{1}{\hat{m}-r}\right) = -n\ln(1-\frac{\overline{x}}{\hat{m}}). \tag{5.4}$$

An easy approach in estimating the parameters of m and q is by creating a likelihood profile for various values of m as follows:

1. Start with $\hat{m}$ equal to the largest observation.

2. Obtain $\hat{q}$ using (5.3).

3. Calculate the loglikelihood at these values.

4. Increase $\hat{m}$ by 1.

5. Repeat steps $2-4$ until a maximum is found.

Remarks 2. *In Olkin, Petkau and Zidek 1981, they show that if the sample mean is less than or equal to the sample variance then $\hat{m}=\infty$ else there exists a finite m that maximizes (5.4), see [33].*

5.1.4 Negative Binomial

The negative binomial distribution is similar to that of the binomial distribution in the sense that there are two parameters i.e. β and r. In the case of the binomial, the parameter m takes on only positive integer values whereas the parameter r of the negative binomial takes any positive real number. This makes the optimization problem a bit more complex. Recall that $p_k=\binom{k+r-1}{k}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k$

The likelihood function is defined below:

$$\begin{aligned} L(\beta,r) &= \prod_{k=0}^{\infty}\left[\binom{k+r-1}{k}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k\right]^{n_k} \\ &= \prod_{k=0}^{\infty}\binom{k+r-1}{k}^{n_k}\left(\frac{1}{1+\beta}\right)^{rn_k}\left(\frac{\beta}{1+\beta}\right)^{kn_k} \end{aligned}$$

Taking the log of the likelihood function above, we have

$$\begin{aligned} \ell(\beta,r) &= \ln(L(\beta,r)) \\ &= \sum_{k=0}^{\infty}\left[n_k\ln\binom{k+r-1}{k}+rn_k\ln\left(\frac{1}{1+\beta}\right)+kn_k\ln\left(\frac{\beta}{1+\beta}\right)\right] \\ &= \sum_{k=0}^{\infty}\left[n_k\ln\binom{k+r-1}{k}-rn_k\ln(1+\beta)+kn_k\ln(\beta)-kn_k\ln(1+\beta)\right] \end{aligned}$$

Taking the derivative w.r.t. β and setting to zero, we have

$$\ell'_\beta(\beta, r) = -\sum_{k=0}^{\infty} rn_k \cdot \frac{1}{1+\beta} \cdot 1 + \sum_{k=0}^{\infty} kn_k \cdot \frac{1}{\beta} - \sum_{k=0}^{\infty} kn_k \cdot \frac{1}{(1+\beta)} \cdot 1 = 0$$

$$\begin{aligned}
&\implies \frac{-\beta \sum_{k=0}^{\infty} rn_k + (1+\beta) \sum_{k=0}^{\infty} kn_k - \beta \sum_{k=0}^{\infty} kn_k}{\beta(1+\beta)} = 0 \\
&\implies -\beta \sum_{k=0}^{\infty} rn_k + \sum_{k=0}^{\infty} kn_k + \beta \cancel{\sum_{k=0}^{\infty} kn_k} - \beta \cancel{\sum_{k=0}^{\infty} kn_k} = 0 \\
&\implies \beta \sum_{k=0}^{\infty} rn_k = \sum_{k=0}^{\infty} kn_k
\end{aligned}$$

Thus,

$$\begin{aligned}
\hat{\beta} &= \frac{\sum_{k=0}^{\infty} kn_k}{r \sum_{k=0}^{\infty} n_k} \\
&= \frac{\sum_{k=0}^{\infty} kn_k}{rn} \\
&= \frac{\overline{x}}{r}. \qquad (5.5)
\end{aligned}$$

For the second parameter r, we again take the derivative w.r.t r and set it to zero, i.e.

$$\begin{aligned}
&\ell_r'(\beta, r) = 0 \\
\implies &\sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln\left(\frac{(k+r-1)\cdot(k+r-2)\cdot\ldots\cdot(k+r-k)\cdot(r-1)!}{k!(k+r-1-k)!}\right) - \sum_{k=0}^{\infty} n_k \ln(1+\beta) = 0 \\
\implies &\sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln\left(\frac{(k+r-1)\cdot(k+r-2)\cdot\ldots\cdot r\cdot \cancel{(r-1)!}}{k!\cancel{(r-1)!}}\right) - \sum_{k=0}^{\infty} n_k \ln(1+\beta) = 0 \\
\implies &\sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \ln\left[\prod_{m=0}^{k-1}(r+m)\right] - n\ln(1+\beta) = 0 \\
\implies &\sum_{k=0}^{\infty} n_k \frac{\partial}{\partial r} \sum_{m=0}^{k-1} \ln(r+m) - n\ln(1+\beta) = 0 \\
\implies &\sum_{k=1}^{\infty} n_k \sum_{m=0}^{k-1} \frac{1}{\hat{r}+m} = n\ln(1+\beta). \qquad (5.6)
\end{aligned}$$

(5.5) and (5.6) can be solved numerically. When we substitute (5.5) into (5.6) we have

$$\sum_{k=1}^{\infty} n_k \left(\sum_{m=0}^{k-1} \frac{1}{\hat{r}+m}\right) = n\ln\left(1+\frac{\bar{x}}{\hat{r}}\right). \qquad (5.7)$$

Note that if the variance is greater than the mean, then there is a unique solution of (5.7), else it is probably not a good model to use because the sample variance does not exceed the mean [24]. We can solve for $\hat{r}$ numerically using the Newton-Raphson method. A useful starting value for r_0 is the moment based estimator of r.

5.2 The (a, b, 1) Class Estimation

Estimating the parameters for $(a, b, 1)$ follows a similar approach as $(a, b, 0)$.

Recall that the probabilities of a zero-modified random variable N are given by

$$p_k^M = \frac{1-p_0^M}{1-p_0} p_k = (1-p_0^M) p_k^T, \quad k = 1, 2, 3, \ldots$$

where $p_k = \left(a + \frac{b}{k}\right) p_{k-1},\ k = 2, 3, \ldots$ and $0 \leq p_0^M < 1$.

There are 3 parameters to be estimated i.e a, b and p_0^M.

Let

$$p_k^M = \begin{cases} p_0^M & , \text{ if } k = 0 \\ \dfrac{(1 - p_0^M)}{1 - p_0} p_k & , \text{ if } k = 1, 2, 3, \ldots \end{cases}$$

Then the likelihood function is

$$\begin{aligned} L\left(a, b, p_0^M\right) &= (p_0^M)^{n_0} \prod_{k=1}^{\infty} (p_k^M)^{n_k} \\ &= (p_0^M)^{n_0} \prod_{k=1}^{\infty} \left[\frac{(1 - p_0^M)}{1 - p_0} p_k\right]^{n_k}. \end{aligned}$$

Taking the log of the likelihood function above,

$$\begin{aligned} \ell\left(a, b, p_0^M\right) &= \ln[L\left(a, b, p_0^M\right)] \\ &= n_0 \ln\left(p_0^M\right) + \sum_{k=1}^{\infty} n_k \left[\ln\left(1 - p_0^M\right) + \ln(p_k) - \ln\left(1 - p_0\right)\right] \\ &= \underbrace{n_0 \ln\left(p_0^M\right) + \ln\left(1 - p_0^M\right) \sum_{k=1}^{\infty} n_k}_{\ell_0} + \underbrace{\sum_{k=1}^{\infty} n_k \left[\ln(p_k) - \ln\left(1 - p_0\right)\right]}_{\ell_1} \end{aligned}$$

where ℓ_0 depends only on the parameter p_0^M and ℓ_1 is independent of p_0^M and depends on the parameters a and b.

Differentiating ℓ_0 w.r.t p_0^M, we get

$$\begin{aligned} \ell_0'\left(p_0^M\right) &= n_0 \cdot \frac{1}{\hat{p}_0^M} + \sum_{k=1}^{\infty} n_k \cdot \frac{1}{(1 - \hat{p}_0^M)} \cdot (-1) \\ &= \frac{n_0}{\hat{p}_0^M} - \frac{1}{(1 - \hat{p}_0^M)} \sum_{k=1}^{\infty} n_k \end{aligned}$$

Setting $\ell_0'\left(p_0^M\right)$ to zero,

$$\begin{aligned}
\frac{n_0}{\hat{p}_0^M} - \frac{1}{(1-\hat{p}_0^M)}\sum_{k=1}^{\infty} n_k &= 0 \\
\implies \frac{\left(1-\hat{p}_0^M\right)n_0 - \hat{p}_0^M\sum_{k=1}^{\infty} n_k}{\hat{p}_0^M(1-p_0)} &= 0 \\
\implies n_0 - n_0\hat{p}_0^M - \hat{p}_0^M\sum_{k=1}^{\infty} n_k &= 0 \\
\implies n_0 - n_0\hat{p}_0^M - \hat{p}_0^M(n-n_0) &= 0 \quad \text{since } n = \sum_{k=0}^{\infty} n_k \text{ and } n - n_0 = \sum_{k=1}^{\infty} n_k \\
\implies n_0 - n\hat{p}_0^M &= 0 \\
\implies \hat{p}_0^M &= \frac{n_0}{n}
\end{aligned}$$

which is the proportion of observations at zero.

Although ℓ_1 depends on the parameters a and b, we cannot maximize w.r.t them because not all values of a and b produce acceptable probability distributions.

5.2.1 Poisson Distribution

Let $p_k^M = \begin{cases} p_0^M & \text{, if } k = 0 \\ \dfrac{(1-p_0^M)}{1-p_0}p_k & \text{, if } k = 1, 2, 3, \ldots \end{cases}$

where $p_k = \dfrac{e^{-\lambda}\lambda^k}{k!}$ and $p_0 = e^{-\lambda}$.

Thus, the likelihood function is

$$L(\lambda) = \left(p_0^M\right)^{n_0}\prod_{k=1}^{\infty}\left[\frac{\left(1-p_0^M\right)}{1-e^{-\lambda}}\cdot\frac{e^{-\lambda}\lambda^k}{k!}\right]^{n_k}.$$

The log likelihood is

$$\begin{aligned}
\ell(\lambda) &= \ln[L(\lambda)] \\
&= n_0 \ln\left(p_0^M\right) + \sum_{k=1}^{\infty} n_k \left[\ln\left(1-p_0^M\right) - \lambda \ln(e) + k\ln(\lambda) - \ln\left(1-e^{-\lambda}\right) - \ln(k!)\right] \\
&= \underbrace{n_0 \ln\left(p_0^M\right) + \ln\left(1-p_0^M\right) \sum_{k=1}^{\infty} n_k}_{\ell_0} + \underbrace{\sum_{k=1}^{\infty} n_k \left[-\lambda + k\ln(\lambda) - \ln(1-e^{-\lambda}) - \ln(k!)\right]}_{\ell_1}
\end{aligned}$$

Differentiating ℓ_1 w.r.t λ, we have

$$\ell_1'(\lambda) = -\sum_{k=1}^{\infty} n_k + \sum_{k=1}^{\infty} k n_k \cdot \frac{1}{\lambda} - \sum_{k=1}^{\infty} n_k \cdot \frac{1}{1-e^{-\lambda}} \cdot e^{-\lambda}$$

Replacing $\sum_{k=1}^{\infty} n_k$ with $n - n_0$ and equating $\ell_1'(\lambda)$ to zero,

$$\begin{aligned}
&-(n-n_0) + \frac{1}{\lambda} n \cdot \frac{\sum_{k=1}^{\infty} k n_k}{n} - (n-n_0)\frac{e^{-\lambda}}{1-e^{-\lambda}} = 0 \\
&\implies -(n-n_0) + \frac{n\overline{x}}{\lambda} - (n-n_0)\frac{e^{-\lambda}}{1-e^{-\lambda}} = 0, \text{ where } \overline{x} = \frac{\sum_{k=1}^{\infty} k n_k}{n} \text{ (sample mean)} \\
&\implies \frac{-\lambda(1-e^{-\lambda})(n-n_0) + (1-e^{-\lambda})n\overline{x} - \lambda e^{-\lambda}(n-n_0)}{\lambda(1-e^{-\lambda})} = 0 \\
&\implies -\lambda(n-n_0) + \cancel{\lambda e^{-\lambda}(n-n_0)} + (1-e^{-\lambda})n\overline{x} - \cancel{\lambda e^{-\lambda}(n-n_0)} = 0 \\
&\implies (1-e^{-\lambda})\overline{x} = \frac{\lambda(n-n_0)}{n}. \qquad (5.8)
\end{aligned}$$

(5.8) can be solved numerically to obtain $\hat{\lambda}$.

Note that, $\hat{p}_0^M = \frac{n_0}{n}$ and $p_0 = e^{-\lambda}$, (5.8) can be rewritten as

$$\overline{x} = \frac{1-\hat{p}_0^M}{1-p_0}\lambda.$$

The modified moment method can be used to obtain the starting value of λ, see [24, p295].

5.2.2 Geometric Distribution

Let $p_k^M = \begin{cases} p_0^M & , \text{ if } k = 0 \\ \dfrac{(1-p_0^M)}{1-p_0} p_k & , \text{ if } k = 1, 2, 3, \ldots \end{cases}$

where $p_k = \dfrac{\beta^k}{(1+\beta)^{k+1}}$ and $p_0 = \dfrac{1}{1+\beta}$.

Thus, the likelihood function is

$$\begin{aligned} L(\beta) &= \left(p_0^M\right)^{n_0} \prod_{k=1}^{\infty} \left[\frac{(1-p_0^M)}{1-p_0} p_k\right]^{n_k} \\ &= \left(p_0^M\right)^{n_0} \prod_{k=1}^{\infty} \left[\frac{1-p_0^M}{1-\left(\frac{1}{1+\beta}\right)} \cdot \frac{\beta^k}{(1+\beta)^{k+1}}\right] \end{aligned}$$

Then the log likelihood is

$$\begin{aligned} \ell(\beta) &= n_0 \ln\left(p_0^M\right) + \sum_{k=1}^{\infty} n_k \left[\ln\left(1-p_0^M\right) - \ln\left(\frac{\beta}{1+\beta}\right) + k\ln(\beta) - (k+1)\ln(1+\beta)\right] \\ &= \underbrace{n_0 \ln\left(p_0^M\right) + \ln\left(1-p_0^M\right) \sum_{k=1}^{\infty} n_k}_{\ell_0} + \underbrace{\sum_{k=1}^{\infty} n_k \left[(k-1)\ln(\beta) + (1-k-1)\ln(1+\beta)\right]}_{\ell_1} \end{aligned}$$

Differentiating ℓ_1 w.r.t β, we get

$$\ell_1'(\beta) = -\frac{1}{\hat{\beta}} \sum_{k=1}^{\infty} n_k(k-1) - \frac{1}{1+\hat{\beta}} \sum_{k=1}^{\infty} k n_k$$

Equating to zero and simplifying,

$$\begin{aligned} &\frac{(1+\hat{\beta}) \sum_{k=1}^{\infty} n_k(k-1) - \hat{\beta} \sum_{k=1}^{\infty} k n_k}{\hat{\beta}(1+\hat{\beta})} = 0 \\ &\implies (1+\hat{\beta}) \sum_{k=1}^{\infty} k n_k - (1+\hat{\beta}) \sum_{k=1}^{\infty} n_k - \hat{\beta} \sum_{k=1}^{\infty} k n_k = 0 \\ &\implies n \cdot \frac{\sum_{k=1}^{\infty} k n_k}{n} - \sum_{k=1}^{\infty} n_k = \hat{\beta} \sum_{k=1}^{\infty} n_k \end{aligned}$$

But $\overline{x} = \frac{\sum_{k=1}^{\infty} k n_k}{n}$ and $(n - n_0) = \sum_{k=1}^{\infty} n_k$

$$\therefore \frac{n\overline{x} - (n - n_0)}{(n - n_0)} = \hat{\beta}$$
$$\implies \hat{\beta} = \frac{n\overline{x}}{(n - n_0)} - 1.$$

5.2.3 Negative Binomial Distribution

Let $p_k = \binom{k+r-1}{k} \left(\frac{1}{1+\beta}\right)^r \left(\frac{\beta}{1+\beta}\right)^k$ and $p_0 = \frac{1}{(1+\beta)^r}$.

The zero-modified negative binomial (or extended truncated negative binomial) distribution is a bit more complex because we need to estimate 3 parameters. The MLE of p_0^M is already known (i.e $\hat{p}_0^M = \frac{n_0}{n}$), thus it remains to estimate β and r.

Therefore, the likelihood function is

$$L(\beta, r) = \left(p_0^M\right)^{n_0} \prod_{k=1}^{\infty} \left[\frac{(1 - p_0^M)}{1 - p_0} p_k\right]^{n_k} \tag{5.9}$$

Taking the log of the likelihood function,

$$\begin{aligned} \ell(\beta, r) &= \left(p_0^M\right)^{n_0} + \sum_{k=1}^{\infty} n_K \left[\ln\left(1 - p_0^M\right) - \ln(1 - p_0) + \ln p_k\right] \\ &= \underbrace{\left(p_0^M\right)^{n_0} + \ln\left(1 - p_0^M\right) \sum_{k=1}^{\infty} n_K}_{\ell_0} \underbrace{-(n - n_0)\ln(1 - p_0) + \sum_{k=1}^{\infty} n_K \ln(p_k)}_{\ell_1}. \end{aligned} \tag{5.10}$$

ℓ_1 can be rewritten as

$$\begin{aligned} \ell_1 &= -(n - n_0)\ln\left[1 - \frac{1}{(1+\beta)^r}\right] + \sum_{k=1}^{\infty} n_k \ln\left[\binom{k+r-1}{k} \left(\frac{1}{1+\beta}\right)^r \left(\frac{\beta}{1+\beta}\right)^k\right] \\ &= -(n - n_0)\ln\left[1 - \frac{1}{(1+\beta)^r}\right] + \sum_{k=1}^{\infty} n_k \left[\ln\binom{k+r-1}{k} - (r+k)\ln(1+\beta) + k\ln(\beta)\right]. \end{aligned} \tag{5.11}$$

$$\begin{aligned}\frac{\partial \ell_1}{\partial \beta} &= -\frac{1}{1+\beta}\sum_{k=1}^{\infty}(r+k)n_k + \frac{1}{\beta}\sum_{k=1}^{\infty}kn_k - \frac{1}{1-\frac{1}{(1+\beta)r}}\cdot\frac{r}{(1+\beta)^{r+1}}(n-n_0) \\ &= -\frac{r(n-n_0)}{1+\beta} - \frac{n\overline{x}}{1+\beta} + \frac{n\overline{x}}{\beta} - \frac{r}{(1-p_0)}\cdot\frac{p_0}{(1+\beta)}(n-n_0)\end{aligned}$$

Setting the above to zero results in,

$$-\beta r(n-n_0)(1-p_0) - \beta(1-p_0)n\overline{x} + (1+\beta)(1-p_0)n\overline{x} - \beta r p_0(n-n_0) = 0$$

$$\implies (1-p_0)n\overline{x} = \cancel{\beta r p_0(n-n_0)} + \beta r(n-n_0) - \cancel{\beta r p_0(n-n_0)}$$

$$\implies (1-p_0)\overline{x} = \beta r\frac{(n-n_0)}{n}$$

$$\implies \hat{\beta} = \frac{\overline{x}}{r}\left(\frac{1-p_0}{1-\hat{p}_0^M}\right).$$

ℓ_1 can be maximized over the (β, r) plane to obtain the MLEs.

This is achieved numerically using maximization procedures such as excel solver.

5.2.4 Binomial distribution

Let $p_k^M = \begin{cases} p_0^M & \text{, if } k = 0 \\ \dfrac{(1-p_0^M)}{1-p_0}p_k & \text{, if } k = 1, 2, 3, \ldots \end{cases}$

where $p_k = \binom{m}{k}q^k(1-q)^{m-k}$ and $p_0 = (1-q)^m$.

