Modeling the Premium in Non-Life Insurance
A Comparison of Generalized Linear and Generalized Linear Mixed Models

Tam Ha
Master’s Thesis, Autumn 2017
This master’s thesis is submitted under the master’s program *Modelling and Data Analysis*, with program option *Finance, Insurance and Risk*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group $E_8$, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.
Abstract

General linear models (GLMs) are based on several assumptions. One of them is that the observations are independent. This may not be the case for insurance data, and could possibly affect the estimation of the insurance premium. Generalized linear mixed models (GLMMs) are an extension of GLMs that include both fixed and random effects. The addition of random effects makes the GLMMs able to account for heterogeneity among subjects, due to unobserved characteristics. Hence, the models are more flexible, but also more complicated. The primary objective was to measure the performance of the two approaches regarding insurance data. To do that, we simulated datasets that resemble real insurance data, based on the characteristics of the insurance data for automobiles. Each dataset consists of claim counts, sizes and rating factors. Some of them were also affected by random effects that reflect the individual behavior characteristics which can not be captured by the fixed effects. The data were simulated for 1000 policyholders over a five-year period. By combining different distributions for the claim counts and sizes, with and without random effects, we were able to draw more general conclusions about the models. Overall, insurance data seems to be well estimated by GLMs. GLMMs on the other hand are more complicated, and parameter estimation shows to be more challenging. They are also much more time-consuming than GLMs. For analyzing cluster-correlated data, theory favors GLMMs. Nevertheless our simulation studies support the use of GLMs in the case of insurance data.
Acknowledgement

Firstly, I wish to thank my supervisor Ingrid Hobæk Haff for introducing me the interesting topic, and for all guidance, advice and feedback during the process of writing this thesis. I would also like to thank Charlotte Bruland for proofreading my thesis and for the continuous support and encouragement during the study.
# List of Tables

1. Some distributions from the exponential dispersion family .......... 10
2. Link functions ........................................................................ 11
3. Models for the insurance data .................................................. 25
4. Fixed effects for the model ....................................................... 26
5. Parameter specifications .......................................................... 30
6. Notation in the presentation of the results ................................. 32
7. Model comparison Part 1: Poisson-gamma fixed effects ............... 32
8. Model comparison Part 1: Poisson-gamma mixed effects ............... 33
10. Model comparison Part 1: Poisson-inverse G. mixed effects ........ 35
11. Model comparison Part 1: negative binomial-gamma fixed effects 37
12. Model comparison Part 1: negative binomial-gamma mixed effects 38
13. Model comparison Part 1: negative binomial-inverse G. fixed effects 39
14. Model comparison Part 1: negative binomial-inverse G. mixed effects 40
15. Model comparison Part 2: Poisson-gamma fixed effects ............... 44
16. Model comparison Part 2: Poisson-gamma mixed effects ............. 44
17. Model comparison Part 2: Poisson-inverse G. fixed effects .......... 45
18. Model comparison Part 2: Poisson-inverse G. mixed effects ........ 45
1. Introduction

Accidents occur every day. Some more severe than others. We cannot prevent accidents from happening. We can, however, protect ourselves against great financial loss if one does occur. Insurance is a way of protection against great economic losses. As a customer one is looking for an insurance policy that covers as much as possible at a low cost. As an insurer you want to make money, and at the same time keep and gain customers. If the cost of insurance is too high, customers will go to a company that is cheaper. In that way you are losing customers to the competition. If the price is too low, the insurer can risk more money going out, than coming in. As a result the company loses money or, in the worst case scenario, becomes insolvent. Individuals that are involved in many accidents, often get high premiums. When the price of insurance is too low, you will also risk attracting customers that have a high rate of claims, since the price they would get with other companies would be higher. This would fuel the initial problem, by increasing the number of claims. Therefore pricing of insurance is crucial, and this is the motivation for the thesis.

There are several factors to consider when pricing insurance. Data about customers are important, but also expertise in the market. In this thesis the focus will fall on actuarial calculations of the pure premium. Generalized linear models (GLMs) are widely used in the insurance industry. The models are based on several assumptions. One of them is that the data are independent of each other. In many statistical problems, the assumption is violated. Longitudinal data in non-life insurance for instance, consists of data where claims reported by the same policyholder often are considered to be somehow related. This can be caused by individual characteristics such as driving skills, swiftness of reflexes, power of concentration or aggressiveness behind the wheel. In theory, this assumption is very important, but exactly how important it is in practice is one of the questions we aim to answer in this thesis. Generalized linear mixed models (GLMMs) extend GLMs by allowing for random effects that take the correlation between the data from the same group or cluster into account, and should be a solution to get around this assumption. A cluster means an individual in our context. One can question whether a GLMM could be able to estimate the pure premium more accurately than GLM.

Insurance is an arrangement designed to protect a policyholder from financial losses and can roughly be divided into two main types. The first one is called life insurance and is related to the risk of an individual life, for instance, death, disability and retirement. The second is non-life insurance and deals with property losses or damages. Examples are insurance for home, travel and automobile. Pricing methods of the two types of insurance differ from each other, and our focus will be on the latter case. More specifically, we will concentrate on automobile insurance.

An insurance policy is the financial contract between an insurer (e.g. an insurance company) and a policyholder. The insurer takes all or part of the risk and demands an agreed amount of money, called insurance premium. This could be either a series of payments
over time or a single payment. For the insurance premium, overhead costs (administrative expenses, capital costs etc.) and profit are taken into account. The part of the insurance premium that corresponding to the risk is called pure premium. It represents the expected payout for reported claims that occurred during the policy period.