The likelihood function is

$$L(m,q) = \left(p_0^M\right)^{n_0}\prod_{k=1}^{\infty}\left[\frac{1-P_0^M}{1-(1-q)^m}\cdot\binom{m}{k}q^k(1-q)^{m-k}\right]^{n_k}$$

Taking the log of the likelihood function,

$$\begin{aligned}\ell(m,q) &= n_0\ln\left(p_0^M\right) + \sum_{k=1}^{\infty}n_k\ln\left[\frac{1-P_0^M}{1-(1-q)^m}\cdot\binom{m}{k}q^k(1-q)^{m-k}\right] \\ &= \ell_0 + \ell_1\end{aligned}$$

$$\text{where} \quad \ell_0 = n_0 \ln\left(p_0^M\right) + \ln\left(1-p_0^M\right)\sum_{k=1}^{\infty} n_k$$

$$\text{and} \quad \ell_1 = \sum_{k=1}^{\infty} n_k \left[\ln\binom{m}{k} + k\ln(q) + (m-k)\ln(1-q) - \ln\left[1-(1-q)^m\right]\right]$$

ℓ_1 can be rewritten as

$$\begin{aligned}
\ell_1 &= \sum_{k=1}^{\infty} n_k \ln\binom{m}{k} + \ln(q)\cdot n\frac{\sum_{k=1}^{\infty} k n_k}{n} + m\ln(1-q)\sum_{k=1}^{\infty} n_k - \ln(1-q)\frac{\sum_{k=1}^{\infty} k n_k}{n}\cdot n \\
&\quad -\ln\left[1-(1-q)^m\right]\sum_{k=1}^{\infty} n_k \\
&= \sum_{k=1}^{\infty} n_k \ln\binom{m}{k} + n\overline{x}\ln(q) + [m(n-n_0)]\ln(1-q) - n\overline{x}\ln(1-q) - (n-n_0)\ln\left[1-(1-q)^m\right]
\end{aligned}$$

where $\overline{x} = \frac{\sum_{k=1}^{\infty} k n_k}{n}$ and $(n-n_0) = \sum_{k=1}^{\infty} n_k$.

Differentiating w.r.t q,

$$\begin{aligned}
\ell_1'(m,q) &= n\overline{x}\cdot\frac{1}{\hat{q}} + [m(n-n_0)]\cdot\frac{1}{(1-\hat{q})}\cdot(-1) - n\overline{x}\cdot\frac{1}{(1-\hat{q})}\cdot(-1) \\
&\quad -(n-n_0)\cdot\frac{1}{1-(1-\hat{q})^m}\cdot\left[m(1-\hat{q})^{m-1}\right] \\
&= \frac{n\overline{x}}{\hat{q}} - \frac{m(n-n_0)}{1-\hat{q}} + \frac{n\overline{x}}{1-\hat{q}} - \frac{m(n-n_0)(1-\hat{q})^{m-1}}{1-(1-\hat{q})^m}
\end{aligned}$$

Setting the above expression to zero and also replacing $(1-\hat{q})^m$ with p_0,

$$\frac{(1-\hat{q})(1-p_0)n\overline{x} - \hat{q}(1-p_0)m(n-n_0) + \hat{q}(1-p_0)n\overline{x} - \hat{q}(1-\hat{q})m(n-n_0)p_0(1-\hat{q})^{-1}}{\hat{q}(1-\hat{q})(1-p_0)} = 0$$

$$\implies (1-p_0)n\overline{x} - \cancel{\hat{q}(1-p_0)n\overline{x}} - m\hat{q}(1-p_0)(n-n_0) + \cancel{\hat{q}(1-p_0)n\overline{x}} - m\hat{q}(n-n_0)p_0 = 0$$

$$\implies (1-p_0)n\overline{x} = m\hat{q}(n-n_0)\left[1-p_0+p_0\right]$$

$$\implies \overline{x} = m\hat{q}(n-n_0)\frac{(n-n_0)}{(1-p_0)n}$$

$$= m\hat{q}\cdot\frac{1}{1-p_0}\cdot\left(\frac{n}{n}-\frac{n_0}{n}\right)$$

$$= \frac{1-\hat{p}_0^M}{1-p_0}m\hat{q} \qquad \text{since } \frac{n_0}{n} = \hat{p}_0^M.$$

We solve for $\hat{q}$ numerically even if m is known. When m is unknown, it needs to be estimated too, by trying different values of m until the maximum of the likelihood function is obtained.

6 Extension of (a, b, 1) class

6.1 Estimation of (a, b, 2) class

The probabilities of a zero-modified random variable N are given by

$$p_k^M = \begin{cases} p_0^M & , \text{ if } k = 0 \\ p_1^M & , \text{ if } k = 1 \\ \dfrac{1-(p_0^M+p_1^M)}{1-(p_0+p_1)}p_k & , \text{ if } k = 2,3,\ldots \end{cases}$$

The likelihood function is

$$L(a,b,p_0^M,p_1^M) = \left(p_0^M\right)^{n_0}\left(p_1^M\right)^{n_1}\prod_{k=2}^{\infty}\left(p_k^M\right)^{n_k}$$

The log of the likelihood function above is:

$$\begin{aligned} \ell(a,b,p_0^M,p_1^M) &= \ln[L(a,b,p_0^M,p_1^M)] \\ &= n_0\ln(p_0^M)+n_1\ln(p_1^M)+\sum_{k=2}^{\infty}n_k\left[\ln(1-p_0^M-p_1^M)+\ln p_k-\ln(1-p_0-p_1)\right] \\ &= \underbrace{n_0\ln(p_0^M)+n_1\ln(p_1^M)+\ln(1-p_0^M-p_1^M)\sum_{k=2}^{\infty}n_k}_{l_{0,1}}+\underbrace{\sum_{k=2}^{\infty}n_k\left[\ln p_k-\ln(1-p_0-p_1)\right]}_{l_2} \end{aligned}$$

where $l_{0,1}$ depends on two parameters p_0^M and p_1^M.

Differentiating $l_{0,1}$ w.r.t p_0^M and setting to zero,

$$\begin{aligned}
&\frac{n_0}{\hat{p}_0^M} + \frac{1}{1-\hat{p}_0^M - p_1^M}(-1)\sum_{k=2}^{\infty} n_k = 0 \\
\implies & n_0(1-\hat{p}_0^M - p_1^M) - \hat{p}_0^M(n - n_0 - n_1) = 0 \\
\implies & n_0(1-p_1^M) - \hat{p}_0^M(n - \cancel{n_0} - n_1 + \cancel{n_0}) = 0 \\
\implies & \hat{p}_0^M(n-n_1) = n_0(1-p_1^M) \\
\implies & \hat{p}_0^M = \frac{n_0}{n-n_1}(1-p_1^M) \qquad (6.1)
\end{aligned}$$

Differentiating $l_{0,1}$ w.r.t p_1^M and setting to zero,

$$\begin{aligned}
&\frac{n_1}{\hat{p}_1^M} + \frac{1}{1-p_0^M - \hat{p}_1^M}(-1)\sum_{k=2}^{\infty} n_k = 0 \\
\implies & n_1(1-p_0^M - \hat{p}_1^M) - \hat{p}_1^M(n - n_0 - n_1) = 0 \\
\implies & n_1(1-p_0^M) - \hat{p}_1^M(n - n_0 - \cancel{n_1} + \cancel{n_1}) = 0 \\
\implies & \hat{p}_1^M(n-n_0) = n_1(1-p_0^M) \\
\implies & \hat{p}_1^M = \frac{n_1}{n-n_0}(1-p_0^M) \qquad (6.2)
\end{aligned}$$

Substituting (6.1) into (6.2) and vice versa results in

$$\hat{p}_0^M = \frac{n_0}{n} \quad \text{and} \quad \hat{p}_1^M = \frac{n_1}{n}.$$

6.1.1 Poisson Distribution

$$\text{Let } p_k^M = \begin{cases} p_0^M & \text{, if } k = 0 \\ p_1^M & \text{, if } k = 1 \\ \dfrac{1-(p_0^M+p_1^M)}{1-(p_0+p_1)}p_k & \text{, if } k = 2,3,\ldots \end{cases}$$

where $p_k = \dfrac{e^{-\lambda}\lambda^k}{k!}$, $p_0 = e^{-\lambda}$ and $p_1 = \lambda e^{-\lambda}$.

The likelihood function is

$$\begin{aligned} L(\lambda) &= \left(p_0^M\right)^{n_0}\left(p_1^M\right)^{n_1}\prod_{k=2}^{\infty}\left[\frac{1-(p_0^M+p_1^M)}{1-(p_0+p_1)}p_k\right]^{n_k} \\ &= \left(p_0^M\right)^{n_0}\left(p_1^M\right)^{n_1}\prod_{k=2}^{\infty}\left[\frac{1-(p_0^M+p_1^M)}{1-(e^{-\lambda}+\lambda e^{-\lambda})}\cdot\frac{e^{-\lambda}\lambda^k}{k!}\right]^{n_k} \end{aligned}$$

Then the loglikelihood is

$$\begin{aligned} \ell(\lambda) &= \ln[L(\lambda)] \\ &= n_0\ln\left(p_0^M\right)+n_1\ln\left(p_1^M\right) \\ &\quad +\sum_{k=2}^{\infty}n_k\left[\ln(1-p_0^M-p_1^M)-\lambda\ln(e)+k\ln(\lambda)-\ln(1-e^{-\lambda}-\lambda e^{-\lambda})-\ln(k!)\right] \\ &= \underbrace{n_0\ln\left(p_0^M\right)+n_1\ln\left(p_1^M\right)+\ln(1-p_0^M-p_1^M)\sum_{k=2}^{\infty}n_k}_{\ell_{0,1}} \\ &\quad \underbrace{+\sum_{k=2}^{\infty}n_k\left[-\lambda+k\ln(\lambda)-\ln(1-e^{-\lambda}-\lambda e^{-\lambda})-\ln(k!)\right]}_{\ell_2} \end{aligned}$$

Since we know the estimates of $\hat{p}_0^M$ and $\hat{p}_1^M$ (from previous page), we proceed to find the estimate of λ. Differentiating ℓ_2 w.r.t λ and setting to zero,

$$
\begin{aligned}
&-\sum_{k=2}^{\infty} n_k + \frac{1}{\lambda}\sum_{k=2}^{\infty} k n_k - \frac{1}{1-e^{-\lambda}-\lambda e^{-\lambda}}(\cancel{e^{-\lambda}} - \cancel{e^{-\lambda}} + \lambda e^{-\lambda}) \sum_{k=2}^{\infty} n_k = 0 \\
&\implies -(n-n_0-n_1) + \frac{1}{\lambda} n \left(\frac{\sum_{k=2}^{\infty} k n_k}{n} - \frac{n_1}{n} \right) - \frac{\lambda e^{-\lambda}}{1-e^{-\lambda}-\lambda e^{-\lambda}}(n-n_0-n_1) = 0 \\
&\implies \frac{n}{\lambda}\left(\overline{x} - \frac{n_1}{n}\right) = (n-n_0-n_1) + \frac{\lambda e^{-\lambda}}{1-e^{-\lambda}-\lambda e^{-\lambda}}(n-n_0-n_1) \\
&\implies \frac{n\overline{x}}{\lambda} = \frac{n_1}{\lambda} + (n-n_0-n_1)\left(1 + \frac{\lambda e^{-\lambda}}{1-e^{-\lambda}-\lambda e^{-\lambda}}\right) \\
&\implies \overline{x} = \frac{n_1}{n} + \frac{\lambda}{n}(n-n_0-n_1)\left(1 + \frac{p_1}{1-p_0-p_1}\right).
\end{aligned}
$$

Note that, because $\hat{p}_0^M = \frac{n_0}{n}$ and $\hat{p}_1^M = \frac{n_1}{n}$, $\overline{x}$ can be rewritten as

$$
\overline{x} = \hat{p}_1^M + \lambda(1-\hat{p}_0^M - \hat{p}_1^M)\left(1 + \frac{p_1}{1-p_0-p_1}\right).
$$

We solve the above equation numerically to obtain $\hat{\lambda}$.

6.1.2 Geometric distribution:

The likelihood function is

$$
L(p_0^M, p_1^M, \beta) = \left(p_0^M\right)^{n_0} \left(p_1^M\right)^{n_1} \prod_{k=2}^{\infty} \left[\frac{1-p_0^M-p_1^M}{1-p_0-p_1} p_k\right]
$$

where $p_k = \dfrac{\beta^k}{(1+\beta)^{k+1}}$, $p_0 = \dfrac{1}{1+\beta}$ and $p_1 = \dfrac{\beta}{(1+\beta)^2}$

Taking the log of the likelihood function above, we get

$$\begin{aligned}
\ell(p_0^M, p_1^M, \beta) &= n_0 \ln p_0^M + n_1 \ln p_1^M \\
&\quad + \sum_{k=2}^{\infty} n_k \left[\ln(1 - p_0^M - p_1^M) + k\ln\beta - \ln(1 - p_0 - p_1) - (k+1)\ln(1+\beta)\right] \\
&= \underbrace{n_0 \ln p_0^M + n_1 \ln p_1^M + \ln(1 - p_0^M - p_1^M) \sum_{k=2}^{\infty} n_k}_{\ell_{0,1}} \\
&\quad \underbrace{+ \sum_{k=2}^{\infty} n_k \left[k\ln\beta - \ln(1 - p_0 - p_1) - (k+1)\ln(1+\beta)\right]}_{\ell_2}
\end{aligned}$$

but

$$\begin{aligned}
1 - p_0 - p_1 &= 1 - \frac{1}{1+\beta} - \frac{\beta}{(1+\beta)^2} \\
&= \frac{(1+\beta)^2 - (1+\beta) - \beta}{(1+\beta)^2} \\
&= \frac{\beta^2}{(1+\beta)^2}
\end{aligned}$$

Differentiate ℓ_2 w.r.t β and set to zero,

$$\begin{aligned}
&\frac{\partial l_2}{\partial \beta} = \frac{1}{\beta}\sum_{k=2}^{\infty} k n_k - \frac{2}{\beta}\sum_{k=2}^{\infty} n_k + \frac{2}{(1+\beta)}\sum_{k=2}^{\infty} n_k - \frac{1}{(1+\beta)}\sum_{k=2}^{\infty} n_k(k+1) = 0 \\
&\implies (1+\beta)\sum_{k=2}^{\infty} k n_k - 2(1+\beta)\sum_{k=2}^{\infty} n_k + 2\beta\sum_{k=2}^{\infty} n_k - \beta\sum_{k=2}^{\infty} n_k(k+1) = 0 \\
&\implies \sum_{k=2}^{\infty} k n_k + \beta\cancel{\sum_{k=2}^{\infty} k n_k} - 2\sum_{k=2}^{\infty} n_k - \cancel{2\beta\sum_{k=2}^{\infty} n_k} + \cancel{2\beta\sum_{k=2}^{\infty} n_k} - \beta\cancel{\sum_{k=2}^{\infty} k n_k} - \beta\sum_{k=2}^{\infty} n_k = 0 \\
&\implies n\left(\frac{\sum_{k=0}^{\infty} k n_k - n_1}{n}\right) - 2(n - n_0 - n_1) - \beta(n - n_0 - n_1) = 0 \\
&\implies n\overline{x} - n_1 - 2(n - n_0 - n_1) = \beta(n - n_0 - n_1) \\
&\implies \hat{\beta} = \frac{n\overline{x} - n_1}{(n - n_0 - n_1)} - 2
\end{aligned}$$

6.1.3 Negative Binomial Distribution

The part of the loglikelihood relevant to r and β is

$$\ell_2 = \sum_{k=2}^{\infty} n_k \left[\ln p_k - \ln(1 - p_0 - p_1)\right]$$

but $p_k = \binom{k+r-1}{k}\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)^k$, $p_0 = \left(\frac{1}{1+\beta}\right)^r$ and $p_1 = r\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right)$.
Also, $1-p_0-p_1 = 1-\left(\frac{1}{1+\beta}\right)^r - r\left(\frac{1}{1+\beta}\right)^r\left(\frac{\beta}{1+\beta}\right) = \frac{(1+\beta)^{r+1} - (1+\beta+r\beta)}{(1+\beta)^{r+1}}$. Hence

$$\begin{aligned}\ell_2 &= \sum_{k=2}^{\infty} n_k \left[\ln\binom{k+r-1}{k} - r\ln(1+\beta) + k\ln\beta - k\ln(1+\beta)\right] \\ &\quad -(n - n_0 - n_1)\left\{\ln[(1+\beta)^{r+1} - (1+\beta+r\beta)] + (r+1)\ln(1+\beta)\right\}\end{aligned}$$

The MLEs $\hat{\beta}$ and $\hat{r}$ are obtained numerically from maximizing ℓ_2.

6.1.4 Binomial Distribution

Again, we consider the part of the log likelihood function relevant to q and m.

$$\ell_2 = \sum_{k=2}^{\infty} \left[\ln p_k - \ln(1 - p_0 - p_1)\right]$$

where $p_k = \binom{m}{k} q^k (1-q)^{m-k}$, $p_0 = (1-q)^m$ and $p_1 = mq(1-q)^{m-1}$.

Also, $1 - p_0 - p_1 = 1 - (1-q)^m - mq(1-q)^{m-1}$

Hence,

$$\begin{aligned}\ell_2 &= \sum_{k=2}^{\infty} n_k \left[\ln\binom{m}{k} + k\ln(q) + (m-k)\ln(1-q) - \ln\left[1 - (1-q)^m - mq(1-q)^{m-1}\right]\right] \\ &= \sum_{k=2}^{\infty} n_k \ln\binom{m}{k} + (n\overline{x} - n_1)\ln(q) + m(n - n_0 - n_1)\ln(1-q) - (n\overline{x} - n_1)\ln(1-q) \\ &\quad -(n - n_0 - n_1)\ln\left[1 - (1-q)^m - mq(1-q)^{m-1}\right]\end{aligned}$$

Differentiate w.r.t q,

$$\begin{aligned}\frac{\partial \ell_2}{\partial q} &= \frac{(n\overline{x}-n_1)}{q} - \frac{m(n-n_0-n_1)}{1-q} + \frac{(n\overline{x}-n_1)}{1-q} \\ &\quad -\frac{(n-n_0-n_1)}{1-(1-q)^m - mq(1-q)^{m-1}}\left[-m(1-q)^{m-1}(-1) - mq(m-1)(1-q)^{m-2}(-1)\right]\end{aligned}$$

Setting to zero and substituting with p_0 and p_1, we have

$$\begin{aligned}&(1-q)(n\overline{x}-n_1)(1-p_0-p_1) - mq(1-p_0-p_1)(n-n_0-n_1) + q(n\overline{x}-n_1)(1-p_0-p_1) \\ &\qquad -q(1-q)[m(1-q)^{-1}p_0 + (m-1)(1-q)^{-1}p_1](n-n_0-n_1) = 0\end{aligned}$$

$$\begin{aligned}&\implies n\overline{x} - n_1 = mq(n-n_0-n_1) + \frac{q[mp_0+mp_1-p_1]}{(1-p_0-p_1)}(n-n_0-n_1) \\ &\implies \overline{x} = \hat{p}_1^M + mq(1-\hat{p}_0^M-\hat{p}_1^M) + q[mp_0+mp_1-p_1] + \frac{1-\hat{p}_0^M-\hat{p}_1^M}{1-p_0-p_1}.\end{aligned}$$

6.2 Estimation of (a, b, r) class

Estimation of the parameters for (a, b, r) where $r \geq 1$ follows the same general principles used in the $(a, b, 0)$ class. Its probability function is defined as

$$p_k^M = \begin{cases} p_0^M & , \text{ if } k = 0 \\ p_1^M & , \text{ if } k = 1 \\ p_2^M & , \text{ if } k = 2 \\ \vdots & \vdots \\ p_{r-1}^M & , \text{ if } k = r-1 \\ \dfrac{1 - \sum_{k=0}^{r-1} p_k^M}{1 - \sum_{k=0}^{r-1} p_k} p_k & , \text{ if } k = r, r+1, r+2, \ldots \end{cases}$$

The likelihood function is defined as

$$L(a, b, p_0^M, p_1^M, \ldots, p_{r-1}^M) = (p_0^M)^{n_0} (p_1^M)^{n_1} \ldots (p_{r-1}^M)^{n_{r-1}} \prod_{k=r}^{\infty} (p_k^M)^{n_k}$$

Thus, its likelihood is

$$\begin{aligned}
\ell\left(a, b, p_0^M, p_1^M, \ldots, p_{r-1}^M\right) &= \ln\left[L\left(a, b, p_0^M, p_1^M, \ldots, p_{r-1}^M\right)\right] \\
&= n_0 \ln p_0^M + n_1 \ln p_1^M + \ldots + n_{r-1} \ln\left(p_{r-1}^M\right) \\
&\quad + \sum_{k=r}^{\infty} n_k \left[\ln\left(1 - p_0^M - \ldots - p_{r-1}^M\right) + \ln(p_k) - \ln\left(1 - p_0 - \ldots - p_{r-1}\right)\right] \\
&= \underbrace{n_0 \ln p_0^M + n_1 \ln p_1^M + \ldots + n_{r-1} \ln\left(p_{r-1}^M\right) + \ln\left(1 - p_0^M - \ldots - p_{r-1}^M\right)}_{\ell_{0,r-1}} \\
&\quad + \underbrace{\sum_{k=r}^{\infty} n_k \left[\ln p_k - \ln\left(1 - p_0 - \ldots - p_{r-1}\right)\right]}_{\ell_r}
\end{aligned}$$

where $\ell_{0,r-1}$ depends on the parameters $p_0^M, p_1^M, \ldots, p_{r-1}^M$.

Differentiating $\ell_{0,r-1}$ w.r.t. p_0^M and setting to zero,

$$\begin{aligned}
& \frac{n_0}{p_0^M} + \frac{1}{1 - p_0^M - p_1^M - \ldots - p_{r-1}^M}(-1) \sum_{k=r}^{\infty} n_k = 0 \\
\implies \quad & n_0\left(1 - p_0^M - p_1^M - \ldots - p_{r-1}^M\right) - p_0^M (n - n_0 - n_1 - \ldots - n_{r-1}) = 0 \\
\implies \quad & p_0^M = \frac{n_0}{n - n_1 - n_2 - \ldots - n_{r-1}}\left(1 - p_1^M - p_2^M - \ldots - p_{r-1}^M\right).
\end{aligned}$$

Similarly, we also get

$$\begin{aligned}
p_1^M &= \frac{n_1}{n - n_0 - n_2 \ldots - n_{r-1}}\left(1 - p_0^M - p_2^M - \ldots - p_{r-1}^M\right) \\
&\vdots \\
p_{r-1}^M &= \frac{n_{r-1}}{n - n_0 - n_1 - \ldots - n_{r-2}}\left(1 - p_0^M - p_1^M - \ldots - p_{r-2}^M\right).
\end{aligned}$$

6.2.1 (a, b, r) Poisson Distribution

$$\begin{aligned}\ell_r &= \sum_{k=r}^{\infty} n_k \left[\ln\frac{e^{-\lambda}\lambda^k}{k!} - \ln\left(1 - p_0 - p_1 - \ldots - p_{r-1}\right)\right] \\ &= \sum_{k=r}^{\infty} n_k \left[-\lambda + k\ln\lambda - \ln(k!) - \ln\left(1 - p_0 - p_1 - \ldots - p_{r-1}\right)\right]\end{aligned}$$

Differentiating w.r.t the parameter λ, we have

$$-\sum_{k=r}^{\infty} n_k + \frac{1}{\lambda}\sum_{k=r}^{\infty} k n_k - \sum_{k=r}^{\infty} n_k \left[\frac{\frac{\partial}{\partial\lambda}\left(1 - e^{-\lambda} - \lambda e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}\right)}{1 - e^{-\lambda} - \lambda e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}}\right] = 0$$

$$\implies \frac{1}{\lambda}\left[\sum_{k=0}^{\infty} k n_k - n_1 - \ldots - n_{r-1}\right] = \sum_{k=r}^{\infty} n_k \left[1 + \frac{\frac{\partial}{\partial\lambda}\left(1 - e^{-\lambda} - \lambda e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}\right)}{1 - e^{-\lambda} - \lambda e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}}\right]$$

$$\implies \frac{n\overline{x}}{\lambda} = \frac{n_1}{\lambda} + \ldots + \frac{n_{r-1}}{\lambda} + \sum_{k=r}^{\infty} n_k \left[\frac{1 - p_0 - \ldots - p_{r-1} + \frac{\partial}{\partial\lambda}\left(1 - e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}\right)}{1 - p_0 - \ldots - p_{r-1}}\right]$$

$$\overline{x} = p_0^M + \ldots + p_{r-1}^M + \frac{\lambda}{n}\left(n - n_1 - \ldots - n_{r-1}\right)\left[\frac{1 - p_0 - \ldots - p_{r-1} + \frac{\partial}{\partial\lambda}\left(1 - e^{-\lambda} - \ldots - \frac{e^{-\lambda}\lambda^{r-1}}{(r-1)!}\right)}{1 - p_0 - \ldots - p_{r-1}}\right]$$

$$\overline{x} = p_0^M + \ldots + p_{r-1}^M + \lambda(1 - p_1^M - \ldots - p_{r-1}^M)\left[1 + \frac{\frac{\partial}{\partial\lambda}\left(1 - p_0 - \ldots - p_{r-1}\right)}{1 - p_0 - \ldots - p_{r-1}}\right]$$

6.2.2 (a, b, r) Geometric Distribution

$$\ell_r = \sum_{k=r}^{\infty} n_k \left[k\ln\beta - (k+1)\ln(1+\beta) - \ln(1 - p_0 - p_1 - \ldots - p_{r-1})\right]$$

Differentiate w.r.t. β and set to zero,

$$\begin{aligned}
&\frac{1}{\beta}\sum_{k=r}^{\infty} kn_k - \frac{1}{1+\beta}\sum_{k=r}^{\infty} kn_k - \frac{1}{1+\beta}\sum_{k=r}^{\infty} n_k - \frac{\frac{\partial}{\partial\beta}(1-p_0-p_1-\ldots-p_{r-1})}{(1-p_0-p_1-\ldots-p_{r-1})}\sum_{k=r}^{\infty} n_k = 0 \\
\implies\quad &(1+\beta)\sum_{k=r}^{\infty} kn_k - \beta\sum_{k=r}^{\infty} kn_k - \beta\sum_{k=r}^{\infty} n_k - \frac{\beta(1+\beta)\frac{\partial}{\partial\beta}(1-p_0-p_1-\ldots-p_{r-1})}{(1-p_0-p_1-\ldots-p_{r-1})}\sum_{k=r}^{\infty} n_k = 0 \\
\implies\quad &n\overline{x} - n_1 - n_2 - \ldots - n_{r-1} = \beta\sum_{k=r}^{\infty} n_k + \frac{\beta(1+\beta)\frac{\partial}{\partial\beta}(1-p_0-p_1-\ldots-p_{r-1})}{(1-p_0-p_1-\ldots-p_{r-1})}\sum_{k=r}^{\infty} n_k \\
\implies\quad &\overline{x} = p_1^M + p_2^M + \ldots + p_{r-1}^M + \beta(1-p_0^M - p_1^M - \ldots - p_{r-1}^M) \\
&\qquad + \frac{\beta(1+\beta)\frac{\partial}{\partial\beta}(1-p_0-p_1-\ldots-p_{r-1})(1-p_0^M - p_1^M - \ldots - p_{r-1}^M)}{(1-p_0-p_1-\ldots-p_{r-1})}
\end{aligned}$$

6.2.3 (a, b, r) Negative Binomial

$$\ell_r = \sum_{k=r}^{\infty} n_k \left[\ln\binom{k+r-1}{k} - r\ln(1+\beta) + k\ln\beta - k\ln(1+\beta) - \ln(1-p_0-p_1-\ldots-p_{r-1})\right]$$

6.2.4 (a, b, r) Binomial

$$\ell_r = \sum_{k=r}^{\infty} n_k \left[\ln\binom{m}{k} + k\ln(q) + (m-k)\ln(1-q) - \ln\left[1-p_0-p_1-\ldots-p_{r-1}\right]\right]$$

7 Data Analysis

The dataset used in this thesis is a Brazilian vehicle insurance data set, containing policy data based on the AUTOSEG (an acronym for Statistical System for Automobiles) and can be accessed at [4]. The system gives information on the number of vehicles exposed, average premium, average insured amount, number of claims and amounts of indemnities, classified according to the vehicles category, model and year, region, city or zip code, and

the profile of the insured. In this thesis, we only look at column 14 of the dataset, which is the number of robbery claims in 2011. The dataset is called *brvehins1* and consists of $n = 1,965,355$ vehicle insurance policies.

Table 7.1: Descriptive Statistics for the Brazilian Claim Data

Variable: Number of Claims						
	$(\mathbf{a}, \mathbf{b}, \mathbf{0})$	$(\mathbf{a}, \mathbf{b}, \mathbf{1})$	$(\mathbf{a}, \mathbf{b}, \mathbf{2})$	$(\mathbf{a}, \mathbf{b}, \mathbf{3})$	$(\mathbf{a}, \mathbf{b}, \mathbf{4})$	$(\mathbf{a}, \mathbf{b}, \mathbf{5})$
Mean	0.03016	1.40276	2.97689	4.71937	6.23676	7.86435
Median	0	1	2	4	5	6
Mode	0	1	2	3	4	5
Standard Deviation	0.31647	1.65291	3.20910	4.89173	6.33527	7.94047
Variance	0.10015	2.73211	10.29830	23.92900	40.13567	63.05100
Minimum	0	1	2	3	4	5
Maximum	164	164	164	164	164	164
Count	1,965,355	42261	8610	3093	1643	951

From Table 7.1, the variance is greater than the mean in all six models considered (i.e.from $(a, b, 0)$ to $(a, b, 5)$). Also notice from Figure 7.1 that there is an excess of zero, one, etc. Thus the (a, b, r) negative binomial distribution including the geometric distribution should be considered in fitting the Brazilian claim data.

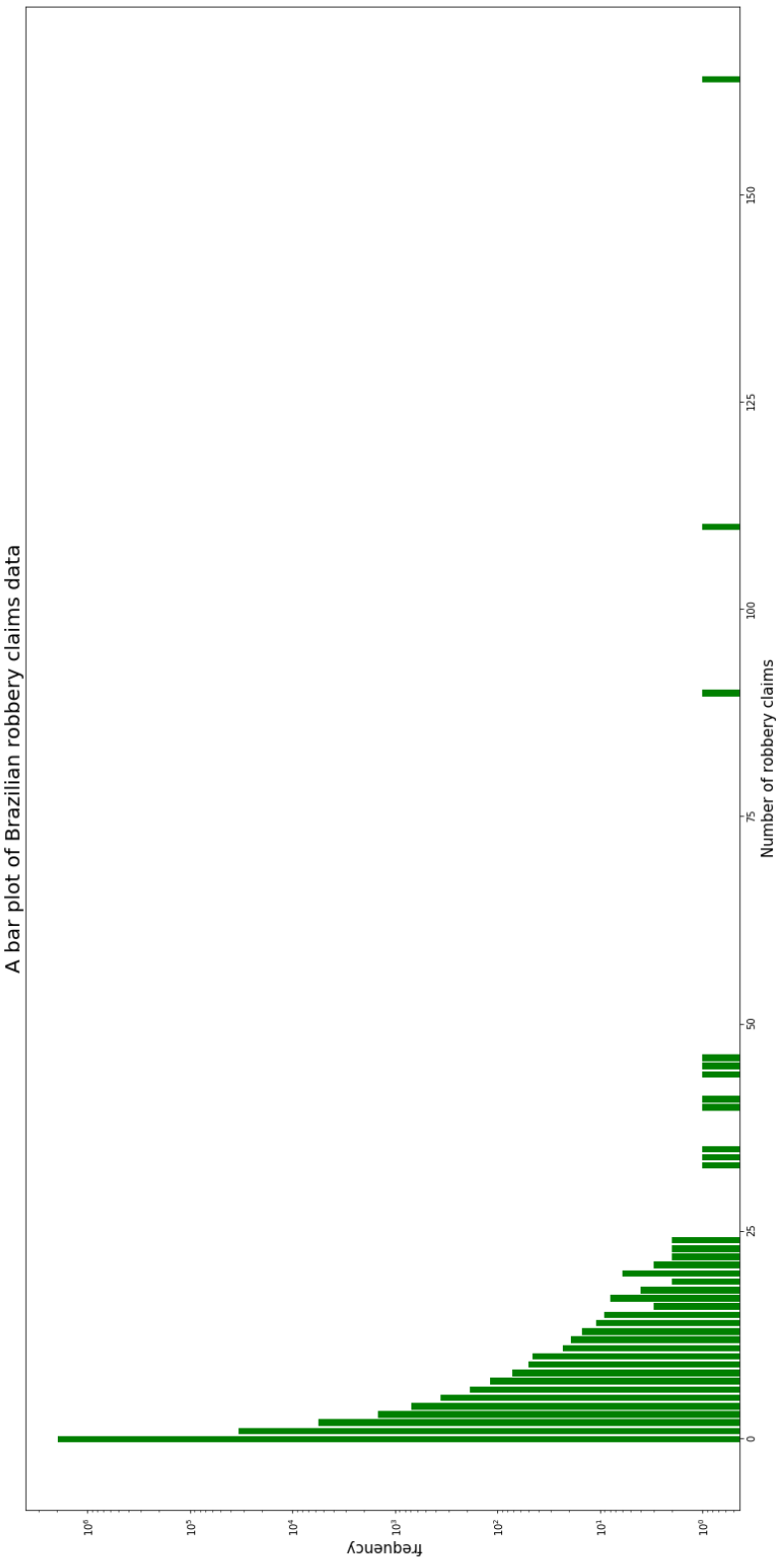


Figure 7.1: A bar plot of Brazilian Robbery claims data

7.0.1 Graphical way to choose the appropriate model for the dataset

Recall that, the recursive formula can be rewritten as

$$k\frac{p_k}{p_{k-1}} = ak + b, \text{ where } k = 1, 2, 3, \ldots \text{ for the } (a, b, 0) \text{ case},$$

$k = 2, 3, \ldots$ for the $(a, b, 1)$ case, $k = 3, 4, \ldots$ for the $(a, b, 2)$ case and so on.

The expression $k\frac{p_k}{p_{k-1}}$ is a linear function of k with slope a. Thus, if we plot $k\frac{p_k}{p_{k-1}}$ against k, we get a straight line that has a positive slope for the negative binomial or geometric distribution, negative slope for the binomial distribution and 0 slope for the Poisson distribution.

Now, we will identify the discrete model that best represents the Brazilian claim data, by plotting the points $\left(k, k\frac{p_k}{p_{k-1}}\right)$. Note that

$$k\frac{p_k}{p_{k-1}} = k\frac{n_k}{n_{k-1}}$$

Table 7.2: Robbery claim data profile from Brazil.

# of claims, $\mathbf{k}$	Frequency, $\mathbf{n_k}$	$\mathbf{k}\frac{\mathbf{n_k}}{\mathbf{n_{k-1}}}$
0	$1,923,094$	NA
1	$33,651$	0.0175
2	$5,517$	0.3279
3	$1,450$	0.7885
4	692	1.9090
5	360	2.6012
6	186	3.1000
7	119	4.4785
8	71	4.7731
9	50	6.3380
10	45	9.0000
11	23	5.6222
12	19	9.9130
13	15	10.2632
14	11	10.2667
15	9	12.2727
16	3	5.3333
17	8	45.3333
18	4	9.0000
19	2	9.5000
20	6	60.0000
21	3	10.5000
22	2	14.6667
23	2	23.0000
24	2	24.0000
33	1	NA
34	1	34.0000
35	1	35.0000
40	1	NA
41	1	41.0000
44	1	NA
45	1	45.0000
46	1	46.0000
90	1	NA
110	1	NA
164	1	NA

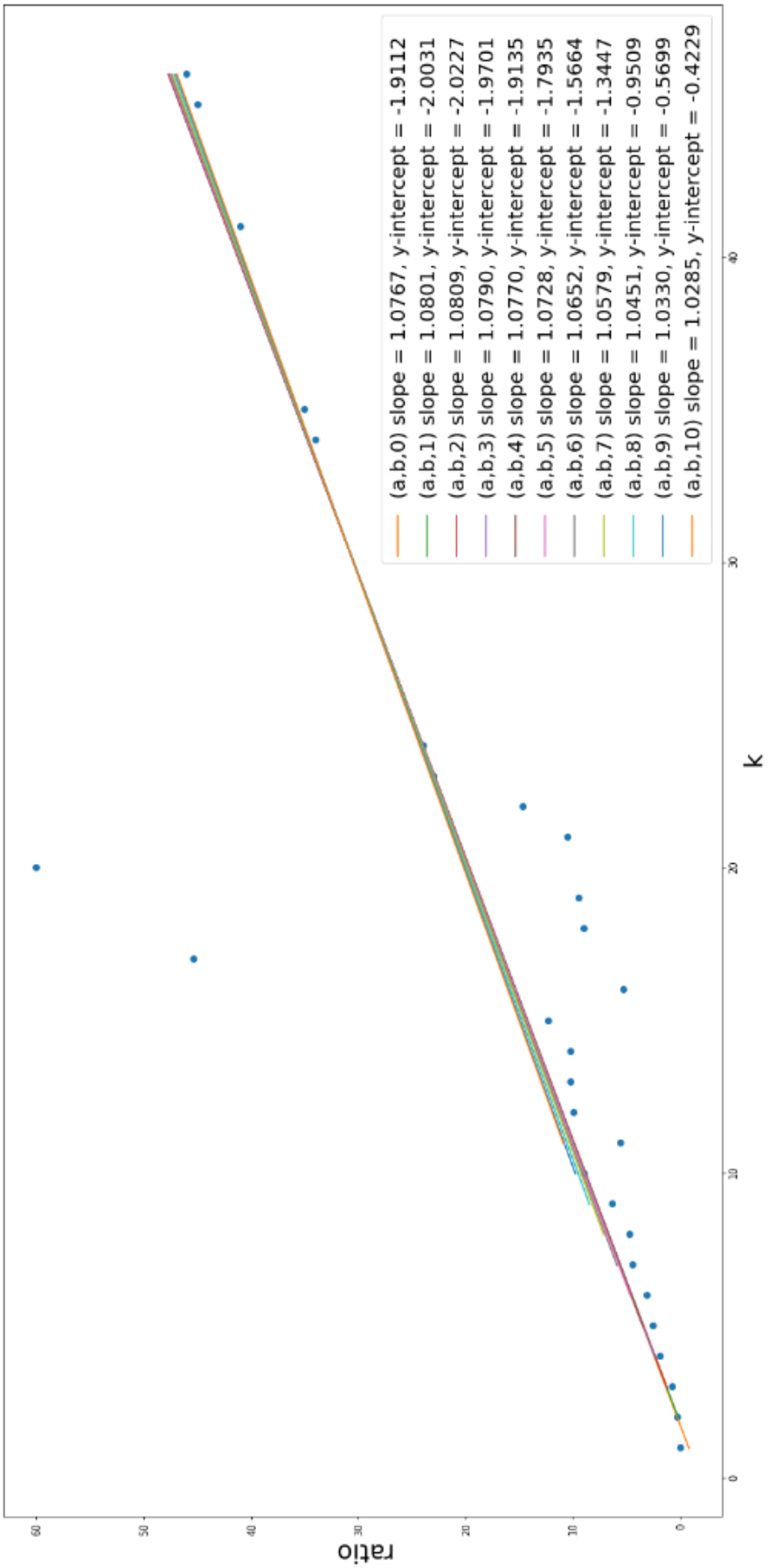


Figure 7.2: Plot of the ratio $\mathrm{kn_k}/\mathrm{n_{k-1}}$ against k

Figure 7.2 plots the value of the quantity of interest $\left(\text{i.e. } k\frac{p_k}{p_{k-1}}\right)$ against the number of robbery claims, k. The quantity of interest from the graph appears approximately linear except at points $k = 17$ and 20. The reliability of the quantities as k increases reduces, since the number of observations (i.e frequency) becomes small and the variability of the results grow, thereby proving a weakness of this approach. In addition, it can be seen from the graph that the points on the left are more reliable than those on the right due to the larger number of observations. Also from the graph, the slope is positive for $(a, b, 0)$ to $(a, b, 10)$ classes. This suggests that the negative binomial distribution including the geometric distribution should be considered in fitting the data. The binomial distribution is probably not a good choice since there is no evidence of a negative slope. Also, the Poisson distribution requires a slope of zero, thus it is probably not a suitable model to consider for this data. So we will fit both the (a, b, r) negative binomial and geometric distributions and then conduct a more formal analysis to choose between them. We will also fit the Poisson and Binomial distributions too, just to confirm our claim that it might not be the best models to fit this data.