In non-life insurance, the pure premium represents the expected cost of the claims that are reported by the policyholders during the insurance period. The mathematical expression can be given by

\[ \Pi_{pu} = E[N|x] \cdot E[Z|x], \]

where \( E[N|x] \) is the conditional expected number of claims during the policy period and \( E[Z|x] \) the conditional expected cost per claim during the same period. \( x \) is here the risk factors such as age, gender, geographical area and so on. A common strategy is to model the components on the right hand side separately.

GLMs have routinely been used for several decades in applied statistics, and finding material on GLMs is not difficult. See for instance Madsen and Thyregod (2010) or Agresti (2015). The GLM framework has also been proven to be efficient in actuarial statistics. An introduction of GLMs in the insurance context can be found in Ohlsson and Johansson (2010) or De Jong et al. (2008). Numerous studies also applied the models for insurance claims modeling. David (2015) illustrate the use of GLMs for calculation of the insurance premium. Zehnwirth (1994) and Murphy et al. (2000) provide a more detailed description of the pricing process, i.e. insurance rate making. The concept of random effects in the context of actuarial science has also been used in various studies. Boucher and Denuit (2006) compare the use of fixed and random effects in Poisson regression, in a case study with a Belgian motor third party liability insurance portfolio. Boucher et al. (2008) illustrate the different use of random effects for modeling insurance claim numbers based on a sample of the automobile portfolio of a company operating in Spain with data from 1991-1998. GLMMs are also used in many other fields. Some examples are biology, psychology, medicine, ecology, engineering. For an excellent illustration of the application of GLMMs in ecology and evolution, see Bolker et al. (2009).

From previous studies, it was shown that GLMMs can improve the estimates of data. Studies show modeling for one set only. We therefore want to see if we experience the same improvement in a more general case, regarding insurance data. More precisely, the aim of the study is to see how well insurance data can be estimated by GLMs and GLMMs, both when affected by random effects and not, according to the premium. The study also aspires to find out how much the estimates of the pure premium are affected when using GLMs in the case of random effects, as well as uncovering the consequence of using the "wrong" model. Lastly we want to see if GLMMs give noticeably better estimates when the true model is affected by random effects.
1. INTRODUCTION

We start, in Chapter 2, by introducing the basic concepts and theory of the classical linear model (LM), the GLMs and the GLMMs. This is the core material of the thesis. Chapter 3 is dedicated to the choice of the distributions, as well as regression models used for modeling. In Chapter 4, we focus on how data are simulated. The metrics used for model comparison are also presented in this chapter. Chapter 5 comprises the results and discussion of the simulation study. Some important findings and concluding remarks are given in Chapter 6. Most of the notations and acronyms used throughout the thesis can be found in Appendix A. The computer program that was used for the simulations in this study is shown in Appendix B.
2. Theory

In this chapter we present the theory that forms the basis for the choice of regression models that are used in our modeling. We will introduce the LM, GLMs and GLMMs. An LM is the simplest form of linear regression, while GLM is a flexible generalization of the LM. GLMM is an extension of GLM that allows random effects.

2.1 Classical linear model

Regression analysis constitutes an important part of statistical modeling. The general objective is to analyze the relationship between a random variable, called response, and one or more explanatory variables, called covariates. The LM has some limitations that makes it unsuitable in many situations. However, it is important to know the basics of the LM to understand the models that are the focus of this thesis. Later on we will see that GLMs and GLMMs can be used, and are more preferable to analyze insurance data. This section is meant to give a brief review of the LM. For a more detailed description, see e.g. Devore and Berk (2011) or Agresti (2015). An LM is on the form

\[ Y_i = \beta_0 + \beta_1 x_{i1} + .. + \beta_k x_{ik} + \epsilon_i = x_i \beta + \epsilon_i, \quad i = 1, \ldots, n \]

where \( x_i = [1, x_{i1}, \ldots, x_{ik}] \) is a row vector of covariates and \( \beta = [\beta_0, \beta_1, \ldots, \beta_k]^t \) is a column vector of coefficients. The model is called linear because the response variable \( Y_i \) can be expressed by a linear combination of the covariates. The errors \( \epsilon_1, \ldots, \epsilon_n \) are assumed to be independent and identically distributed (iid), typically with a normal distribution with mean 0 and variance \( \sigma^2 \). Other distributions for \( \epsilon_i \) could be an alternative, but are rarely used in practice. Followed by the assumptions for \( \epsilon_i \), we have that \( Y_1, \ldots, Y_n \) are independent and normally distributed with the conditional expected value and variance given by

\[
E[Y_i|x_i] = E[x_i \beta + \epsilon_i|x_i] = x_i \beta + E[\epsilon_i] = x_i \beta = \mu_i
\]

and

\[
\text{var}[Y_i|x_i] = \text{var}[x_i \beta + \epsilon_i|x_i] = \text{var}[\epsilon_i] = \sigma^2.
\]

As short hand notation, we use the expression \( Y_i \sim N(x_i \beta, \sigma^2) \). In matrix form, the model is given by

\[
Y = X \beta + \epsilon,
\]

where \( Y = [Y_1, \ldots, Y_n]^t \), \( X = [x_1^t, \ldots, x_n^t]^t \) and \( \epsilon = [\epsilon_1, \ldots, \epsilon_n]^t \). \( X \) is sometimes called a design matrix. When \( \epsilon_1, \ldots, \epsilon_n \) are assumed to be iid normally distributed with mean 0 and variance \( \sigma^2 \), we have that \( Y \) is multivariate normally distributed with mean vector \( \mu = X \beta \) and variance-covariance matrix \( \sigma^2 I_n \), where \( I_n \) is an identity matrix of size \( n \times n \). For simplicity we use \( Y \sim N(X \beta, \sigma^2 I_n) \).