8 Results

In this chapter, we fit the various discrete models discussed above to the robbery claims data from Brazil using the maximum likelihood estimate. We then analyze which of the models gives an approximately good fit. We also use the Akaike Information Criterion to choose a better model for this dataset.

8.1 (a, b, 0) Fit for the Four Discrete Distributions using MLE

From Table 8.1 below, the robbery claims data as well as fitted Poisson, Geometric, Negative Binomial and Binomial distributions are given. It is clear from the table that the negative binomial probabilities produce expected counts or frequencies that are "relatively" closer to the observed values. Note that the expected counts are found by multiplying the sample size (i.e. $n = 1,965,355$) by the probability assigned by the model. In addition, based on the values of the loglikelihood in all four models, the negative binomial model seems to be the best fit since it has the highest loglikelihood i.e. $-237,918.71$. The second best fit is the geometric distribution followed by the Poisson distribution. The binomial distribution clearly is a really bad fit for this dataset since the variance is greater than the mean.

Table 8.1: Fitted (a, b, 0) distributions to Brazilian vehicle robbery data

# of claims	Frequency	Poisson	Geometric	Neg Binomial	Binomial
0	1,923,094	1,906,958.15	1,907,808.80	1,923,174.05	1,906,953.05
1	33,651	57,520.55	55,861.23	31,006.15	57,530.60
2	5,517	867.51	1,635.63	7,466.64	862.71
3	1,450	8.72	47.89	2,357.28	8.57
4	692	0.07	1.40	832.49	0.06
5	360	0.00	0.04	312.70	0.00
6	186	0.00	0.00	122.14	0.00
7	119	0.00	0.00	49.02	0.00
8	71	0.00	0.00	20.06	0.00
9	50	0.00	0.00	8.34	0.00
10	45	0.00	0.00	3.51	0.00
11	23	0.00	0.00	1.49	0.00
12	19	0.00	0.00	0.64	0.00
13	15	0.00	0.00	0.27	0.00
14	11	0.00	0.00	0.12	0.00
15	9	0.00	0.00	0.05	0.00
16	3	0.00	0.00	0.02	0.00
17	8	0.00	0.00	0.01	0.00
18	4	0.00	0.00	0.00	0.00
19	2	0.00	0.00	0.00	0.00
20	6	0.00	0.00	0.00	0.00
21	3	0.00	0.00	0.00	0.00
22	2	0.00	0.00	0.00	0.00
23	2	0.00	0.00	0.00	0.00
24	2	0.00	0.00	0.00	0.00
33	1	0.00	0.00	0.00	0.00
34	1	0.00	0.00	0.00	0.00
35	1	0.00	0.00	0.00	0.00
40	1	0.00	0.00	0.00	0.00
41	1	0.00	0.00	0.00	0.00
44	1	0.00	0.00	0.00	0.00
45	1	0.00	0.00	0.00	0.00
46	1	0.00	0.00	0.00	0.00
90	1	0.00	0.00	0.00	0.00
110	1	0.00	0.00	0.00	0.00
164	1	0.00	0.00	0.00	0.00
Parameters		$\lambda : 0.03016$	$\beta : 0.03016$	$\beta : 0.87091$ $r : 0.03463$	$q : 0.00018$ $m : 170$
Loglikelihood		$-286,706.18$	$-267,720.76$	$-237,918.71$	$-287,198.93$

8.2 (a, b, 1) Fit for the Four Discrete Distributions using MLE

The dataset as well as the fitted zero-modified (ZM) Poisson, ZM geometric, ZM negative binomial and ZM binomial distributions are given in Table 8.2 below. It is clear from the dataset that there is an excess of zero, so the ZM geometric fit appears to fit better than the $(a, b, 0)$ geometric fit in Table 8.1 above since it has the highest loglikelihood amongst the rest of the distributions. But it seems the ZM negative binomial fit does not fit well as compared to the (a,b,0) negative binomial fit in the table above. Again the ZM Poisson and ZM binomial distributions are the worst fits in this class.

Table 8.2: Zero-modified fitted distributions to Brazilian vehicle robbery data

# of claims	Frequency	ZM Poisson	ZM Geometric	ZM Neg Binomial	ZM Binomial
0	1,923,094	1,923,094.00	1,923,094.00	1,923,094.00	1,923,094.00
1	33,651	41,386.39	30,127.06	18,896.10	41,391.26
2	5,517	862.50	8,650.06	9,391.92	857.84
3	1,450	11.98	2,483.60	5,299.79	11.78
4	692	0.12	713.09	3,168.87	0.12
5	360	0.00	204.74	1,958.69	0.00
6	186	0.00	58.79	1237.02	0.00
7	119	0.00	16.88	793.14	0.00
8	71	0.00	4.85	514.25	0.00
9	50	0.00	1.39	336.31	0.00
10	45	0.00	0.40	221.44	0.00
11	23	0.00	0.11	146.63	0.00
12	19	0.00	0.03	97.53	0.00
13	15	0.00	0.01	65.13	0.00
14	11	0.00	0.00	43.64	0.00
15	9	0.00	0.00	29.32	0.00
16	3	0.00	0.00	19.75	0.00
17	8	0.00	0.00	13.33	0.00
18	4	0.00	0.00	9.02	0.00
19	2	0.00	0.00	6.11	0.00
20	6	0.00	0.00	4.15	0.00
21	3	0.00	0.00	2.82	0.00
22	2	0.00	0.00	1.92	0.00
23	2	0.00	0.00	1.31	0.00
24	2	0.00	0.00	0.89	0.00
33	1	0.00	0.00	0.03	0.00
34	1	0.00	0.00	0.02	0.00
35	1	0.00	0.00	0.01	0.00
40	1	0.00	0.00	0.00	0.00
41	1	0.00	0.00	0.00	0.00
44	1	0.00	0.00	0.00	0.00
45	1	0.00	0.00	0.00	0.00
46	1	0.00	0.00	0.00	0.00
90	1	0.00	0.00	0.00	0.00
110	1	0.00	0.00	0.00	0.00
164	1	0.00	0.00	0.00	0.00
Parameters		p_0^M : 0.97850 λ : 0.04168	p_0^M : 0.97850 β : 0.40276	p_0^M : 0.97850 β : 2.32026 r : 0.42248	p_0^M : 0.97850 q : 0.00024 m : 171
Loglikelihood		$-\infty$	$-238,761.04$	$-248,030.52$	$-\infty$

8.3 (a, b, 2) Fit for the Four Discrete Distributions using MLE

In Table 8.3 below, we fit the brazilian claim data to the $(a, b, 2)$ class of discrete distributions with focus on only the four distributions of interest. The $(a, b, 2)$ negative binomial (or Z1M negative binomial as named in the table below) seems to give the best fit in this class with the highest loglikelihood of $-236,633.74$. The second best fit is the $(a, b, 2)$ geometric distribution (also called Z1M geometric) with a loglikelihood of $-236,753.09$. Again, the Poisson and binomial fits seems to be very poor fits for the dataset (which supports our claim in the previous section).

Table 8.3: Fitted (a,b,2) distributions to Brazilian vehicle robbery data

# of claims	Frequency	Z1M Poisson	Z1M Geometric	Z1M Negative Binomial	Z1M Binomial
0	1,923,094	1,923,094.00	1,923,094.00	1,923,094.00	1,923,094.00
1	33,651	33,651.00	33,651.00	33,651.00	33,651.00
2	5,517	8,601.19	4,355.33	4,402.19	8,601.34
3	1,450	8.80	2,152.21	2,089.00	8.65
4	692	0.01	1,063.52	1,032.54	0.01
5	360	0.00	525.54	522.59	0.00
6	186	0.00	259.70	268.62	0.00
7	119	0.00	128.33	139.59	0.00
8	71	0.00	63.42	73.13	0.00
9	50	0.00	31.34	38.55	0.00
10	45	0.00	15.49	20.42	0.00
11	23	0.00	7.65	10.86	0.00
12	19	0.00	3.78	5.80	0.00
13	15	0.00	1.87	3.10	0.00
14	11	0.00	0.92	1.67	0.00
15	9	0.00	0.46	0.90	0.00
16	3	0.00	0.23	0.48	0.00
17	8	0.00	0.11	0.26	0.00
18	4	0.00	0.06	0.14	0.00
19	2	0.00	0.03	0.08	0.00
20	6	0.00	0.01	0.04	0.00
21	3	0.00	0.01	0.02	0.00
22	2	0.00	0.00	0.01	0.00
23	2	0.00	0.00	0.01	0.00
24	2	0.00	0.00	0.00	0.00
33	1	0.00	0.00	0.00	0.00
34	1	0.00	0.00	0.00	0.00
35	1	0.00	0.00	0.00	0.00
40	1	0.00	0.00	0.00	0.00
41	1	0.00	0.00	0.00	0.00
44	1	0.00	0.00	0.00	0.00
45	1	0.00	0.00	0.00	0.00
46	1	0.00	0.00	0.00	0.00
90	1	0.00	0.00	0.00	0.00
110	1	0.00	0.00	0.00	0.00
164	1	0.00	0.00	0.00	0.00
Parameters		p_0^M : 0.97850 p_1^M : 0.01712 λ : 0.00307	p_0^M : 0.97850 p_1^M : 0.01712 β : 0.97689	p_0^M : 0.97850 p_1^M : 0.01712 β : 1.23958 r : 0.57208	p_0^M : 0.97850 p_1^M : 0.01712 q : 0.00002 m : 171
Loglikelihood		$-\infty$	$-236,753.09$	$-236,633.74$	$-\infty$

8.4 (a, b, 3) Fit for the Four Discrete Distributions using MLE

Again, in Table 8.4 below, we fit the brazilian claim data to $(a, b, 3)$ Poisson, geometric, negative binomial and binomial distributions. So far it is clear that the Poisson and binomial distributions gives a very poor fit to the data. The $(a, b, 2)$ (or Z12M) negative binomial fit seems to have the highest loglikelihood of $-236.339.31$ followed by the $(a, b, 2)$ (or Z12M) geometric fit with a loglikelihood of $-236, 339.31$. So far we can clearly see the two potential models of choice is the negative binomial and geometric distributions. Thus in the next section, we focus on just these two distributions to determine the best model.

Table 8.4: Fitted (a,b,3) distributions to Brazilian vehicle robbery data

# of claims	Frequency	Z12M Poisson	Z12M Geometric	Z12M Negative Binomial	Z12M Binomial
0	1,923,094	1,923,094.00	1,923,094.00	1,923,094.00	1,923,094.00
1	33,651	33,651.00	33,651.00	33,651.00	33,651.00
2	5,517	5,517.00	5,517.00	5,517.00	5,517.00
3	1,450	3,092.12	686.86	1,183.90	3,091.83
4	692	0.07	534.33	763.28	0.06
5	360	0.00	415.67	472.11	0.00
6	186	0.00	323.36	283.78	0.00
7	119	0.00	251.55	167.04	0.00
8	71	0.00	195.69	96.76	0.00
9	50	0.00	152.23	55.35	0.00
10	45	0.00	118.43	31.34	0.00
11	23	0.00	92.13	17.59	0.00
12	19	0.00	71.67	9.81	0.00
13	15	0.00	55.75	5.43	0.00
14	11	0.00	43.37	3.00	0.00
15	9	0.00	33.74	1.64	0.00
16	3	0.00	26.25	0.90	0.00
17	8	0.00	20.42	0.49	0.00
18	4	0.00	15.88	0.27	0.00
19	2	0.00	12.36	0.14	0.00
20	6	0.00	9.61	0.08	0.00
21	3	0.00	7.48	0.04	0.00
22	2	0.00	5.82	0.02	0.00
23	2	0.00	4.53	0.01	0.00
24	2	0.00	3.52	0.01	0.00
33	1	0.00	0.37	0.00	0.00
34	1	0.00	0.29	0.00	0.00
35	1	0.00	0.22	0.00	0.00
40	1	0.00	0.06	0.00	0.00
41	1	0.00	0.05	0.00	0.00
44	1	0.00	0.02	0.00	0.00
45	1	0.00	0.02	0.00	0.00
46	1	0.00	0.01	0.00	0.00
90	1	0.00	0.00	0.00	0.00
110	1	0.00	0.00	0.00	0.00
164	1	0.00	0.00	0.00	0.00
Parameters		p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 λ : 0.00009	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 β : 3.50307	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 β : 1.05690 r : 2.01886	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 q : 0.0000005 m : 171
Loglikelihood		$-\infty$	$-236,854.81$	$-236,339.31$	$-\infty$

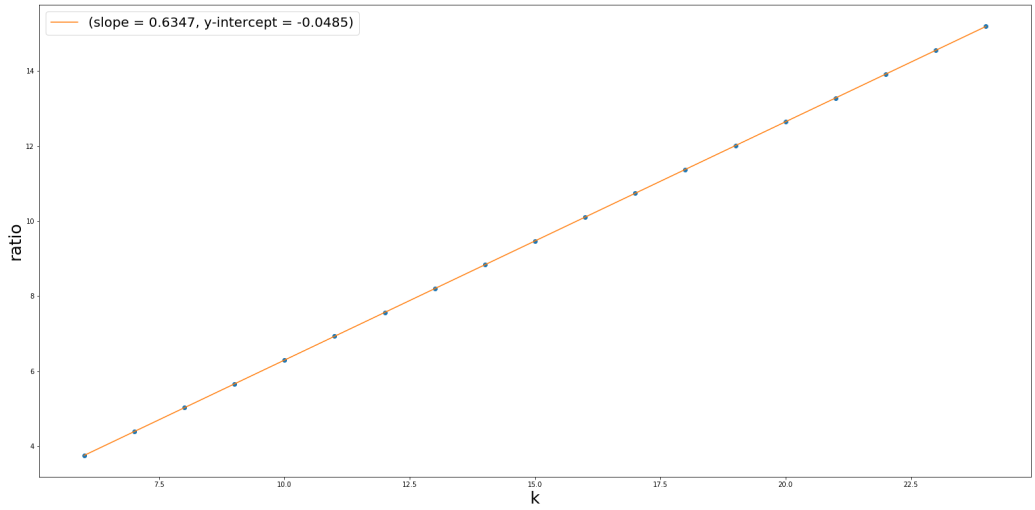
8.5 (a, b, r) Negative Binomial and Geometric Fit using MLE where r = 4 and 5

Table 8.5 gives the $(a, b, 4), (a, b, 5)$ negative binomial and geometric fits of the Brazilian robbery claim dataset. It can be seen from the table that the $(a, b, 5)$ (or Z1234M) negative binomial distribution appears to have a better fit since it has the largest loglikelihood i.e. $-236,217.09$. The second best fit is the $(a, b, 4)$(or Z123M) negative binomial distribution which has the second largest loglikelihood. The next best fit is the $(a, b, 5)$ (or Z1234M) geometric distribution and finally the $(a, b, 4)$ (or Z123M) geometric distribution. So far we have seen that as the value of r increases, the fit gets a little bit better, however the number of parameters increases making the model more sophisticated. So at this stage, we implore the use of a statistical tool in selecting the best model (which gives a balance between a good fit and the number of parameters involved) amongst the ones discussed so far. This brings us to the Akaike Information Criterion used in best model selection. This is discussed in the next section.

Table 8.5: (a, b, r) fitted negative binomial and geometric distributions to Brazilian data, where r=4, 5

# of claims	Frequency	Z123M Neg Binom	Z1234M Neg Binom	Z123M Geometric	Z1234M Geometric
0	1,923,094	1,923,094.00	1,923,094.00	1,923,094.00	1923094.00
1	33,651	33,651.00	33,651.00	33,651.00	33,651.00
2	5,517	5,517.00	5,517.00	5,517.00	5,517.00
3	1,450	1,450.00	1,450.00	1,450.00	1,450.00
4	692	510.15	692.00	196.54	692.00
5	360	369.94	353.74	173.03	63.83
6	186	258.56	221.66	152.33	59.55
7	119	175.87	139.15	134.11	55.55
8	71	117.15	87.48	118.07	51.82
9	50	76.76	55.05	103.94	48.34
10	45	49.62	34.67	91.51	45.10
11	23	31.72	21.85	80.56	42.07
12	19	20.09	13.78	70.93	39.25
13	15	12.62	8.70	62.44	36.61
14	11	7.88	5.49	54.97	34.16
15	9	4.89	3.47	48.40	31.86
16	3	3.01	2.19	42.61	29.72
17	8	1.85	1.38	37.51	27.73
18	4	1.13	0.87	33.02	25.87
19	2	0.69	0.55	29.07	24.13
20	6	0.42	0.35	25.60	22.51
21	3	0.25	0.22	22.53	21.00
22	2	0.15	0.14	19.84	19.59
23	2	0.09	0.09	17.46	18.28
24	2	0.06	0.06	15.38	17.05
33	1	0.00	0.00	4.88	9.12
34	1	0.00	0.00	4.30	8.51
35	1	0.00	0.00	3.79	7.94
40	1	0.00	0.00	2.00	5.61
41	1	0.00	0.00	1.76	5.23
44	1	0.00	0.00	1.20	4.25
45	1	0.00	0.00	1.06	3.96
46	1	0.00	0.00	0.93	3.70
90	1	0.00	0.00	0.00	0.17
110	1	0.00	0.00	0.00	0.04
164	1	0.00	0.00	0.00	0.00
Parameters		p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 p_3^M : 0.00074 β : 1.31359 r : 2.38600	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 p_3^M : 0.00074 p_4^M : 0.00035 β : 1.73752 r : 0.92357	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 p_3^M : 0.00074 β : 7.35971	p_0^M : 0.97850 p_1^M : 0.01712 p_2^M : 0.00281 p_3^M : 0.00074 p_4^M : 0.00035 β : 13.8980
Loglikelihood		$-236,271.21$	$-236,217.09$	$-237,029.48$	$-236,975.10$

Figure 8.1: Plot of the ratio $\mathrm{kn_k/n_{k-1}}$ against k for $(\mathrm{a}, \mathrm{b}, 5)$ negative binomial



From figure 8.1 above, the slope $a = 0.6347$ and the y-intercept $b = -0.0485$.

8.6 Model Selection using Akaike Information Criterion (AIC)

The AIC is one of the most popular tools in statistical modeling and was introduced in 1973 by Hirotugu Akaike as an extension to the maximum likelihood principle, see [23]. The basic notion of the AIC is that by continually adding parameters to a model, we would always fit a little bit better, but we are also trading off against overfitting and actually loosing inofrmation about the real underlying pattern. Thus AIC represents a tradeoff between the number of parameters you add and the incremental amount of error that you

are accounting for by doing so. Hence, the AIC will be employed to choose the best model for the Brazilian dataset. We take a look at the definition of AIC below. For more details see [1] and [9, page 60].

Definition 4. *Suppose there is a statistical model of some data. Let p represent the number of parameters in the fitted model and ℓ represent the maximum loglikelihood. Then the AIC value of the model is*

$$AIC = -2\ell + 2p.$$

We now find the AIC of the discrete models discussed above.

Table 8.6: Results of AIC Analysis

Model	loglikelihood (ℓ)	Number of Parameters $(\mathbf{p})$	AIC
$(a, b, 0)$ Poisson	$-286,706.18$	1	$573,414.36$
$(a, b, 0)$ Geometric	$-267,720.76$	1	$535,443.52$
$(a, b, 0)$ Negative Binomial	$-237,918.71$	2	$475,841.42$
$(a, b, 0)$ Binomial	$-287,198.93$	2	$574,401.86$
ZM Poisson	$-\infty$	2	∞
ZM Geometric	$-238,761.04$	2	$477,526.08$
ZM Negative Binomial	$-248,030.52$	3	$496,067.04$
ZM Binomial	$-\infty$	3	∞
Z1M Poisson	$-\infty$	3	∞
Z1M Geometric	$-236,753.09$	3	$473,512.18$
Z1M Negative Binomial	$-236,633.74$	4	$473,275.48$
Z1M Binomial	$-\infty$	4	∞
Z12M Poisson	$-\infty$	4	∞
Z12M Geometric	$-236,854.81$	4	$473,717.62$
Z12M Negative binomial	$-236,339.31$	5	$472,688.62$
Z12M Binomial	$-\infty$	5	∞
Z123M Negative Binomial	$-236,271.21$	6	$472,554.42$
Z1234M Negative Binomial	$-236,217.09$	7	$472,448.18$
Z123M Geometric	$-237,029.48$	5	$474,068.96$
Z1234M Geometric	$-236,975.10$	6	$473,962.20$

From Table 8.6, the model with the better fit has the lowest AIC value. Hence the best model for the 2011 Brazilian Robbery claims dataset is the $(a, b, 5)$ or Z1234M Negative Binomial fit. Note that the AIC values of the Poisson and Binomial distributions are positive infinity thereby these models are the very least models to consider since they have the largest AIC and do not fit well to the dataset.

9 Conclusion

In this study, applications of Panjer's class of discrete distributions have been studied. We looked at the $(a, b, 0)$ and $(a, b, 1)$ class of distributions and derived how each of the parameters are estimated. In addition, we extended the parameter estimation to (a, b, r) by adjusting the notion of $(a, b, 1)$. Then we applied the study of the various models discussed in modeling the Brazilian claim count data. We fit the data to (a, b, r) class of discrete distributions where $0 \leq r \leq 5$. The best fitted model was the $(a, b, 5)$ or Z1234M negative binomial distribution since it had the lowest AIC value. The estimated parameters of the $(a, b, 5)$ negative binomial model using the Brazilian dataset are $a = 0.6347$ and $b = -0.0485$. This study shows we should look beyond $(a, b, 1)$ class of distributions when we encounter a dataset that has an excess in the first r claims, where $r > 1$. When encountered with such a dataset, the (a, b, r) class of distributions should be considered in model selection.

10 Future Work

In this thesis we only looked at the Poisson, geometric, negative binomial and binomial distributions in (a, b, r) class of discrete distributions. But we know in the class of $(a, b, 1)$ discrete distributions, there include two additional distributions i.e. the extended truncated negative binomial (ETNB) and logarithmic distributions. In continued studies, it would be interesting to look at the characteristics or behaviour of these two distributions in the (a, b, r) class aside the four main distributions discussed above.

References

[1] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[2] Katrien Antonio and Emiliano A Valdez. Statistical concepts of a priori and a posteriori risk classification in insurance. *AStA Advances in Statistical Analysis*, 96(2):187–224, 2012.

[3] Katrien— FREES ANTONIO, Edward W Frees, Emiliano A Valdez, et al. A multilevel analysis of intercompany claim counts. *ASTIN Bulletin-Actuarial Studies in non LifeInsurance*, 40(1):151, 2010.

[4] AUTOSEG. brvehins1, 2011. data retrieved from Brazil AUTOSEG system, `http://www2.susep.gov.br/menuestatistica/Autoseg/principal.aspx`.

[5] Gregori Baetschmann and Rainer Winkelmann. A dynamic hurdle model for zero-inflated count data: with an application to health care utilization. *University of Zurich, Department of Economics, Working Paper*, (151), 2014.

[6] David Bahnemann. Distributions for actuaries. *CAS monograph series*, 2, 2015.

[7] Noureddine Benlagha, Lanouar Charfeddine, and Imen Karaa. Modelling accident occurrence in car insurance implementation on tunisian data. *Asian-African Journal of Economics and Econometrics*, 12(2):395–406, 2012.

[8] Jean-Philippe Boucher, Michel Denuit, and Montserrat Guillén. Risk classification for claim counts: a comparative analysis of various zeroinflated mixed poisson and hurdle models. *North American Actuarial Journal*, 11(4):110–131, 2007.

[9] Kenneth P Burnham and David R Anderson. A practical information-theoretic approach. *Model selection and multimodel inference, 2nd ed. Springer, New York*, 2, 2002.

[10] A Colin Cameron and Pravin K Trivedi. *Regression analysis of count data*, volume 53. Cambridge university press, 2013.

[11] AC Cameron and PK Trivedi. Essentials of count data regression (chapter 15). *A Companion to Theoretical Econometrics. Malden, MA: Blackwell Publishing Ltd*, 1999.

[12] Piet De Jong, Gillian Z Heller, et al. Generalized linear models for insurance data. *Cambridge Books*, 2008.

[13] Michel Denuit, Xavier Maréchal, Sandra Pitrebois, and Jean-François Walhin. *Actuarial modelling of claim counts: Risk classification, credibility and bonus-malus systems*. John Wiley & Sons, 2007.

[14] Marcel B. Finan. An introductory guide in the construction of actuarial models: a preparation for the actuarial exam c/4. *Arkansas Tech University. Available online at the address* `http://faculty.atu.edu/mfinan/actuarieshall/CGUIDE.pdf` *[accessed 11-Feb-2018]*, 2017.

[15] Christian Gourieroux and Joann Jasiak. Dynamic factor models. *Econometric Reviews*, 20(4):385–424, 2001.

[16] Christian Gourieroux, Alain Monfort, and Alain Trognon. Pseudo maximum likelihood methods: Applications to poisson models. *Econometrica: Journal of the Econometric Society*, pages 701–720, 1984.

[17] Christian Gourieroux, Alain Monfort, and Alain Trognon. Pseudo maximum likelihood methods: Theory. *Econometrica: Journal of the Econometric Society*, pages 681–700, 1984.

[18] William H Greene. Accounting for excess zeros and sample selection in poisson and negative binomial regression models. 1994.

[19] Shiferaw Gurmu. Generalized hurdle count data regression models. *Economics Letters*, 58(3):263–268, 1998.

[20] Jerry A Hausman, Bronwyn H Hall, and Zvi Griliches. Econometric models for count data with an application to the patents-r&d relationship. Technical report, national bureau of economic research, 1984.

[21] David C Heilbron. Zero-altered and other regression models for count data with added zeros. *Biometrical Journal*, 36(5):531–547, 1994.

[22] Klaus Th Hess, Anett Liewald, and Klaus D Schmidt. An extension of panjer's recursion. *ASTIN Bulletin: The Journal of the IAA*, 32(2):283–297, 2002.

[23] Akaike Hirotogu. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory. Budapest: Akademiai Kado*, 1973.

[24] Stuart A Klugman, Harry H Panjer, and Gordon E Willmot. *Loss models: from data to decisions*, volume 715. John Wiley & Sons, 2012.

[25] Diane Lambert. Zero-inflated poisson regression, with an application to defects in manufacturing. *Technometrics*, 34(1):1–14, 1992.

[26] Tom Loeys, Beatrijs Moerkerke, Olivia De Smet, and Ann Buysse. The analysis of zero-inflated count data: Beyond zero-inflated poisson regression. *British Journal of Mathematical and Statistical Psychology*, 65(1):163–180, 2012.

[27] Dan Ma. The $(a, b, 0)$ class. `https://actuarialmodelingtopics.wordpress.com/2018/11/12/the-ab0-class/`, November 2018.

[28] Dan Ma. The $(a, b, 1)$ class. `https://actuarialmodelingtopics.wordpress.com/2019/01/06/the-ab1-class/`, January 2019.

[29] Peter McCullagh and John A Nelder. Generalized linear models 2nd edition chapman and hall. *London, UK*, 1989.

[30] Younès Mouatassim and El Hadj Ezzahid. Poisson regression and zero-inflated poisson regression: application to private health insurance data. *European actuarial journal*, 2(2):187–204, 2012.

[31] John Mullahy. Specification and testing of some modified count data models. *Journal of econometrics*, 33(3):341–365, 1986.

[32] John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.

[33] Ingram Olkin, A John Petkau, and James V Zidek. A comparison of n estimators for the binomial distribution. *Journal of the American Statistical Association*, 76(375):637–642, 1981.

[34] Cyprian Ondieki Omari, Shalyne Gathoni Nyambura, and Joan Martha Wairimu Mwangi. Modeling the frequency and severity of auto insurance claims using statistical distributions. 2018.

[35] Harry H Panjer. Recursive evaluation of a family of compound distributions. *ASTIN Bulletin: The Journal of the IAA*, 12(1):22–26, 1981.

[36] KM Sakthivel and CS Rajitha. A comparative study of zero-inflated, hurdle models with artificial neural network in claim count modeling. *International Journal of Statistics and Systems*, 12(2):265–276, 2017.

[37] Bjørn Sundt and William S Jewell. Further results on recursive evaluation of compound distributions. *ASTIN Bulletin: The Journal of the IAA*, 12(1):27–39, 1981.

[38] Gordon Willmot. Sundt and jewell's family of discrete distributions. *ASTIN Bulletin: The Journal of the IAA*, 18(1):17–29, 1988.

[39] Kelvin KW Yau, Kui Wang, and Andy H Lee. Zero-inflated negative binomial mixed regression modeling of over-dispersed count data with extra zeros. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 45(4):437–452, 2003.

[40] Karen CH Yip and Kelvin KW Yau. On modeling claim frequency data in general insurance with extra zeros. *Insurance: Mathematics and Economics*, 36(2):153–163, 2005.

A Python Code for Fitting Dataset to a Poisson Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.special import factorial
from scipy.stats import poisson


def poisson_Dist(X):
   X = poisson_dataset(X)
   beta = np.mean(X)
   var = np.var(X)
   return beta


def poisson_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
   poisson_data = list(chain.from_iterable(uncollapsed_set))
   dataset = np.array(poisson_data)
   return dataset
```

```
def get_loglikelihood(data:list,lamda:float):
    N = len(data)
    result = -lamda * N + np.sum(data)*np.log(lamda) \
          - np.sum(np.log(factorial(data)))
    return result


def main(data_dict):
    N = np.sum([val for val in data_dict.values()])
    parameter = poisson_Dist(data_dict)
    likelihood = get_loglikelihood(poisson_dataset(data_dict),
       ↪ parameter)
    p_k = {k: N * poisson.pmf(k, parameter, loc=0) for k in data_
       ↪ dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','Poisson␣expected'])
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Poisson␣
       ↪ expected']].copy()
    return df_final, likelihood, parameter


if __name__ == '__main__':
```

```
data_dict = {0: 1923094,
       1: 33651,
       2: 5517,
       3: 1450,
       4: 692,
       5: 360,
       6: 186,
       7: 119,
       8: 71,
       9: 50,
       10: 45,
       11: 23,
       12: 19,
       13: 15,
       14: 11,
       15: 9,
       16: 3,
       17: 8,
       18: 4,
       19: 2,
       20: 6,
```

```
            21: 3,
            22: 2,
            23: 2,
            24: 2,
            33: 1,
            34: 1,
            35: 1,
            40: 1,
            41: 1,
            44: 1,
            45: 1,
            46: 1,
            90: 1,
            110: 1,
            164: 1
           }
df = main(data_dict)
```

B Python Code for Fitting Dataset to a Geometric Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.stats import geom


def geometric_Dist(X):
   X = geom_dataset(X)
   beta = np.mean(X)
   return beta


def geom_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
   geom_data = list(chain.from_iterable(uncollapsed_set))
   dataset = np.array(geom_data)
   return dataset


def get_loglikelihood(data:list,beta:float):
   N = len(data)
   result = np.sum(data) * np.log(beta) \
         - np.sum(data) * np.log(1+beta) \
         - N*np.log(1+beta)
```

```
    return result


def main(data_dict):
    N = np.sum([val for val in data_dict.values()])
    parameter = geometric_Dist(data_dict)
    likelihood = get_loglikelihood(geom_dataset(data_dict),
        ↪ parameter)
    p_k = {k: N * geom.pmf(k=k, p = 1/(1+parameter), loc=-1) for k
        ↪  in data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
        ↪ ','Geometric␣expected'])
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Geometric␣
        ↪ expected']].copy()
    return df_final, likelihood, parameter


if __name__ == '__main__':
        df = main(data_dict)
```

C Python Code for Fitting Dataset to a Negative Binomial Distribution

```
from itertools import chain
import numpy as np
from scipy.special import gammaln,factorial
from scipy.special import psi
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import nbinom
import pandas as pd


# X is a numpy array representing the data
# initial params is a numpy array representing the initial values
   ↪ of
# size and prob parameters
def fit_nbinom(X, initial_params=None):
   infinitesimal = np.finfo(np.float).eps


   def log_likelihood(params, *args):
      r, p = params
      X = args[0]
```

```
    N = X.size


    #MLE estimate based on the formula on Wikipedia:
    # http://en.wikipedia.org/wiki/Negative_binomial_
       ↪ distribution#Maximum_likelihood_estimation
    result = np.sum(gammaln(X + r)) \
       - np.sum(np.log(factorial(X))) \
       - N*(gammaln(r)) \
       + N*r*np.log(p) \
       + np.sum(X*np.log(1-(p if p < 1 else 1-infinitesimal)))


    return -result


def log_likelihood_deriv(params, *args):
    r, p = params
    X = args[0]
    N = X.size


    pderiv = (N*r)/p - np.sum(X)/(1-(p if p < 1 else 1-
       ↪ infinitesimal))
    rderiv = np.sum(psi(X + r)) \
```

```
        - N*psi(r) \
        + N*np.log(p)

    return np.array([-rderiv, -pderiv])

if initial_params is None:
    m = np.mean(X)
    v = np.var(X)
    size = (m**2)/(v-m) if v > m else 10

    #convert mu/size parameterization to prob/size
    p0 = size / ((size+m) if size+m != 0 else 1)
    r0 = size
    initial_params = np.array([r0, p0])

bounds = [(infinitesimal, None), (infinitesimal, 1)]
optimres = optim(log_likelihood,
                 x0=initial_params,
                 #fprime=log_likelihood_deriv,
                 args=(X,),
                 approx_grad=1,
```

```
                bounds=bounds)


    params = optimres[0]
    return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
        ↪ -1}


def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset



def main(claim_dict: dict):
    data = generate_nbinom_dataset(claim_dict)
    result = fit_nbinom(data)
    loglikelihood = get_loglikelihood(data,result['r'],
                              result['prob'])
    final_result = result.copy()
    final_result['loglikelihood'] = loglikelihood
    return final_result
```

```
def get_loglikelihood(data:list,r:float,p:float):
    infinitesimal = np.finfo(np.float).eps
    N = len(data)
    result = np.sum(gammaln(data + r)) \
        - np.sum(np.log(factorial(data))) \
        - N*(gammaln(r)) \
        + N*r*np.log(p) \
        + np.sum(data*np.log(1-(p if p < 1 else 1-infinitesimal)))
    return result


def get_df(data_dict):
    N = np.sum([val for val in data_dict.values()])
    result = main(data_dict)
    p_k = {k: N * nbinom.pmf(k=k,n=result['r'],p=result['prob'])
        ↪ for k in data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
        ↪ ','Negative␣Binomial␣expected'])
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Negative␣
        ↪ Binomial␣expected']].copy()
```

```
   return df_final, result


if __name__ == '__main__':

      df = get_df(data_dict)
```

D Python Code for Fitting Dataset to a Binomial Distribution

```
import numpy as np

from scipy.special import factorial

from scipy.stats import binom

import pandas as pd

#============================================================

# 1. Start with the largest observation

#============================================================

m_0_hat = max(data_dict)



#============================================================

# 2. obtain initial estimate of q as mean/m_0_hat

#============================================================
```

```
def get_q_from_m(data:dict,m):
    sigma_fx = np.sum([x*y for x,y in data.items()])
    sigma_f = np.sum([y for y in data.values()])
    q = (1/m)*(sigma_fx/sigma_f)
    return q


#============================================================
# 3. Calculate the log-likelihood at these values
#============================================================


def nCr(n,r):
    f = factorial
    result = f(n)/(f(r)*f(n-r))
    return result


def summand(k,n_k,m,q):
    result = n_k*(np.log(nCr(m,k))+k*np.log(q)
                +(m-k)*(np.log(1-q)))
    return result
```

```
def log_likelihood(X:dict,m:float,q:float):
   summand_array = []
   for x,y in X.items():
      res = summand(k=x, n_k=y, m=m, q=q)
      summand_array.append(res)
   result = np.sum(summand_array)
   return result
#============================================================
# 4. Increase m by 1 and repeat 2-4 until max is found
#============================================================

def find_binom_max_likelihood(data:dict):
   count,result_list = 0,[]
   m = max(data)
   q_0_hat = get_q_from_m(data,m)
   l = log_likelihood(data, m_0_hat, q_0_hat)
   result_list.append((m,q_0_hat,l))

   count +=1
   m +=1
   q_hat = get_q_from_m(data, m)
```

```
    l = log_likelihood(data, m, q_hat)
    result_list.append((m,q_hat,l))

    while result_list[count][-1] > result_list[count-1][-1]:
        count +=1
        m +=1
        q_hat = get_q_from_m(data, m)
        l = log_likelihood(data, m, q_hat)
        result_list.append((m,q_hat,l))
    return result_list[-2]
#============================================================


max_values = find_binom_max_likelihood(data_dict)
N = np.sum([y for y in data_dict.values()])
p_k = {k: N * binom.pmf(k=k,n=max_values[0],p=max_values[1]) for
    ↪ k in data_dict.keys()}
df = pd.DataFrame(list(p_k.items()),columns=['Number of Claims','
    ↪ Binomial expected'])
df.loc[:,'Observed'] = list(data_dict.values())
df_final = df[['Number of Claims','Observed','Binomial expected'
    ↪ ]].copy()
```