Remark 2.1. (Normal linear model)

When \( \epsilon_1, \ldots, \epsilon_n \) are assumed to be normally distributed with mean 0 and variance \( \sigma^2 \), the model is referred to as the normal linear model.


2. THEORY

2.1.1 Estimation

The standard approach for the normal linear model is the least squares estimation (LSE) method and will be outlined in this section. Assume that we have an observed vector $\mathbf{y} = [y_1, .., y_n]^t$ for the random vector $\mathbf{Y}$. The idea behind LSE is to find an estimate for $\mathbf{\beta}$ that minimizes the sum of the squared errors

$$ f(\beta_0, .., \beta_k) = \|\mathbf{y} - \mathbf{\mu}\|^2 = \sum_{i=1}^{n}(y_i - \mu_i)^2. $$

Let us start with a simple regression model, i.e. the case of only one covariate. The model is on the form

$$ Y_i = \beta_0 + \beta_1 x_i + \epsilon_i $$

with the expected value $\mu_i = \beta_0 + \beta_1 x_i$, where $\beta_0$ and $\beta_1$ are unknown parameters. The sum of the squared errors is given by

$$ f(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2. $$

Let $\hat{\beta}_0$ and $\hat{\beta}_1$ be the values that minimize $f(\beta_0, \beta_1)$. To obtain $\hat{\beta}_0$ and $\hat{\beta}_1$ we find the first derivatives of $f(\beta_0, \beta_1)$ with respect to $\beta_0$ and $\beta_1$ respectively and set them equal to zero, i.e.

$$ \frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = -2 \sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)) = 0 $$

$$ \frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} = -2 \sum_{i=1}^{n}2(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)(x_i)) = 0. $$

With some manipulations, we can rewrite the equations to the form

$$ n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i $$

$$ \hat{\beta}_0 \sum_{i=1}^{n} x_i + \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i. $$

These are often called normal equations. By the use of basic calculus it can be shown that the LSEs for $\beta_0$ and $\beta_1$ are

$$ \hat{\beta}_1 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} $$

and

$$ \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, $$

where

$$ \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \quad \text{and} \quad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i. $$

Hence, the fitted values are $\hat{\mu}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ for $i = 1, .., n$. Now assume a multiple model for the response variable, i.e. $Y_i = \beta_0 + \beta_1 x_{i1} + .. + \beta_k x_{ik} + \epsilon_i$. The sum of the squared errors is then

$$ f(\beta_0, \beta_1, .., \beta_k) = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + .. + \beta_k x_{ik}))^2. $$
The idea behind the LSE is the same for the multiple models, but now we have a system of \( k + 1 \) equations to solve. This can be written on matrix form as

\[
f(\beta_0, \beta_1, \ldots, \beta_k) = (y - X\beta)'(y - X\beta) = y'y - 2y'X\beta + \beta'X'X\beta.
\]

When using LSE to estimate \( \beta_0, \ldots, \beta_k \), the normal equations in matrix form are \( X'X \hat{\beta} = X'y \), such that

\[
(X'X)^{-1}X'y = (X'X)^{-1}X'y
\]

(2.1)

and hence,

\[
\hat{\mu} = X\hat{\beta}.
\]

(2.2)

For GLMs and GLMMs the parameters are usually estimated using maximum likelihood. It is in general not possible to obtain an analytic form for the maximum likelihood estimate (MLE), and numerical optimization is needed.

**Remark 2.2. (Equivalence of the LSE and the MLE)**

In the case when the errors are assumed to be normally distributed, the LSE coincides with the MLE.

**Proof.** To prove that LSE is equivalent to MLE for the normal linear model, we first define the log-likelihood function for the normal distribution

\[
\ell(\beta) = \log \prod_{i=1}^{n} f(y_i) = \sum_{i=1}^{n} \log \{ f(y_i) \}
\]

\[
= \sum_{i=1}^{n} \left\{ \frac{1}{2} \left( -\log(2\pi) - 2\log(\sigma) \right) - \frac{1}{2\sigma^2} (y_i - x_i'\beta)^2 \right\}
\]

\[
= \frac{n}{2} \left\{ -\log(2\pi) - 2\log(\sigma) \right\} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i'\beta)^2
\]

\[
= \frac{n}{2} \left\{ -\log(2\pi) - 2\log(\sigma) \right\} - \frac{1}{2\sigma^2} (y - X\beta)'(y - X\beta).
\]

Taking the first derivative of the log-likelihood function with respect to \( \beta \) and setting it equal to zero we have

\[
-2\frac{1}{2\sigma^2}X'(y - X\hat{\beta}) = 0
\]

\[
X'X\hat{\beta} = X'y
\]

\[
\hat{\beta} = (X'X)^{-1}X'y
\]

and

\[
\hat{\mu} = X\hat{\beta}
\]

which is equal to (2.1) and (2.2). That is, maximizing the log-likelihood is equivalent to LSE in (2.1). □
2. Theory

When $\epsilon_1, \ldots, \epsilon_n$ are not normally distributed, the solution in (2.1) is still LSE, but the MLE will not have the same form.