E Python Code for Fitting Dataset to a ZM Poisson Distribution

```
from itertools import chain

import numpy as np

import pandas as pd

from scipy.optimize import fmin_l_bfgs_b as optim

from scipy.stats import poisson


def getmean(X):

   X = poisson_dataset(X)

   lamda = np.mean(X)

   return lamda


def getlamda_hat(N, data_dict, n0):

   mean = getmean(data_dict)

   p_0_M = n0/N

   p0 = poisson.pmf(k=0, mu =mean)

   lamda_hat = (mean *(1-p0))/(1-p_0_M)

   return lamda_hat
```

```
def poisson_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    poiss_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(poiss_data)
    return dataset


def get_loglikelihood(X,n0, p_0_M, lamda_hat:float):
    N = np.sum([val for val in data_dict.values()])
    p_0 = poisson.pmf(k=0, mu =lamda_hat)
    l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
    ↪ > 0}
    n_k_p_k = {n_k: poisson.pmf
          (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
          ↪ dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0)) for n_k,p_k in n_k
    ↪ _p_k.items()]
    l1 = np.sum(summand)
    result = l0+l1


    return result
```

```
def fit_zmnbinom(X, initial_params=None):
   N = np.sum([val for val in data_dict.values()])
   infinitesimal = np.finfo(np.float).eps
   n0 = 370412 #data_dict[0]
   p_0_M = n0 / 421240
   sliced_data_dict = {key: val for key, val in data_dict.items()
      ↪  if key > 0}
   def loglikelihood(params, *args):
      lamda_hat = params
      X = args[0]
      N = X.size
      p_0 = poisson.pmf(k=0, mu =lamda_hat)
      l0 = n0 * np.log(p_0_M) + (N)*np.log(1-p_0_M)
      n_k_p_k = {n_k: poisson.pmf
            (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
               ↪ dict.items()}
      summand = [n_k *(np.log(p_k)-np.log(1-p_0)) for n_k,p_k in
         ↪ n_k_p_k.items()]
      l1 = np.sum(summand)
      result = l0+l1
```

```
        return -result

    if initial_params is None:
        lamda0 = getlamda_hat(N, data_dict, n0)
        initial_params = np.array([lamda0])

    bounds = [(infinitesimal, None)]
    optimres = optim(loglikelihood,
                 x0=initial_params,
                 args=(X,),
                 approx_grad=1,
                 bounds=bounds)

    params = optimres[0]
    return params[0]

def main(data_dict):
    X = poisson_dataset(data_dict)
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
```

```
    p_0_M = n0 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 0}
    N_modified = np.sum([val for val in sliced_data_dict.values()
    ↪ ])
    lamda_hat = fit_zmnbinom(X)
    likelihood = get_loglikelihood(data_dict,n0, p_0_M, lamda_hat)
    p_0 = poisson.pmf(k=0, mu =lamda_hat)
    p_k = {k: N*((1-p_0_M)/(1-p_0)) * poisson.pmf(k=k, mu =lamda_
    ↪ hat, loc=0) for k in sliced_data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
    ↪ ','ZM␣Poisson␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df = pd.concat([df_0,df],sort=False)
    df.loc[:,'Observed'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','Observed','ZM␣Poisson␣
    ↪ expected']].copy()
    return df_final, likelihood, lamda_hat


if __name__ == '__main__':
      df = main(data_dict)
```

F Python Code for Fitting Dataset to a ZM Geometric Distribution

```
from itertools import chain

import numpy as np

import pandas as pd



from scipy.stats import geom

def getmean(X):

   X = geom_dataset(X)

   beta = np.mean(X)

   #var = np.var(X)

   return beta



def getbeta_hat(N, data_dict, n0):

   mean = getmean(data_dict)

   beta_hat = (N*mean)/(N-n0) - 1

   return beta_hat



def geom_dataset(X:dict):

   uncollapsed_set = [[x]*y for x,y in X.items()]
```

```
    geom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(geom_data)
    return dataset


def get_loglikelihood(X,n0, p_0_M, beta_hat:float):
    N = np.sum([val for val in data_dict.values()])
    p_0 = 1/(1+beta_hat)
    l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 0}
    n_k_p_k = {n_k: geom.pmf
          (k=k, p = 1/(1+beta_hat), loc = -1) for k,n_k in sliced_
             ↪ data_dict.items()}
    p_0 = 1/(1+beta_hat)
    summand = [n_k *(np.log(p_k)-np.log(1-p_0)) for n_k,p_k in n_k
       ↪ _p_k.items()]
    l1 = np.sum(summand)
    result = l0+l1
    return result


def main(data_dict):
```

```
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    p_0_M = n0 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 0}
    beta_hat = getbeta_hat(N, data_dict, n0)
    likelihood = get_loglikelihood(data_dict,n0, p_0_M, beta_hat)
    p_0 = geom.pmf(k=0, p = 1/(1+beta_hat))
    p_k = {k: N*((1-p_0_M)/(1-p_0))* geom.pmf(k=k, p = 1/(1+beta_
       ↪ hat),loc=0) for k in sliced_data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','ZM␣Geometric␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df = pd.concat([df_0,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','ZM␣Geometric␣
       ↪ expected']].copy()
    return df_final, likelihood, beta_hat, p_0_M


if __name__ == '__main__':
       df = main(data_dict)
```

G Python Code for Fitting Dataset to a ZM Negative Binomial Distribution

```
from itertools import chain
import numpy as np
from scipy.special import gammaln,factorial
from scipy.special import psi
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import nbinom
import pandas as pd


def fit_zmnbinom(X, initial_params=None):
   infinitesimal = np.finfo(np.float).eps
   n0 = data_dict[0]
   def loglikelihood2(params, *args):
      r, p = params
      X = args[0]
      N = X.size
      p_0_M = n0 / N
      p_0 = p**r
      l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
```

```
sliced_data_dict = {key: val for key, val in data_dict.
   ↪ items() if key > 0}
X_sliced = generate_nbinom_dataset(sliced_data_dict)
N_sliced = X_sliced.size
l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
l1 = np.sum(gammaln(X_sliced + r)) \
   - np.sum(np.log(factorial(X_sliced))) \
   - N_sliced*(gammaln(r)) \
   + N_sliced*r*np.log(p) \
   + np.sum(X_sliced*np.log(1-(p if p < 1 else 1-
      ↪ infinitesimal))) \
   - N_sliced*np.log(1-p**r)
result = l0+l1
return -result
if initial_params is None:
N = X.size
m = np.mean(X)
v = np.var(X)
p0M = n0 / N
b = (v-m)/m
p0 = 1/(1+b)
```

```
        r0 = (m*(1-p0))/(b*(1-p0M))
        initial_params = np.array([r0, p0])


    bounds = [(infinitesimal, None), (infinitesimal, 1)]
    optimres = optim(loglikelihood2,
                x0=initial_params,
                args=(X,),
                approx_grad=1,
                bounds=bounds)


    params = optimres[0]
    return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
        ↪ -1}


def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset


def main(claim_dict: dict):
```

```
    N = np.sum([val for val in claim_dict.values()])
    n0 = claim_dict[0]
    p_0_M = n0 / N
    data = generate_nbinom_dataset(claim_dict)
    result = fit_zmnbinom(data)
    loglikelihood = get_loglikelihood(data_dict,data,result['r'],
                                result['prob'], n0, p_0_M)
    final_result = result.copy()
    final_result['loglikelihood'] = loglikelihood
    return final_result


def get_loglikelihood(data_dict:dict, data:list,r:float,p:float,
    ↪ n0, p_0_M):
    N = data.size
    p_0 = p**r
    l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 0}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p, loc=0) for k,n_k in
       ↪ sliced_data_dict.items()}
```

```
    summand = [n_k *(np.log(p_k)-np.log(1-p_0)) for n_k,p_k in n_k
       ↪ _p_k.items()]
    l1 = np.sum(summand)
    result = l0+l1
    return result


def get_df(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    p_0_M = n0 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 0}
    result = main(data_dict)
    p_0 = (result['prob'])**result['r']
    p_k = {k: N*((1-p_0_M)/(1-p_0)) * nbinom.pmf(k=k,n=result['r'
       ↪ ],p=result['prob'],loc=0) for k in sliced_data_dict.keys
       ↪ ()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','ZM␣Negative␣Binomial␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df = pd.concat([df_0,df],sort=False)
```

```
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number of Claims','frequency','ZM Negative
        ↪ Binomial expected']].copy()
    return df_final, result, p_0_M


if __name__ == '__main__':
        df = main(data_dict)
```

H Python Code for Fitting Dataset to a ZM Binomial Distribution

```
import numpy as np
from scipy.special import factorial
from scipy.stats import binom
import pandas as pd


N = np.sum([val for val in data_dict.values()])
n0 = data_dict[0]
p_0_M = n0 / N
```

```
# Start with the largest observation
m_0_hat = max(data_dict)


def get_q_from_m(data:dict,m):
    N = np.sum([val for val in data.values()])
    n0 = data_dict[0]
    p_0_M = n0 / N
    sigma_fx = np.sum([x*y for x,y in data.items()])
    sigma_f = np.sum([y for y in data.values()])
    mean = (sigma_fx/sigma_f)
    q = (1/m)*mean
    p0 = binom.pmf(k=0,n=m,p=q)
    q_hat = (mean * (1-p0))/(m*(1-p_0_M))
    return q_hat


def nCr(n,r):
    f = factorial
    result = f(n)/(f(r)*f(n-r))
    return result


def summand(k,n_k,m,q):
```

```
    p_0 = binom.pmf(k=0,n=m,p=q)
    result = n_k*(np.log(nCr(m,k))+k*np.log(q)
                +(m-k)*(np.log(1-q)) - np.log(1-p_0))
    return result


def log_likelihood(X:dict,m:float,q:float): #X is sliced data
    summand_array = []
    l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)
    for x,y in X.items():
        res = summand(k=x, n_k=y, m=m, q=q)
        summand_array.append(res)
    l1 = np.sum(summand_array)
    result = l0 + l1
    return result


def find_zmbinom_max_likelihood(data:dict):
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪   if key > 0}
    count,result_list = 0,[]
    m = max(data)
    q_0_hat = get_q_from_m(data,m)
```

```
    l = log_likelihood(sliced_data_dict, m_0_hat, q_0_hat)
    result_list.append((m,q_0_hat,l))

    count +=1
    m +=1
    q_hat = get_q_from_m(data, m)
    l = log_likelihood(sliced_data_dict, m, q_hat)
    result_list.append((m,q_hat,l))

    while result_list[count][-1] > result_list[count-1][-1]:
        count +=1
        m +=1
        q_hat = get_q_from_m(data, m)
        l = log_likelihood(sliced_data_dict, m, q_hat)
        result_list.append((m,q_hat,l))
    return result_list[-2]

max_values = find_zmbinom_max_likelihood(data_dict)
sliced_data_dict = {key: val for key, val in data_dict.items() if
    ↪ key > 0}
N = np.sum([y for y in data_dict.values()])
```

```
p_0 = binom.pmf(k=0,n=max_values[0],p=max_values[1])
p_k = {k: N*((1-p_0_M)/(1-p_0)) * binom.pmf(k=k,n=max_values[0],p
   ↪ =max_values[1]) for k in sliced_data_dict.keys()}
df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims','
   ↪ ZM␣Binomial␣expected'])
df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
df = pd.concat([df_0,df],sort=False)
df.loc[:,'Observed'] = list(data_dict.values())
df_final = df[['Number␣of␣Claims','Observed','ZM␣Binomial␣
   ↪ expected']].copy()
```

I Python Code for Fitting Dataset to a Z1M Poisson Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import poisson
```

```
def getmean(X):
    X = poisson_dataset(X)
    lamda = np.mean(X)
    return lamda


def getlamda_hat(N, data_dict, n0, n1):
    mean = getmean(data_dict)
    p_0_M = n0/N
    p_1_M = n1/N
    p0 = poisson.pmf(k=0, mu =mean)
    p1 = poisson.pmf(k=1, mu =mean)
    lamda_hat = (mean *(1-p0-p1))/(1-p_0_M-p_1_M)
    return lamda_hat


def poisson_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    poiss_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(poiss_data)
    return dataset


def get_loglikelihood(X,n0,n1, p_0_M,p_1_M, lamda_hat:float):
```

```
    N = np.sum([val for val in data_dict.values()])
    p_0 = poisson.pmf(k=0, mu =lamda_hat)
    p_1 = poisson.pmf(k=1, mu =lamda_hat)
    l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np.
       ↪ log(1-p_0_M-p_1_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 1}
    n_k_p_k = {n_k: poisson.pmf
          (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
             ↪ dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1)) for n_k,p_k in
       ↪  n_k_p_k.items()]
    l2 = np.sum(summand)
    result = l01+l2


    return result


def fit_zmnbinom(X, initial_params=None):
    N = np.sum([val for val in data_dict.values()])
    infinitesimal = np.finfo(np.float).eps
    n0 = 1923094 #data_dict[0]
```

```
n1 = 33651

p_0_M = n0 / 1965355

p_1_M = n1 / 1965355

sliced_data_dict = {key: val for key, val in data_dict.items()
   ↪  if key > 1}

def loglikelihood(params, *args):

   lamda_hat = params

   X = args[0]

   N = X.size

   p_0 = poisson.pmf(k=0, mu =lamda_hat)

   p_1 = poisson.pmf(k=1, mu =lamda_hat)

   l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np
      ↪ .log(1-p_0_M-p_1_M)

   n_k_p_k = {n_k: poisson.pmf
         (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
            ↪ dict.items()}

   summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1)) for n_k,p_k
      ↪  in n_k_p_k.items()]

   l2 = np.sum(summand)

   result = l01+l2
```

```
        return -result

    if initial_params is None:
        lamda0 = getlamda_hat(N, data_dict, n0, n1)
        initial_params = np.array([lamda0])

    bounds = [(infinitesimal, None)]
    optimres = optim(loglikelihood,
                 x0=initial_params,
                 args=(X,),
                 approx_grad=1,
                 bounds=bounds)

    params = optimres[0]
    return params[0]

def main(data_dict):
    X = poisson_dataset(data_dict)
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
```

```
p_0_M = n0 / N

p_1_M = n1 / N

sliced_data_dict = {key: val for key, val in data_dict.items()
   ↪  if key > 1}

lamda_hat = fit_zmnbinom(X)

likelihood = get_loglikelihood(data_dict,n0,n1, p_0_M,p_1_M,
   ↪ lamda_hat)

p_0 = poisson.pmf(k=0, mu =lamda_hat)

p_1 = poisson.pmf(k=1, mu =lamda_hat)

p_k = {k: N*((1-p_0_M-p_1_M)/(1-p_0-p_1)) * poisson.pmf(k=k,
   ↪ mu =lamda_hat, loc=0) for k in sliced_data_dict.keys()}

df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
   ↪ ','Z1M␣Poisson␣expected'])

df_0 = pd.DataFrame([[0, n0]], columns=df.columns)

df_1 = pd.DataFrame([[1, n1]], columns=df.columns)

df = pd.concat([df_0,df_1,df],sort=False)

df.loc[:,'Observed'] = list(data_dict.values())

df_final = df[['Number␣of␣Claims','Observed','Z1M␣Poisson␣
   ↪ expected']].copy()

return df_final, likelihood, lamda_hat,p_0_M, p_1_M
```

```
if __name__ == '__main__':

      df = main(data_dict)
```

J Python Code for Fitting Dataset to a Z1M Geometric Distribution

```
from itertools import chain

import numpy as np

import pandas as pd

from scipy.stats import geom

def getmean(X):

   X = geom_dataset(X)

   beta = np.mean(X)

   #var = np.var(X)

   return beta


def getbeta_hat(N, data_dict, n0, n1):

   mean = getmean(data_dict)

   beta_hat = ((N*mean -n1)/(N-n0-n1)) - 2

   return beta_hat
```

```
def geom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    geom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(geom_data)
    return dataset


def get_loglikelihood(X,n0,n1,p_0_M,p_1_M, beta_hat:float):
    N = np.sum([val for val in X.values()])
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/(1+beta_hat)**2
    l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np.
       ↪ log(1-p_0_M-p_1_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 1}
    n_k_p_k = {n_k: (beta_hat**k)/((1+beta_hat)**(k+1)) for k,n_k
       ↪ in sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1)) for n_k,p_k in
       ↪  n_k_p_k.items()]
    l2 = np.sum(summand)
    result = l01+l2
```

```
    return result


def main(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    p_0_M = n0 / N
    p_1_M = n1 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
        ↪  if key > 1}
    beta_hat = getbeta_hat(N, data_dict, n0, n1)
    likelihood = get_loglikelihood(data_dict,n0,n1,p_0_M,p_1_M,
        ↪ beta_hat)
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/((1+beta_hat)**2)
    p_k = {k: N*((1-p_0_M-p_1_M)/(1-p_0-p_1))*(beta_hat**k)/(1+
        ↪ beta_hat)**(k+1) for k in sliced_data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
        ↪ ','ZM_2␣Geometric␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
```

```
    df = pd.concat([df_0,df_1,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number of Claims','frequency','ZM_2 Geometric
        ↪ expected']].copy()
    return df_final, likelihood, beta_hat, p_0_M, p_1_M


if __name__ == '__main__':
        df = main(data_dict)
```

K Python Code for Fitting Dataset to a Z1M Negative Binomial Distribution

```
from itertools import chain
import numpy as np
from scipy.special import gammaln,factorial
from scipy.special import psi
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import nbinom
import pandas as pd
```

```
def fit_zmnbinom(X, initial_params=None):
   infinitesimal = np.finfo(np.float).eps
   n0 = data_dict[0]
   n1 = data_dict[1]

   def log_likelihood(params, *args):
      r, p = params
      X = args[0]
      N = X.size
      p_0_M = n0 / N
      p_1_M = n1 / N
      p_0 = p**r
      p_1 = nbinom.pmf(k=1,n=r,p=p)
      l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np
         ↪ .log(1-p_0_M-p_1_M)
      sliced_data_dict = {key: val for key, val in data_dict.
         ↪ items() if key > 1}
      n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p,loc=0) for k,n_k in
         ↪ sliced_data_dict.items()}
```

```
    summand = [n_k *(np.log(p_k if p_k < 1 else 1-infinitesimal
      ↪ )-np.log(1-p_0-p_1)) for n_k,p_k in n_k_p_k.items()]
    l2 = np.sum(summand)
    result = l01+l2
    return -result
    def log_likelihood_deriv(params, *args):
    r, p = params
    X = args[0]
    N = X.size

    pderiv = (N*r)/p - np.sum(X)/(1-(p if p < 1 else 1-
      ↪ infinitesimal))
    rderiv = np.sum(psi(X + r)) \
       - N*psi(r) \
       + N*np.log(p)

    return np.array([-rderiv, -pderiv])

if initial_params is None:
    N = X.size
    p0M = n0 / N
```

```
    p1M = n1 / N
    m = np.mean(X)
    v = np.var(X)

    b = (v-m)/m
    #p0 = 1/(1+b)
    p0 = nbinom.pmf(k=0,n=1,p=1/(1+b))
    p1 = nbinom.pmf(k=1,n=1,p=1/(1+b))
    r0 = (m*(1-p0-p1))/(b*(1-p0M-p1M))
    initial_params = np.array([r0, p0])

bounds = [(0, None), (infinitesimal, 1)]
optimres = optim(log_likelihood,
             x0=initial_params,
             args=(X,),
             approx_grad=1,
             bounds=bounds)

params = optimres[0]
return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
   ↪ -1}
```

```
def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset


def main(claim_dict: dict):
    N = np.sum([val for val in claim_dict.values()])
    n0 = claim_dict[0]
    n1 = claim_dict[1]
    p_0_M = n0 / N
    p_1_M = n1 / N
    data = generate_nbinom_dataset(claim_dict)
    result = fit_zmnbinom(data)
    loglikelihood = get_loglikelihood(data_dict,data,result['r'],
                                result['prob'], n0, n1, p_0_M,p_1_M)
    final_result = result.copy()
    final_result['loglikelihood'] = loglikelihood
    return final_result
```

```
def get_loglikelihood(data_dict:dict, data:list,r:float,p:float,
    ↪ n0,n1, p_0_M, p_1_M):
    N = data.size
    p_0 = p**r
    p_1 = nbinom.pmf(k=1,n=r,p=p)
    l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np.
       ↪ log(1-p_0_M-p_1_M)
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 1}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p, loc=0) for k,n_k in
       ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1)) for n_k,p_k in
       ↪  n_k_p_k.items()]
    l2 = np.sum(summand)
    result = l01+l2
    return result


def get_df(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
```

```
    p_0_M = n0 / N
    p_1_M = n1 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 1}
    result = main(data_dict)
    p_0 = (result['prob'])**result['r']
    p_1 = nbinom.pmf(k=1,n=result['r'],p=result['prob'])
    p_k = {k: N*((1-p_0_M-p_1_M)/(1-p_0-p_1)) * nbinom.pmf(k=k,n=
    ↪ result['r'],p=result['prob'],loc=0) for k in sliced_data
    ↪ _dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
    ↪ ','Z1M␣Negative␣Binomial␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df = pd.concat([df_0,df_1,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Z1M␣Negative␣
    ↪ Binomial␣expected']].copy()
    return df_final, result, p_0_M, p_1_M


if __name__ == '__main__':
```

```
    df = get_df(data_dict)
```

L Python Code for Fitting Dataset to a Z1M Binomial Distribution

```
import numpy as np
from scipy.special import factorial
from scipy.stats import binom
import pandas as pd
N = np.sum([val for val in data_dict.values()])
n0 = data_dict[0]
n1 = data_dict[1]
p_0_M = n0 / N
p_1_M = n1 / N


m_0_hat = max(data_dict)


def get_q_from_m(data:dict,m):
   N = np.sum([val for val in data.values()])
   n0 = data_dict[0]
```

```
    n1 = data_dict[1]
    p_0_M = n0 / N
    p_1_M = n1 / N
    sigma_fx = np.sum([x*y for x,y in data.items()])
    sigma_f = np.sum([y for y in data.values()])
    mean = (sigma_fx/sigma_f)
    q = (1/m)*mean
    p0 = binom.pmf(k=0,n=m,p=q)
    p1 = binom.pmf(k=1,n=m,p=q)
    q_hat = (mean * (1-p0-p1))/(m*(1-p_0_M-p_1_M))
    return q_hat


def nCr(n,r):
    f = factorial
    result = f(n)/(f(r)*f(n-r))
    return result


def summand(k,n_k,m,q):
    p_0 = binom.pmf(k=0,n=m,p=q)
    p_1 = binom.pmf(k=1,n=m,p=q)
    result = n_k*(np.log(nCr(m,k))+k*np.log(q)
```

```
            +(m-k)*(np.log(1-q)) - np.log(1-p_0-p_1))
    return result


def log_likelihood(X:dict,m:float,q:float): #X is sliced data
    summand_array = []
    l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np.
        ↪ log(1-p_0_M-p_1_M)
    for x,y in X.items():
        res = summand(k=x, n_k=y, m=m, q=q)
        summand_array.append(res)
    l2 = np.sum(summand_array)
    result = l01 + l2
    return result


def find_zmbinom_max_likelihood(data:dict):
    sliced_data_dict = {key: val for key, val in data_dict.items()
        ↪  if key > 1}
    count,result_list = 0,[]
    m = max(data)
    q_0_hat = get_q_from_m(data,m)
    l = log_likelihood(sliced_data_dict, m_0_hat, q_0_hat)
```

```
    result_list.append((m,q_0_hat,l))


    count +=1
    m +=1
    q_hat = get_q_from_m(data, m)
    l = log_likelihood(sliced_data_dict, m, q_hat)
    result_list.append((m,q_hat,l))


    while result_list[count][-1] > result_list[count-1][-1]:
        count +=1
        m +=1
        q_hat = get_q_from_m(data, m)
        l = log_likelihood(sliced_data_dict, m, q_hat)
        result_list.append((m,q_hat,l))
    return result_list[-2]


max_values = find_zmbinom_max_likelihood(data_dict)
sliced_data_dict = {key: val for key, val in data_dict.items() if
    ↪  key > 1}
N = np.sum([y for y in data_dict.values()])
p_0 = binom.pmf(k=0,n=max_values[0],p=max_values[1])
```

```
p_1 = binom.pmf(k=1,n=max_values[0],p=max_values[1])
p_k = {k: N*((1-p_0_M-p_1_M)/(1-p_0-p_1)) * binom.pmf(k=k,n=max_
   ↪ values[0],p=max_values[1]) for k in sliced_data_dict.keys()
   ↪ }
df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims','
   ↪ Z1M␣Binomial␣expected'])
df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
df = pd.concat([df_0,df_1,df],sort=False)
df.loc[:,'Observed'] = list(data_dict.values())
df_final = df[['Number␣of␣Claims','Observed','Z1M␣Binomial␣
   ↪ expected']].copy()
```

M Python Code for Fitting Dataset to a Z12M Poisson Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.optimize import fmin_l_bfgs_b as optim
```

```
from scipy.stats import poisson


def getmean(X):
   X = poisson_dataset(X)
   lamda = np.mean(X)
   return lamda


def getlamda_hat(N, data_dict, n0, n1, n2):
   mean = getmean(data_dict)
   p_0_M = n0/N
   p_1_M = n1/N
   p_2_M = n2/N
   p0 = poisson.pmf(k=0, mu =mean)
   p1 = poisson.pmf(k=1, mu =mean)
   p2 = poisson.pmf(k=2, mu =mean)
   lamda_hat = (mean *(1-p0-p1-p2))/(1-p_0_M-p_1_M-p_2_M)
   return lamda_hat


def poisson_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
   poiss_data = list(chain.from_iterable(uncollapsed_set))
```

```
    dataset = np.array(poiss_data)
    return dataset


def get_loglikelihood(X,n0,n1,n2, p_0_M,p_1_M,p_2_M, lamda_hat:
    ↪ float):
    N = np.sum([val for val in data_dict.values()])
    p_0 = poisson.pmf(k=0, mu =lamda_hat)
    p_1 = poisson.pmf(k=1, mu =lamda_hat)
    p_2 = poisson.pmf(k=2, mu =lamda_hat)
    l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(p
       ↪ _2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 2}
    n_k_p_k = {n_k: poisson.pmf
          (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
             ↪ dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2)) for n_k,p_
       ↪ k in n_k_p_k.items()]
    l3 = np.sum(summand)
    result = l012+l3
    return result
```

```
def fit_zmnbinom(X, initial_params=None):
   N = np.sum([val for val in data_dict.values()])
   infinitesimal = np.finfo(np.float).eps
   n0 = 1923094 #data_dict[0]
   n1 = 33651
   n2 = 5517
   p_0_M = n0 / 1965355
   p_1_M = n1 / 1965355
   p_2_M = n2 / 1965355
   sliced_data_dict = {key: val for key, val in data_dict.items()
      ↪  if key > 1}
   def loglikelihood(params, *args):
      lamda_hat = params
      X = args[0]
      N = X.size
      p_0 = poisson.pmf(k=0, mu =lamda_hat)
      p_1 = poisson.pmf(k=1, mu =lamda_hat)
      p_2 = poisson.pmf(k=2, mu =lamda_hat)
      l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.
         ↪ log(p_2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
```

```
    #sliced_data_dict = {key: val for key, val in X.items() if
       ↪ key > 2}
    n_k_p_k = {n_k: poisson.pmf
          (k=k, mu =lamda_hat, loc=0) for k,n_k in sliced_data_
             ↪ dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2)) for n_k
       ↪ ,p_k in n_k_p_k.items()]
    l3 = np.sum(summand)
    result = l012+l3
    return -result


if initial_params is None:
    lamda0 = getlamda_hat(N, data_dict, n0, n1, n2)
    initial_params = np.array([lamda0])


bounds = [(infinitesimal, None)]
optimres = optim(loglikelihood,
             x0=initial_params,
             args=(X,),
             approx_grad=1,
             bounds=bounds)
```

```
    params = optimres[0]
    return params[0]


def main(data_dict):
    X = poisson_dataset(data_dict)
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 2}
    N_modified = np.sum([val for val in sliced_data_dict.values()
    ↪ ])
    lamda_hat = fit_zmnbinom(X)
    likelihood = get_loglikelihood(data_dict,n0,n1,n2, p_0_M,p_1_M
    ↪ ,p_2_M,lamda_hat)
    p_0 = poisson.pmf(k=0, mu =lamda_hat)
```

```
    p_1 = poisson.pmf(k=1, mu =lamda_hat)
    p_2 = poisson.pmf(k=2, mu =lamda_hat)
    p_k = {k: N*((1-p_0_M-p_1_M-p_2_M)/(1-p_0-p_1-p_2)) * poisson.
       ↪ pmf(k=k, mu =lamda_hat) for k in sliced_data_dict.keys()
       ↪ }
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','Z12M␣Poisson␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
    df = pd.concat([df_0,df_1,df_2,df],sort=False)
    df.loc[:,'Observed'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','Observed','Z12M␣Poisson␣
       ↪ expected']].copy()
    return df_final, likelihood, lamda_hat,p_0_M, p_1_M, p_2_M


if __name__ == '__main__':
       df = main(data_dict)
```

N Python Code for Fitting Dataset to a Z12M Geometric Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
#from scipy.special import gammaln,factorial
from scipy.stats import geom
def getmean(X):
   X = geom_dataset(X)
   beta = np.mean(X)
   #var = np.var(X)
   return beta


def getbeta_hat(N, data_dict, n0, n1, n2):
   mean = getmean(data_dict)
   beta_hat = ((N*mean -n1 - n2)/(N-n0-n1-n2)) - 3
   return beta_hat


def geom_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
```

```
    geom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(geom_data)
    return dataset


def get_loglikelihood(X,n0,n1,n2,p_0_M,p_1_M,p_2_M, beta_hat:
    ↪ float):
    N = np.sum([val for val in data_dict.values()])
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/(1+beta_hat)**2
    p_2 = (beta_hat**2)/((1+beta_hat)**3)
    l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(p
        ↪ _2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
        ↪ > 2}
    n_k_p_k = {n_k: (beta_hat**k)/((1+beta_hat)**(k+1)) for k,n_k
        ↪ in sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2)) for n_k,p_
        ↪ k in n_k_p_k.items()]
    l3 = np.sum(summand)
    result = l012+l3
    return result
```

```
def main(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 2}
    beta_hat = getbeta_hat(N, data_dict, n0, n1, n2)
    likelihood = get_loglikelihood(data_dict,n0,n1,n2,p_0_M,p_1_M,
    ↪ p_2_M,beta_hat)
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/((1+beta_hat)**2)
    p_2 = (beta_hat**2)/((1+beta_hat)**3)
    p_k = {k: N*((1-p_0_M-p_1_M-p_2_M)/(1-p_0-p_1-p_2))*((beta_hat
    ↪ **k)/(1+beta_hat)**(k+1)) for k in sliced_data_dict.keys
    ↪ ()}
```

```
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
        ↪ ','ZM_3␣Geometric␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
    df = pd.concat([df_0,df_1,df_2,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','ZM_3␣Geometric␣
        ↪ expected']].copy()
    return df_final, likelihood, beta_hat, p_0_M, p_1_M, p_2_M


if __name__ == '__main__':
        df = main(data_dict)
```

O Python Code for Fitting Dataset to a Z12M Negative Binomial Distribution

```
from itertools import chain
import numpy as np
from scipy.special import gammaln,factorial
```

```
from scipy.special import psi
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import nbinom
import pandas as pd



def fit_zmnbinom(X, initial_params=None):
   infinitesimal = np.finfo(np.float).eps
   n0 = data_dict[0]
   n1 = data_dict[1]
   n2 = data_dict[2]


   def log_likelihood(params, *args):
      r, p = params
      X = args[0]
      N = X.size
      p_0_M = n0 / N
      p_1_M = n1 / N
      p_2_M = n2 / N
      p_0 = p**r
      p_1 = nbinom.pmf(k=1,n=r,p=p)
```

```
    p_2 = nbinom.pmf(k=2,n=r,p=p)
    l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.
      ↪ log(p_2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
    sliced_data_dict = {key: val for key, val in data_dict.
      ↪ items() if key > 2}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p,loc=0) for k,n_k in
      ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k if p_k < 1 else 1-infinitesimal
      ↪ )-np.log(1-p_0-p_1-p_2)) for n_k,p_k in n_k_p_k.items
      ↪ ()]
    l3 = np.sum(summand)
    result = l012+l3
    return -result


def log_likelihood_deriv(params, *args):
    r, p = params
    X = args[0]
    N = X.size


    pderiv = (N*r)/p - np.sum(X)/(1-(p if p < 1 else 1-
      ↪ infinitesimal))
```

```
    rderiv = np.sum(psi(X + r)) \
       - N*psi(r) \
       + N*np.log(p)

    return np.array([-rderiv, -pderiv])

if initial_params is None:
    N = X.size
    p0M = n0 / N
    p1M = n1 / N
    p2M = n2 / N
    m = np.mean(X)
    v = np.var(X)
    b = (v-m)/m
    p0 = nbinom.pmf(k=0,n=1,p=1/(1+b))
    p1 = nbinom.pmf(k=1,n=1,p=1/(1+b))
    p2 = nbinom.pmf(k=2,n=1,p=1/(1+b))
    r0 = (m*(1-p0-p1-p2))/(b*(1-p0M-p1M-p2M))
    initial_params = np.array([r0, p0])

bounds = [(0, None), (infinitesimal, 1)]
```

```
    optimres = optim(log_likelihood,
                 x0=initial_params,
                 args=(X,),
                 approx_grad=1,
                 bounds=bounds)


    params = optimres[0]
    return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
       ↪ -1}


def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset


def main(claim_dict: dict):
    N = np.sum([val for val in claim_dict.values()])
    n0 = claim_dict[0]
    n1 = claim_dict[1]
    n2 = claim_dict[2]
```

```
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    data = generate_nbinom_dataset(claim_dict)
    result = fit_zmnbinom(data)
    loglikelihood = get_loglikelihood(data_dict,data,result['r'],
                                result['prob'], n0, n1,n2, p_0_M,p_1
                                    ↪ _M,p_2_M)
    final_result = result.copy()
    final_result['loglikelihood'] = loglikelihood
    return final_result


def get_loglikelihood(data_dict:dict, data:list,r:float,p:float,
    ↪ n0,n1,n2, p_0_M, p_1_M, p_2_M):
    N = data.size
    p_0 = p**r
    p_1 = nbinom.pmf(k=1,n=r,p=p)
    p_2 = nbinom.pmf(k=2,n=r,p=p)
    l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(p
       ↪ _2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
```

```
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 2}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p, loc=0) for k,n_k in
    ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2)) for n_k,p_
    ↪ k in n_k_p_k.items()]
    l3 = np.sum(summand)
    result = l012+l3
    return result


def get_df(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 2}
    result = main(data_dict)
```

```
    p_0 = (result['prob'])**result['r']
    p_1 = nbinom.pmf(k=1,n=result['r'],p=result['prob'])
    p_2 = nbinom.pmf(k=2,n=result['r'],p=result['prob'])
    p_k = {k: N*((1-p_0_M-p_1_M-p_2_M)/(1-p_0-p_1-p_2)) * nbinom.
       ↪ pmf(k=k,n=result['r'],p=result['prob'],loc=0) for k in
       ↪ sliced_data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','Z12M␣Negative␣Binomial␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
    df = pd.concat([df_0,df_1,df_2,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Z12M␣Negative␣
       ↪ Binomial␣expected']].copy()
    return df_final, result, p_0_M, p_1_M,p_2_M


if __name__ == '__main__':
        df = get_df(data_dict)
```

P Python Code for Fitting Dataset to a Z12M Binomial Distribution

```
import numpy as np

from scipy.special import factorial

from scipy.stats import binom

import pandas as pd

N = np.sum([val for val in data_dict.values()])

n0 = data_dict[0]

n1 = data_dict[1]

n2 = data_dict[2]

p_0_M = n0 / N

p_1_M = n1 / N

p_2_M = n2 / N



m_0_hat = max(data_dict)



def get_q_from_m(data:dict,m):

   N = np.sum([val for val in data.values()])

   n0 = data_dict[0]

   n1 = data_dict[1]
```

```
    n2 = data_dict[2]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    sigma_fx = np.sum([x*y for x,y in data.items()])
    sigma_f = np.sum([y for y in data.values()])
    mean = (sigma_fx/sigma_f)
    q = (1/m)*mean
    p0 = binom.pmf(k=0,n=m,p=q)
    p1 = binom.pmf(k=1,n=m,p=q)
    p2 = binom.pmf(k=2,n=m,p=q)
    q_hat = (mean * (1-p0-p1-p2))/(m*(1-p_0_M-p_1_M-p_2_M))
    return q_hat


def nCr(n,r):
    f = factorial
    result = f(n)/(f(r)*f(n-r))
    return result


def summand(k,n_k,m,q):
    p_0 = binom.pmf(k=0,n=m,p=q)
```

```
    p_1 = binom.pmf(k=1,n=m,p=q)
    p_2 = binom.pmf(k=2,n=m,p=q)
    result = n_k*(np.log(nCr(m,k))+k*np.log(q)
                +(m-k)*(np.log(1-q)) - np.log(1-p_0-p_1-p_2))
    return result


def log_likelihood(X:dict,m:float,q:float): #X is sliced data
    summand_array = []
    #l01 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) +(N-n0-n1)*np.
        ↪ log(1-p_0_M-p_1_M)
    l012 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(p
        ↪ _2_M)+(N-n0-n1-n2)*np.log(1-p_0_M-p_1_M-p_2_M)
    for x,y in X.items():
        res = summand(k=x, n_k=y, m=m, q=q)
        summand_array.append(res)
    l3 = np.sum(summand_array)
    result = l012 + l3
    return result


def find_zmbinom_max_likelihood(data:dict):
```

```
sliced_data_dict = {key: val for key, val in data_dict.items()
   ↪  if key > 2}
count,result_list = 0,[]
m = max(data)
q_0_hat = get_q_from_m(data,m)
l = log_likelihood(sliced_data_dict, m_0_hat, q_0_hat)
result_list.append((m,q_0_hat,l))


count +=1
m +=1
q_hat = get_q_from_m(data, m)
l = log_likelihood(sliced_data_dict, m, q_hat)
result_list.append((m,q_hat,l))


while result_list[count][-1] > result_list[count-1][-1]:
   count +=1
   m +=1
   q_hat = get_q_from_m(data, m)
   l = log_likelihood(sliced_data_dict, m, q_hat)
   result_list.append((m,q_hat,l))
return result_list[-2]
```

```
max_values = find_zmbinom_max_likelihood(data_dict)
sliced_data_dict = {key: val for key, val in data_dict.items() if
   ↪  key > 2}
N = np.sum([y for y in data_dict.values()])
p_0 = binom.pmf(k=0,n=max_values[0],p=max_values[1])
p_1 = binom.pmf(k=1,n=max_values[0],p=max_values[1])
p_2 = binom.pmf(k=2,n=max_values[0],p=max_values[1])
p_k = {k: N*((1-p_0_M-p_1_M-p_2_M)/(1-p_0-p_1-p_2)) * binom.pmf(k
   ↪ =k,n=max_values[0],p=max_values[1]) for k in sliced_data_
   ↪ dict.keys()}
df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims','
   ↪ Z12M␣Binomial␣expected'])
df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
df = pd.concat([df_0,df_1,df_2,df],sort=False)
df.loc[:,'Observed'] = list(data_dict.values())
df_final = df[['Number␣of␣Claims','Observed','Z12M␣Binomial␣
   ↪ expected']].copy()
```

Q Python Code for Fitting Dataset to a Z123M Negative Binomial Distribution

```
from itertools import chain
import numpy as np
from scipy.special import gammaln,factorial
from scipy.special import psi
from scipy.optimize import fmin_l_bfgs_b as optim
from scipy.stats import nbinom
import pandas as pd


def fit_zmnbinom(X, initial_params=None):
   infinitesimal = np.finfo(np.float).eps
   n0 = data_dict[0]
   n1 = data_dict[1]
   n2 = data_dict[2]
   n3 = data_dict[3]


   def log_likelihood(params, *args):
      r, p = params
      X = args[0]
```

```
N = X.size

#n0 = 370412

p_0_M = n0 / N

p_1_M = n1 / N

p_2_M = n2 / N

p_3_M = n3 / N

p_0 = p**r

p_1 = nbinom.pmf(k=1,n=r,p=p)

p_2 = nbinom.pmf(k=2,n=r,p=p)

p_3 = nbinom.pmf(k=3,n=r,p=p)