2.2 Generalized linear models

GLMs have similarities to the LM, but in addition, they contain some extensions. For the LM we assumed an additive model structure

$$\mu_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}.$$ 

For GLMs the expected value of a response variable is related to the covariates through a function $g(\cdot)$. The model is given by

$$\eta_i = g(\mu_i) = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik} = x_i \beta, \quad i = 1, \ldots, n$$

where $\eta_i$ is called a linear predictor and $g(\cdot)$ a link function. The model given in matrix form is

$$\eta = g(\mu) = X \beta,$$

where $\eta = [\eta_1, \ldots, \eta_n]^t$. The link function makes GLMs more flexible and is often chosen based on knowledge of the response distribution, i.e. how the variables are related to one another. In actuarial modeling for instance, it is, in general more reasonable to have a multiplicative model structure, and GLMs allow this structure through a log-link

$$\log(\mu_i) = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}, \quad \text{or} \quad \mu = \exp(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}).$$

In addition, the log-link makes sure that the predictions are always positive. This is commonly the case in risk modeling. Another example where GLM is more appropriate than the LM is when modeling probabilities, where it is common to use a logit link to ensure that the predictions of the probability lie between zero and one.

For GLMs, $Y_1, \ldots, Y_n$ are assumed to be independent and follow the same distribution from the exponential dispersion family.

2.2.1 Exponential dispersion family

A probability distribution that can be given on the form (2.3) is a member of the exponential dispersion family. For the LM, we typically assumed to have a normal distribution. In many situations, the response variable is non-normal and may also be a count variable. GLMs assumed the distribution to be in the exponential dispersion family.

Definition 2.1. (Exponential dispersion family)

A probability distribution is a member of the exponential dispersion family if the density function can be expressed by

$$f(y_i; \theta_i, \phi) = \exp\left(\frac{y_i \theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\right),$$

(2.3)
2. THEORY

where $\theta_i$ is called the natural parameter, $\phi$ the dispersion parameter, $a(\phi)$ the dispersion function and $c(y_i, \phi)$ and $b(\theta_i)$ are specific functions for different distributions.

Usually, we have that $a(\phi) = \phi$ or $a(\phi) = \phi/w_i$ for a known weight $w_i$. Examples of distributions in the exponential dispersion family are the Poisson, binomial, normal, exponential and gamma distribution. Table 1 shows some distributions on the form (2.3).

**Remark 2.3. (Mean and variance expressions)**

For the exponential dispersion family, it can be shown that $E[Y_i] = b'(\theta_i)$ and $\text{var}[Y_i] = a(\phi)b''(\theta_i)$.

**Proof.** The density function for all distributions in the exponential dispersion family is on the general form

$$f(y_i; \theta_i, \phi) = \exp\left(\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\right).$$

With the first derivative of the function $f(\cdot)$

$$\frac{\partial f(y_i; \theta_i, \phi)}{\partial \theta_i} = \frac{y_i - b'(\theta_i)}{a(\phi)} f(y_i; \theta_i, \phi),$$

we have the integral of the left hand side

$$\int \frac{\partial f(y_i; \theta_i, \phi)}{\partial \theta_i} dy_i = \frac{\partial}{\partial \theta_i} \int f(y_i; \theta_i, \phi) dy_i = \frac{\partial}{\partial \theta_i} 1 = 0$$

and the integral of the right hand side

$$\int \frac{y_i - b'(\theta_i)}{a(\phi)} f(y_i; \theta_i, \phi) dy_i$$

$$= \frac{1}{a(\phi)} \int \{ (y_i - b'(\theta_i)) f(y_i; \theta_i, \phi) dy_i \}$$

$$= \frac{1}{a(\phi)} \left\{ \int y_i f(y_i; \theta_i, \phi) dy_i - b'(\theta_i) \int f(y_i; \theta_i, \phi) dy_i \right\}$$

$$= \frac{1}{a(\phi)} \left\{ E[Y_i] - b'(\theta_i) \right\}.$$

This gives us the equation

$$0 = \frac{1}{a(\phi)} \left\{ E[Y_i] - b'(\theta_i) \right\}$$

and hence

$$E[Y_i] = b'(\theta_i). \quad (2.4)$$
2. THEORY

For the variance, we need the second derivative of the function $f$

$$\frac{\partial^2 f(y_i; \theta_i, \phi)}{\partial \theta_i^2} = -\frac{b''(\theta_i)}{a(\phi)} f(y_i; \theta_i, \phi) + \left\{ \frac{y_i - b'(\theta_i)}{a(\phi)} \right\}^2 f(y_i; \theta_i, \phi).$$

With the integral of the left hand side

$$\int \frac{\partial^2 f(y_i; \theta_i, \phi)}{\partial \theta_i^2} dy_i = \frac{\partial^2}{\partial \theta_i^2} \int f(y_i; \theta_i, \phi) dy_i = \frac{\partial^2}{\partial \theta_i^2} 1 = 0$$

and the integral of the right hand side

$$\int \left\{ -\frac{b''(\theta_i)}{a(\phi)} f(y_i; \theta_i, \phi) + \left( \frac{y_i - b'(\theta_i)}{a(\phi)} \right)^2 f(y_i; \theta_i, \phi) \right\} dy_i$$

$$= -\frac{b''(\theta_i)}{a(\phi)} \int f(y_i; \theta_i, \phi) dy_i + \int \left( \frac{y_i - b'(\theta_i)}{a(\phi)} \right)^2 f(y_i; \theta_i, \phi) dy_i$$