#l0 = n0 * np.log(p_0_M) + (N-n0)*np.log(1-p_0_M)

l0123 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.
   ↪ log(p_2_M)+n3 * np.log(p_3_M)+(N-n0-n1-n2-n3)*np.log
   ↪ (1-p_0_M-p_1_M-p_2_M-p_3_M)

sliced_data_dict = {key: val for key, val in data_dict.
   ↪ items() if key > 3}

n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p,loc=0) for k,n_k in
   ↪ sliced_data_dict.items()}

#summand = [n_k *(np.log(p_k if p_k < 1 else 1-
   ↪ infinitesimal)-np.log(1-p_0 )) for n_k,p_k in n_k_p_k
   ↪ .items()]
```

```
    summand = [n_k *(np.log(p_k if p_k < 1 else 1-infinitesimal
       ↪ )-np.log(1-p_0-p_1-p_2-p_3)) for n_k,p_k in n_k_p_k.
       ↪ items()]
    l4 = np.sum(summand)
    result = l0123+l4
    return -result
    def log_likelihood_deriv(params, *args):
    r, p = params
    X = args[0]
    N = X.size


    pderiv = (N*r)/p - np.sum(X)/(1-(p if p < 1 else 1-
       ↪ infinitesimal))
    rderiv = np.sum(psi(X + r)) \
       - N*psi(r) \
       + N*np.log(p)


    return np.array([-rderiv, -pderiv])


if initial_params is None:
    N = X.size
```

```
    p0M = n0 / N
    p1M = n1 / N
    p2M = n2 / N
    p3M = n3 / N
    m = np.mean(X)
    v = np.var(X)


    b = (v-m)/m
    #p0 = 1/(1+b)
    p0 = nbinom.pmf(k=0,n=1,p=1/(1+b))
    p1 = nbinom.pmf(k=1,n=1,p=1/(1+b))
    p2 = nbinom.pmf(k=2,n=1,p=1/(1+b))
    p3 = nbinom.pmf(k=3,n=1,p=1/(1+b))
    r0 = (m*(1-p0-p1-p2-p3))/(b*(1-p0M-p1M-p2M-p3M))
    initial_params = np.array([r0, p0])


bounds = [(0, None), (infinitesimal, 1)]
optimres = optim(log_likelihood,
             x0=initial_params,
             args=(X,),
             approx_grad=1,
```

```
                bounds=bounds)


    params = optimres[0]
    return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
        ↪ -1}


def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset


def main(claim_dict: dict):
    N = np.sum([val for val in claim_dict.values()])
    n0 = claim_dict[0]
    n1 = claim_dict[1]
    n2 = claim_dict[2]
    n3 = claim_dict[3]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
```

```
    p_3_M = n3 / N
    data = generate_nbinom_dataset(claim_dict)
    result = fit_zmnbinom(data)
    loglikelihood = get_loglikelihood(data_dict,data,result['r'],
                                result['prob'], n0, n1,n2,n3, p_0_M,
                                    ↪ p_1_M,p_2_M,p_3_M)
    final_result = result.copy()
    final_result['loglikelihood'] = loglikelihood
    return final_result


def get_loglikelihood(data_dict:dict, data:list,r:float,p:float,
    ↪ n0,n1,n2,n3, p_0_M, p_1_M, p_2_M,p_3_M):
    N = data.size
    p_0 = p**r
    p_1 = nbinom.pmf(k=1,n=r,p=p)
    p_2 = nbinom.pmf(k=2,n=r,p=p)
    p_3 = nbinom.pmf(k=3,n=r,p=p)
    l0123 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(
       ↪ p_2_M)+n3 * np.log(p_3_M)+(N-n0-n1-n2-n3)*np.log(1-p_0_M
       ↪ -p_1_M-p_2_M-p_3_M)
```

```
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 3}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p, loc=0) for k,n_k in
    ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2-p_3)) for n_
    ↪ k,p_k in n_k_p_k.items()]
    l4 = np.sum(summand)
    result = l0123+l4
    return result


def get_df(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    n3 = data_dict[3]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    p_3_M = n3 / N
```

```
sliced_data_dict = {key: val for key, val in data_dict.items()
   ↪  if key > 3}
result = main(data_dict)
p_0 = (result['prob'])**result['r']
p_1 = nbinom.pmf(k=1,n=result['r'],p=result['prob'])
p_2 = nbinom.pmf(k=2,n=result['r'],p=result['prob'])
p_3 = nbinom.pmf(k=3,n=result['r'],p=result['prob'])
p_k = {k: N*((1-p_0_M-p_1_M-p_2_M-p_3_M)/(1-p_0-p_1-p_2-p_3))
   ↪ * nbinom.pmf(k=k,n=result['r'],p=result['prob'],loc=0)
   ↪ for k in sliced_data_dict.keys()}
df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
   ↪ ','Z123M␣Negative␣Binomial␣expected'])
df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
df_3 = pd.DataFrame([[3, n3]], columns=df.columns)
df = pd.concat([df_0,df_1,df_2,df_3,df],sort=False)
df.loc[:,'frequency'] = list(data_dict.values())
df_final = df[['Number␣of␣Claims','frequency','Z123M␣Negative␣
   ↪ Binomial␣expected']].copy()
return df_final, result, p_0_M, p_1_M,p_2_M, p_3_M
```

```
if __name__ == '__main__':

      df = get_df(data_dict)
```

R Python Code for Fitting Dataset to a Z1234M Negative Binomial Distribution

```
from itertools import chain

import numpy as np

from scipy.special import gammaln,factorial

from scipy.special import psi

from scipy.optimize import fmin_l_bfgs_b as optim

from scipy.stats import nbinom

import pandas as pd




def fit_zmnbinom(X, initial_params=None):

   infinitesimal = np.finfo(np.float).eps

   n0 = data_dict[0]

   n1 = data_dict[1]
```

```
n2 = data_dict[2]

n3 = data_dict[3]

n4 = data_dict[4]


def log_likelihood(params, *args):

    r, p = params

    X = args[0]

    N = X.size

    p_0_M = n0 / N

    p_1_M = n1 / N

    p_2_M = n2 / N

    p_3_M = n3 / N

    p_4_M = n4 / N

    p_0 = p**r

    p_1 = nbinom.pmf(k=1,n=r,p=p)

    p_2 = nbinom.pmf(k=2,n=r,p=p)

    p_3 = nbinom.pmf(k=3,n=r,p=p)

    p_4 = nbinom.pmf(k=4,n=r,p=p)

    l01234 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.
        ↪ log(p_2_M) + n3 * np.log(p_3_M) + n4 * np.log(p_4_M)
        ↪ +(N-n0-n1-n2-n3-n4)*np.log(1-p_0_M-p_1_M-p_2_M-p_3_M-
```

```
        ↪ p_4_M)
    sliced_data_dict = {key: val for key, val in data_dict.
        ↪ items() if key > 4}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p,loc=0) for k,n_k in
        ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k if p_k < 1 else 1-infinitesimal
        ↪ )-np.log(1-p_0-p_1-p_2-p_3-p_4)) for n_k,p_k in n_k_p
        ↪ _k.items()]
    l5 = np.sum(summand)
    result = l01234+l5
    return -result


def log_likelihood_deriv(params, *args):
    r, p = params
    X = args[0]
    N = X.size


    pderiv = (N*r)/p - np.sum(X)/(1-(p if p < 1 else 1-
        ↪ infinitesimal))
    rderiv = np.sum(psi(X + r)) \
        - N*psi(r) \
```

```
        + N*np.log(p)

    return np.array([-rderiv, -pderiv])

if initial_params is None:
    N = X.size
    p0M = n0 / N
    p1M = n1 / N
    p2M = n2 / N
    p3M = n3 / N
    p4M = n4 / N
    m = np.mean(X)
    v = np.var(X)

    b = (v-m)/m
    #p0 = 1/(1+b)
    p0 = nbinom.pmf(k=0,n=1,p=1/(1+b))
    p1 = nbinom.pmf(k=1,n=1,p=1/(1+b))
    p2 = nbinom.pmf(k=2,n=1,p=1/(1+b))
    p3 = nbinom.pmf(k=3,n=1,p=1/(1+b))
    p4 = nbinom.pmf(k=4,n=1,p=1/(1+b))
```

```
        r0 = (m*(1-p0-p1-p2-p3-p4))/(b*(1-p0M-p1M-p2M-p3M-p4M))
        initial_params = np.array([r0, p0])


    bounds = [(0, None), (infinitesimal, 1)]
    optimres = optim(log_likelihood,
                 x0=initial_params,
                 args=(X,),
                 approx_grad=1,
                 bounds=bounds)


    params = optimres[0]
    return {'r': params[0], 'prob': params[1],'beta':(1/params[1])
        ↪ -1}


def generate_nbinom_dataset(X:dict):
    uncollapsed_set = [[x]*y for x,y in X.items()]
    nbinom_data = list(chain.from_iterable(uncollapsed_set))
    dataset = np.array(nbinom_data)
    return dataset


def main(claim_dict: dict):
```

```
N = np.sum([val for val in claim_dict.values()])

n0 = claim_dict[0]

n1 = claim_dict[1]

n2 = claim_dict[2]

n3 = claim_dict[3]

n4 = claim_dict[4]

p_0_M = n0 / N

p_1_M = n1 / N

p_2_M = n2 / N

p_3_M = n3 / N

p_4_M = n4 / N

data = generate_nbinom_dataset(claim_dict)

result = fit_zmnbinom(data)

loglikelihood = get_loglikelihood(data_dict,data,result['r'],

                          result['prob'], n0, n1,n2,n3,n4, p_0

                            ↪ _M,p_1_M,p_2_M,p_3_M,p_4_M)

final_result = result.copy()

final_result['loglikelihood'] = loglikelihood

return final_result
```

```
def get_loglikelihood(data_dict:dict, data:list,r:float,p:float,
    ↪ n0,n1,n2,n3,n4, p_0_M, p_1_M, p_2_M,p_3_M,p_4_M):
    N = data.size
    p_0 = p**r
    p_1 = nbinom.pmf(k=1,n=r,p=p)
    p_2 = nbinom.pmf(k=2,n=r,p=p)
    p_3 = nbinom.pmf(k=3,n=r,p=p)
    p_4 = nbinom.pmf(k=4,n=r,p=p)
    l01234 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log
       ↪ (p_2_M) + n3 * np.log(p_3_M) + n4 * np.log(p_4_M)+(N-n0-
       ↪ n1-n2-n3-n4)*np.log(1-p_0_M-p_1_M-p_2_M-p_3_M-p_4_M)
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 4}
    n_k_p_k = {n_k: nbinom.pmf(k=k,n=r,p=p, loc=0) for k,n_k in
       ↪ sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2-p_3-p_4))
       ↪ for n_k,p_k in n_k_p_k.items()]
    l5 = np.sum(summand)
    result = l01234+l5
    return result
```

```
def get_df(data_dict):

   N = np.sum([val for val in data_dict.values()])

   n0 = data_dict[0]

   n1 = data_dict[1]

   n2 = data_dict[2]

   n3 = data_dict[3]

   n4 = data_dict[4]

   p_0_M = n0 / N

   p_1_M = n1 / N

   p_2_M = n2 / N

   p_3_M = n3 / N

   p_4_M = n4 / N

   sliced_data_dict = {key: val for key, val in data_dict.items()
      ↪ if key > 4}

   result = main(data_dict)

   p_0 = (result['prob'])**result['r']

   p_1 = nbinom.pmf(k=1,n=result['r'],p=result['prob'])

   p_2 = nbinom.pmf(k=2,n=result['r'],p=result['prob'])

   p_3 = nbinom.pmf(k=3,n=result['r'],p=result['prob'])

   p_4 = nbinom.pmf(k=4,n=result['r'],p=result['prob'])
```

```
    p_k = {k: N*((1-p_0_M-p_1_M-p_2_M-p_3_M-p_4_M)/(1-p_0-p_1-p_2-
       ↪ p_3-p_4)) * nbinom.pmf(k=k,n=result['r'],p=result['prob'
       ↪ ],loc=0) for k in sliced_data_dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','Z1234M␣Negative␣Binomial␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
    df_3 = pd.DataFrame([[3, n3]], columns=df.columns)
    df_4 = pd.DataFrame([[4, n4]], columns=df.columns)
    df = pd.concat([df_0,df_1,df_2,df_3,df_4,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','Z1234M␣Negative
       ↪ ␣Binomial␣expected']].copy()
    return df_final, result, p_0_M, p_1_M,p_2_M, p_3_M,p_4_M


if __name__ == '__main__':
       df = get_df(data_dict)
```

S Python Code for Fitting Dataset to a Z123M Geometric Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.stats import geom
def getmean(X):
   X = geom_dataset(X)
   beta = np.mean(X)
   return beta


def getbeta_hat(N, data_dict, n0, n1, n2, n3):
   mean = getmean(data_dict)
   beta_hat = ((N*mean -n1 - n2 - n3)/(N-n0-n1-n2-n3)) - 4
   return beta_hat


def geom_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
   geom_data = list(chain.from_iterable(uncollapsed_set))
   dataset = np.array(geom_data)
```

```
    return dataset


def get_loglikelihood(X,n0,n1,n2,n3,p_0_M,p_1_M,p_2_M,p_3_M, beta
    ↪ _hat:float):
    N = np.sum([val for val in X.values()])
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/(1+beta_hat)**2
    p_2 = (beta_hat**2)/((1+beta_hat)**3)
    p_3 = (beta_hat**3)/((1+beta_hat)**4)
    l0123 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log(
       ↪ p_2_M)+n3 * np.log(p_3_M)+(N-n0-n1-n2-n3)*np.log(1-p_0_M
       ↪ -p_1_M-p_2_M-p_3_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 3}
    n_k_p_k = {n_k: (beta_hat**k)/((1+beta_hat)**(k+1)) for k,n_k
       ↪ in sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2-p_3)) for n_
       ↪ k,p_k in n_k_p_k.items()]
    l4 = np.sum(summand)
    result = l0123+l4
```

```
    return result


def main(data_dict):
    N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    n3 = data_dict[3]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    p_3_M = n3 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
    ↪  if key > 3}
    N_modified = np.sum([val for val in sliced_data_dict.values()
    ↪ ])
    beta_hat = getbeta_hat(N, data_dict, n0, n1, n2, n3)
    likelihood = get_loglikelihood(data_dict,n0,n1,n2,n3,p_0_M,p_1
    ↪ _M,p_2_M,p_3_M,beta_hat)
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/((1+beta_hat)**2)
```

```
    p_2 = (beta_hat**2)/((1+beta_hat)**3)
    p_3 = (beta_hat**3)/((1+beta_hat)**4)
    p_k = {k: N*((1-p_0_M-p_1_M-p_2_M-p_3_M)/(1-p_0-p_1-p_2-p_3))*
       ↪ ((beta_hat**k)/(1+beta_hat)**(k+1)) for k in sliced_data
       ↪ _dict.keys()}
    df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
       ↪ ','ZM_4␣Geometric␣expected'])
    df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
    df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
    df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
    df_3 = pd.DataFrame([[3, n3]], columns=df.columns)
    df = pd.concat([df_0,df_1,df_2,df_3,df],sort=False)
    df.loc[:,'frequency'] = list(data_dict.values())
    df_final = df[['Number␣of␣Claims','frequency','ZM_4␣Geometric␣
       ↪ expected']].copy()
    return df_final, likelihood, beta_hat, p_0_M, p_1_M, p_2_M, p_
       ↪ 3_M


if __name__ == '__main__':
       df = main(data_dict)
```

T Python Code for Fitting Dataset to a Z1234M Geometric Distribution

```
from itertools import chain
import numpy as np
import pandas as pd
from scipy.stats import geom
def getmean(X):
   X = geom_dataset(X)
   beta = np.mean(X)
   return beta


def getbeta_hat(N, data_dict, n0, n1, n2, n3, n4):
   mean = getmean(data_dict)
   beta_hat = ((N*mean -n1 - n2 - n3 - n4)/(N-n0-n1-n2-n3-n4)) -
      ↪ 5
   return beta_hat


def geom_dataset(X:dict):
   uncollapsed_set = [[x]*y for x,y in X.items()]
   geom_data = list(chain.from_iterable(uncollapsed_set))
```

```
    dataset = np.array(geom_data)
    return dataset


def get_loglikelihood(X,n0,n1,n2,n3,n4,p_0_M,p_1_M,p_2_M,p_3_M,p_
    ↪ 4_M,beta_hat:float):
    p_0 = 1/(1+beta_hat)
    p_1 = beta_hat/(1+beta_hat)**2
    p_2 = (beta_hat**2)/((1+beta_hat)**3)
    p_3 = (beta_hat**3)/((1+beta_hat)**4)
    p_4 = (beta_hat**4)/((1+beta_hat)**5)
    l01234 = n0 * np.log(p_0_M) + n1 * np.log(p_1_M) + n2 * np.log
       ↪ (p_2_M) + n3 * np.log(p_3_M) + n4 * np.log(p_4_M)+(N-n0-
       ↪ n1-n2-n3-n4)*np.log(1-p_0_M-p_1_M-p_2_M-p_3_M-p_4_M)
    sliced_data_dict = {key: val for key, val in X.items() if key
       ↪ > 4}
    n_k_p_k = {n_k: (beta_hat**k)/((1+beta_hat)**(k+1)) for k,n_k
       ↪ in sliced_data_dict.items()}
    summand = [n_k *(np.log(p_k)-np.log(1-p_0-p_1-p_2-p_3-p_4))
       ↪ for n_k,p_k in n_k_p_k.items()]
    l5 = np.sum(summand)
    result = l01234+l5
```

```
    return result
if __name__ == '__main__':
        N = np.sum([val for val in data_dict.values()])
    n0 = data_dict[0]
    n1 = data_dict[1]
    n2 = data_dict[2]
    n3 = data_dict[3]
    n4 = data_dict[4]
    p_0_M = n0 / N
    p_1_M = n1 / N
    p_2_M = n2 / N
    p_3_M = n3 / N
    p_4_M = n4 / N
    sliced_data_dict = {key: val for key, val in data_dict.items()
       ↪  if key > 4}
    N_modified = np.sum([val for val in sliced_data_dict.values()
       ↪ ])
    beta_hat = getbeta_hat(N, data_dict, n0, n1, n2, n3, n4)
    likelihood = get_loglikelihood(data_dict,n0,n1,n2,n3,n4,p_0_M,
       ↪ p_1_M,p_2_M,p_3_M,p_4_M,beta_hat)
```

```
p_0 = 1/(1+beta_hat) #geom.pmf(k=0, p = 1/(1+beta_hat))
p_1 = beta_hat/((1+beta_hat)**2)
p_2 = (beta_hat**2)/((1+beta_hat)**3)#geom.pmf(k=1, p = 1/(1+
    ↪ beta_hat))
p_3 = (beta_hat**3)/((1+beta_hat)**4)
p_4 = (beta_hat**4)/((1+beta_hat)**5)
p_k = {k: N*((1-p_0_M-p_1_M-p_2_M-p_3_M-p_4_M)/(1-p_0-p_1-p_2-
    ↪ p_3-p_4))*((beta_hat**k)/(1+beta_hat)**(k+1)) for k in
    ↪ sliced_data_dict.keys()}
df = pd.DataFrame(list(p_k.items()),columns=['Number␣of␣Claims
    ↪ ','ZM_5␣Geometric␣expected'])
df_0 = pd.DataFrame([[0, n0]], columns=df.columns)
df_1 = pd.DataFrame([[1, n1]], columns=df.columns)
df_2 = pd.DataFrame([[2, n2]], columns=df.columns)
df_3 = pd.DataFrame([[3, n3]], columns=df.columns)
df_4 = pd.DataFrame([[4, n4]], columns=df.columns)
df = pd.concat([df_0,df_1,df_2,df_3,df_4,df],sort=False)
df.loc[:,'frequency'] = list(data_dict.values())
df_final = df[['Number␣of␣Claims','frequency','ZM_5␣Geometric␣
    ↪ expected']].copy()
```