$$= -\frac{b''(\theta_i)}{a(\phi)} + \frac{1}{a(\phi)^2} \int \{y_i - E[Y_i]\}^2 f(y_i; \theta_i, \phi) dy_i$$

$$= -\frac{b''(\theta_i)}{a(\phi)} + \frac{1}{a(\phi)^2} E \{ (y_i - E[Y_i])^2 \}$$

we have the equation

$$0 = -\frac{b''(\theta_i)}{a(\phi)} + \frac{\text{var}[Y_i]}{a(\phi)^2}$$

and hence,

$$\text{var}[Y_i] = a(\phi)b''(\theta_i).$$

(2.5)
Table 1: Some distributions from the exponential dispersion family

**Poisson:**

\[ f(y; \lambda) = \frac{\lambda^y}{y!} e^{-\lambda} \]

\[ = \exp \left\{ y \log(\lambda) - \lambda + (- \log(y!)) \right\} \]

where \( a(\phi) = 1, \theta = \log(\lambda) \Rightarrow \lambda = \exp(\theta), \ b(\theta) = \exp(\theta) \) and \( c(y) = -\log(y!) \).

**gamma:**

\[ f(y; k, \mu) = \frac{(k/\mu)^k}{\Gamma(k)} \exp \left( \frac{-ky}{\mu} \right) y^{k-1} \]

\[ = \exp \left\{ k \log(k) + k \log \left( \frac{1}{\mu} \right) - \log(\Gamma(k)) - \frac{ky}{\mu} + (k - 1) \log(y) \right\} \]

where \( a(\phi) = \phi = \frac{1}{k}, \theta = \frac{-1}{\mu} \Rightarrow \mu = \frac{1}{\phi}, \ b(\theta) = -\log(-\theta) \) and \( c(y) = \left( \frac{1}{\phi} - 1 \right) \log(y) - \frac{\log(\phi)}{\phi} - \log \left( \Gamma \left( \frac{1}{\phi} \right) \right) \)

**NORMAL:**

\[ f(y; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(y - \mu)^2}{2\sigma^2} \right\} \]

\[ = \exp \left\{ \log \left( \frac{1}{\sqrt{2\pi\sigma^2}} \right) - \frac{y^2 + 2y\mu - \mu^2}{2\sigma^2} \right\} \]

\[ = \exp \left\{ -\frac{1}{2} \log(2\pi\sigma^2) - \frac{y^2}{2\sigma^2} + y\mu \frac{1}{2\sigma^2} - \frac{\mu^2}{2\sigma^2} \right\} \]

\[ = \exp \left\{ \frac{y\mu - \frac{1}{2}\mu^2}{\sigma^2} + \left( -\frac{1}{2} \log(2\pi\sigma^2) - \frac{y^2}{2\sigma^2} \right) \right\} \]

where \( a(\phi) = \phi = \sigma^2, \theta = \mu, \ b(\theta) = \frac{1}{2}\theta^2 \) and \( c(y) = -\frac{1}{2} \log(2\pi\sigma^2) - \frac{y^2}{2\sigma^2} \)
2. THEORY

2.2.2 Link function

For the LM we have that the response variable can be written as a linear combination of the covariates. For GLMs, the relationship is linear through the link function \( g(\cdot) \). Note that the LM is just a special case of GLMs where the response variable is typically assumed to be normal and \( g(\mu_i) = \mu_i \), i.e. \( g(\cdot) \) is the identity link function. There are many other link functions. Madsen and Thyregod (2010) presented the commonly used functions and some of them are listed in Table 2.

Definition 2.2. (The link function)
The relation between the mean value \( \mu_i \) and the linear prediction \( \eta_i \) can be described by a smooth and monotone function \( g(\cdot) \) called the link function.

In the case of \( g(\mu) = \theta = x_i \beta \), the link function \( g(\cdot) \) is called a canonical link. For example, we have that the log-link is the canonical link for the Poisson distribution and the identity-link is the canonical link for the normal distribution. Using the canonical link simplifies the parameter estimation, and hence, it tends to be used by default in statistical software. Although the canonical links have some theoretical advantages, they will not necessarily provide the best fit available to a given dataset. However, with the modern computing, the use of the canonical link is not crucial.

Remark 2.4. (Inverse mapping)
The mean value \( \mu_i \) can be expressed by the inverse mapping \( g(\cdot)^{-1} \) of the linear prediction \( \eta_i \)

\[
\mu = g^{-1}(\eta), \quad \mu_i = g^{-1}(x_i \beta)
\]

Table 2: Some commonly used link functions (Madsen and Thyregod, 2010)

<table>
<thead>
<tr>
<th>Name</th>
<th>Link function: ( \eta = g(\mu) )</th>
<th>( \mu = g^{-1}(\eta) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>( \mu )</td>
<td>( \eta )</td>
</tr>
<tr>
<td>logarithm</td>
<td>( \log(\mu) )</td>
<td>( \exp(\eta) )</td>
</tr>
<tr>
<td>logit</td>
<td>( \log(\mu/(1 - \mu)) )</td>
<td>( \exp(\eta)/(1 + \exp(\eta)) )</td>
</tr>
<tr>
<td>power</td>
<td>( \mu^k )</td>
<td>( \eta^{1/k} )</td>
</tr>
<tr>
<td>squareroot</td>
<td>( \sqrt{\mu} )</td>
<td>( \eta^2 )</td>
</tr>
</tbody>
</table>

2.2.3 Estimation

Estimation of \( \beta \) for GLMs is typically done by use of the maximum likelihood method. The idea is to find an estimate for \( \beta \) which maximizes the likelihood function given the
2. THEORY

observed data \( Y = y \). If \( y_i \), for \( i = 1, \ldots, n \), are iid with the probability density function (pdf) or probability mass function (pmf) \( f(y_i) \), the likelihood function is then

\[
L(\beta) = \prod_{i=1}^{n} f(y_i).
\]

Maximizing \( L(\beta) \) is equivalent to maximizing the logarithm of it. Since the latter case is easier to work with, it is also preferable. Taking the logarithm of \( L(\beta) \), we get the log-likelihood

\[
\ell(\beta) = \log \prod_{i=1}^{n} f(y_i) = \sum_{i=1}^{n} \log \{ f(y_i) \}.
\]

In the case of the exponential dispersion family, we have that

\[
\ell(\beta) = \sum_{i=1}^{n} \log \left\{ f(y_i; \theta_i, \phi) \right\} = \sum_{i=1}^{n} \log \left\{ \exp \left( \frac{y_i \theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi) \right) \right\} = \sum_{i=1}^{n} \frac{y_i \theta_i - b(\theta_i)}{a(\phi)} + \sum_{i=1}^{n} c(y_i, \phi).
\]

To find the maximum of \( L(\beta) \) with respect to \( \beta \), we differentiate \( \ell(\beta) \) with respect to all \( \beta_j \) and solve the equation

\[
\frac{\partial \ell(\beta)}{\partial \beta_j} = 0, \quad j = 0, \ldots, k,
\tag{2.6}
\]

where \( \beta \) is related to \( Y_i \) through \( E[Y_i] = \mu \) and \( \eta_i = g(\mu_i) = x_i \beta \). By the chain rule for differentiation we have that

\[
\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \beta_j} = \sum_{i=1}^{n} \left( \frac{\partial \ell_i}{\partial \theta_i} \frac{\partial \theta_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \eta_i}{\partial \beta_j} \right).
\tag{2.7}
\]

Using \( E[Y_i] = b'(\theta_i) \) and \( var[Y_i] = a(\phi) b''(\theta_i) \) for the exponential dispersion family (see Remark 2.3), we can write the factors on the right hand side of (2.7) as

\[
\frac{\partial \ell_i}{\partial \theta_i} = \frac{y_i - b'(\theta_i)}{a(\phi)} = \frac{y_i - \mu_i}{a(\phi)} \quad \text{since} \quad \frac{\partial \mu_i}{\partial \theta_i} = b''(\theta_i) = \frac{var[Y_i]}{a(\phi)}
\]

\[
\frac{\partial \ell_i}{\partial \mu_i} = \frac{a(\phi)}{var[Y_i]} \quad \frac{\partial \ell_i}{\partial \eta_i} = \frac{a(\phi)}{var[Y_i]} \quad \frac{\partial \ell_i}{\partial \beta_j} = x_{ij}.
\]

Equation (2.6) can now be written as

\[
\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{y_i - \mu_i}{a(\phi)} \frac{a(\phi)}{var[Y_i]} \frac{\partial \mu_i}{\partial \eta_i} x_{ij} = \sum_{i=1}^{n} x_{ij} \frac{\partial \mu_i}{\partial \eta_i} (var[Y_i])^{-1}(y_i - \mu_i) = 0,
\]

12
or

\[ X^tDV^{-1}(y - \mu) = 0 \]

in matrix form, where \( D \) and \( V \) are the diagonal matrices

\[
D = \begin{bmatrix}
\frac{\partial \mu_1}{\partial \eta_1} & 0 \\
0 & \ddots & \frac{\partial \mu_n}{\partial \eta_n}
\end{bmatrix}, \quad V = \begin{bmatrix}
\text{var}[Y_1] & 0 \\
0 & \ddots & \text{var}[Y_n]
\end{bmatrix}.
\]

There is in general no closed form solution for the MLE for GLMs. Several numerical methods for calculating the MLE are available and we will discuss some of them in Section 2.2.4.

### 2.2.4 Iterative algorithms

In this section we will describe two commonly used iterative algorithms for GLMs, the Newton-Raphson algorithm and the Fisher scoring algorithm. The idea of the algorithms in light of GLMs is based on the second order Taylor expansion of the log-likelihood. Let us introduce some useful definitions before we move on to discussing the algorithms.

**Definition 2.3. (Score vector)**

The score vector is defined as a column vector with score functions (the partial derivatives of the log-likelihood) as the entries

\[
S(\beta) = \begin{bmatrix}
S_1(\beta) \\
\vdots \\
S_k(\beta)
\end{bmatrix} = \begin{bmatrix}
\frac{\partial \ell(\beta)}{\partial \beta_1} \\
\vdots \\
\frac{\partial \ell(\beta)}{\partial \beta_k}
\end{bmatrix}.
\]

**Definition 2.4. (Hessian matrix)**

The Hessian matrix is the matrix of the partial derivatives of the score vector

\[
H(\beta) = \begin{bmatrix}
\frac{\partial^2 \ell(\beta)}{\partial \beta_1^2} & \cdots & \frac{\partial^2 \ell(\beta)}{\partial \beta_1 \partial \beta_k} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 \ell(\beta)}{\partial \beta_k \partial \beta_1} & \cdots & \frac{\partial^2 \ell(\beta)}{\partial \beta_k^2}
\end{bmatrix}.
\]

**Definition 2.5. (Fisher information matrix)**

The Fisher information matrix is defined as the negative expected Hessian matrix

\[
I(\beta) = -E[H(\beta)].
\]
Newton-Raphson algorithm
The second order Taylor expansion around the unknown single parameter $\beta$ is given by

$$\ell(\beta) \approx \ell(\beta^0) + \ell'(\beta^0)(\beta - \beta^0) + \frac{1}{2} \ell''(\beta^0)(\beta - \beta^0)^2$$

(2.8)

for a value $\beta^0$ close to $\beta$. Solving the equation $\ell'(\beta) = 0$ for $\beta$ yields the expression

$$\beta = \beta^0 - \frac{\ell'(\beta^0)}{\ell''(\beta^0)}.$$

If we assume $\hat{\beta}^s$ to be the estimate after $s$ iterations, a new estimate $\hat{\beta}^{s+1}$ can be obtained by

$$\hat{\beta}^{s+1} = \hat{\beta}^s - \frac{\ell'(\hat{\beta}^s)}{\ell''(\hat{\beta}^s)}.$$

This method requires a starting point $\beta^0$ and continuing for $s = 1, 2, ..$ until for example when $|\hat{\beta}^{s+1} - \hat{\beta}^s| < \epsilon$ or $|\ell'(\hat{\beta}^s)| < \epsilon$ where $\epsilon$ is a small fixed number. $\hat{\beta}^{s+1} = \hat{\beta}^s$ means that $\ell'(\hat{\beta}^s) = 0$ and we have reached a stationary point. The second derivative test, $\ell''(\beta) < 0$ is often used to make sure that this is a maximum. The initial value is important in situations with several local maximum points. With a bad choice of the initial value the algorithm may converge to the "wrong" solution of $\ell'(\beta) = 0$.

The multi-parameter analogue to (2.8) is

$$\ell(\beta) \approx \ell(\beta^0) + S(\beta^0)(\beta - \beta^0) + \frac{1}{2} (\beta - \beta^0)^t H(\beta^0)(\beta - \beta^0).$$

where $S(\beta^0)$ is the score vector evaluated at $\beta^0$ and $H(\beta^0)$ is the Hessian matrix evaluated at $\beta^0$. A new estimate is now given by

$$\hat{\beta}^{s+1} = \hat{\beta}^s - [H(\hat{\beta}^s)]^{-1} S(\hat{\beta}^s)$$

and the MLE is obtained by Algorithm 1. The condition for a maximum is that the Hessian matrix $H(\hat{\theta})$ is negative definite, which is true if all the eigenvalues are negative.

**Algorithm 1** Newton-Raphson algorithm

**Input:** $\epsilon$, start value $\beta$

1. compute: $S(\beta)$, $H(\beta)$
2. while $S(\beta) > \epsilon$ do
3. \hspace{1cm} $\beta = \beta - [H(\beta)]^{-1} S(\beta)$ \hspace{1cm} % new estimate of $\beta$
4. \hspace{1cm} compute: $S(\beta)$, $H(\beta)$
5. Return $\beta$ \hspace{1cm} % MLE of $\beta$
2. THEORY

**Fisher scoring algorithm**
Replacing the Hessian matrix with the Fisher information matrix from Definition 2.5 yields

\[
\hat{\beta}^{s+1} = \hat{\beta}^s + \left[ I(\hat{\beta}^s) \right]^{-1} S(\hat{\beta}^s).
\]

This is called the Fisher scoring algorithm. The modification will according to Knight (2000) make the algorithm more robust, but when both algorithms converge, the Newton-Rapshon will often converge to the solution more quickly.

2.3 Generalized linear mixed models

GLMs constitute a broad class of statistical models. However, there are situations where the assumption of independent observations is not fulfilled. This section will introduce another class of models which is more appropriate for handling clustered data.

Clustered data arise when the observations can be divided into smaller sub-groups, where observations within each sub-group are more 'alike' than observations across them. In longitudinal data, for instance, we typically have observations obtained from the same subjects over time. Each subject then represents a cluster. It is often reasonable to assume that observations of each subject are independent of the ones from others, but the observations within a cluster are correlated. The within-group correlation contradicts the assumption of independence for GLMs. One way to deal with this deficiency is to use GLMMs.

Let \( y_{ic} \) be the \( c \)-th observation in cluster \( i \) for \( c = 1, \ldots, m \) and \( i = 1, \ldots, n \). With a column vector \( \gamma_i \) of random variables (random effects), a GLMM can be given on the form

\[
g(\mu_{ic}) = x_{ic}\beta + u_{ic}\gamma_i, \quad \mu_{ic} = E[Y_{ic}|\gamma_i],
\]

where \( x_{ic} \) and \( u_{ic} \) (typically a subset of \( x_{ic} \)) are row vectors of covariates and \( \beta \) (fixed effects) is a column vector of parameters. The observations are now assumed to be independent given the random effects, with a conditional probability distribution from the exponential dispersion family. The mean and variance properties in (2.4) and (2.5) still hold for GLMMs, since the conditional probability distribution is a member of the exponential dispersion family. All observations within a cluster are influenced by the same random effects, but the random effects differ across the clusters. The random effects are usually assumed to be normally distributed with mean zero and unknown variance, but other distributions can be used. Lee and Nelder (1996) referred the models with non-normal \( \gamma_i \) as hierarchical generalized linear models. GLMMs can be used in various fields, e.g. medical studies (Breslow and Clayton, 1993) and biological studies (Bolker et al., 2009). We will in Section 3.3 illustrate the use of GLMMs in the context of non-life insurance.
2. THEORY

Pure random effect model
A GLMM without random effects is just a GLM. If we instead have a model where all
effects are assumed to be random, we have the pure random effect model

\[ g(\mu_{ic}) = u_{ic}\gamma_i. \]

Such models are useful when observations within clusters are correlated, and only the
variation between groups is of interest.

Random intercept model
The simplest form of a mixed model is the random intercept model, where \( u_{ic} \) is a vector
of ones’. The model can be expressed by

\[ g(\mu_{ic}) = x_{ic}\beta + \gamma_i. \]

The model is named random intercept because the random effect allow the intercept term
to vary among clusters, i.e. each cluster has it own intercept term. Rewriting (2.10) to

\[ g(\mu_{ic}) = \gamma_i + \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k, \]

we have the intercept term \( \beta_0 + \gamma_i \) for subject \( i \).

2.3.1 Estimation
Estimating parameters in GLMMs usually involves maximum likelihood. In general, a
closed analytical solution for maximizing the likelihood function does not exists. Sev-
eral approximation methods are proposed to estimate the model parameters. Breslow
and Clayton (1993) illustrated the use of penalized quasi-likelihood (PQL), Zhao et al.
(2006) demonstrate how the Markov chain Monte Carlo (MCMC) method can be used,
Pinheiro and Chao (2006) noted that the PQL method can often produce biased esti-
mates and prefer the Laplacian and adaptive Gaussian quadrature approximation. All
methods are available in different \texttt{R} packages. The PQL method is implemented in the
glmmPQL function from the \texttt{MASS} package, the MCMC method in the \texttt{MCMCglmm}
function from the \texttt{MCMCglmm} package, the Laplace approximation and adaptive Gauss
Hermite quadrature (AGHQ), a type of adaptive Gaussian quadrature, are available in
the \texttt{glmer} function from the \texttt{lme4} package, which is the one we utilize in our modeling
part. At present, the AGHQ is only implemented for models with a single scalar random
effect. So for models with several random effects, we used the Laplace approximation.
For models with a single random effect, the AGHQ was preferred. Descriptions of both
approximation methods will be given in Section 2.3.2.

With the observation \( y \) of the vector \( Y = [Y_1, \ldots, Y_n]^t \) where \( Y_i = [Y_{i1}, \ldots, Y_{im}]^t \), and
the random effects \( \gamma = [\gamma_1, \ldots, \gamma_n]^t \), we have the conditional density (or mass) function

\[ f(y|\gamma; \beta) = \prod_{i=1}^n \prod_{c=1}^m f(y_{ic}|\gamma_i; \beta). \]
2. THEORY

The joint likelihood function of GLMMs is given in terms of \( f(y|\gamma; \beta) \) and \( f(\gamma; D) \), i.e.

\[
L(\beta, D; \gamma, y) = f(y|\gamma; \beta) f(\gamma; D), \quad \gamma \sim N(0, D), \quad \gamma_i \sim N(0, d)
\]

where \( D \) is a block diagonal matrix with entries \( d \), and \( d \) the variance-covariance matrix for the random effects \( \gamma_i \). To obtain the marginal likelihood for \((\beta, D)\), the random effects are integrated out,

\[
L(\beta, D; y) = \int_{\gamma} L(\beta, D; \gamma, y) \, d\gamma
\]

\[
= \int_{\gamma} f(y|\gamma; \beta) f(\gamma; D) \, d\gamma
\]

\[
= \int_{\gamma} \prod_{i=1}^{n} \prod_{c=1}^{m} f(y_{ic}|\gamma_i; \beta) f(\gamma_i; d) \, d\gamma_i
\]

\[
= \prod_{i=1}^{n} \int_{\gamma} \prod_{c=1}^{m} f(y_{ic}|\gamma_i; \beta) f(\gamma_i; d) \, d\gamma_i
\]

\[
= \prod_{i=1}^{n} \int_{\gamma} \prod_{c=1}^{m} \left( \frac{y_{ij} \theta_{ij} - b(\theta_{ij})}{a(\phi)} + c(y_{ij}, \phi) \right) f(\gamma_i; d) \, d\gamma_i,
\]

where the dimension of the multiple integral is equal to the dimension of \( \gamma \). In general there is no closed analytical solution for maximizing \( L(\beta, D; y) \). We will briefly discuss the two approaches for approximating the integral in (2.11) that are available in the \textit{lme4} package.

2.3.2 Approximation methods

Laplace approximation

The method approximates integrals on the form

\[
\int \exp \{ r(t) \} \, dt.
\]

The following outline is derived from Madsen and Thyregod (2010). Let \( \Psi = (\beta, D) \) be the model parameters to be estimated. The log-likelihood function

\[
\ell(\Psi; \gamma, y) = \log \{ L(\Psi; \gamma, y) \}
\]

approximated by a second-order Taylor expansion around the optimum \( \hat{\gamma} = \hat{\gamma}_\Psi \) of the log-likelihood function with respect to the unobserved random effects \( \gamma \), yields

\[
\ell(\Psi; \gamma, y) \approx \ell(\Psi; \hat{\gamma}; y) - \frac{1}{2}(\gamma - \hat{\gamma})^T h(\hat{\gamma})(\gamma - \hat{\gamma}).
\]

Here, \( h(\hat{\gamma}) = -H(\hat{\gamma}) \) is the negative Hessian matrix evaluated at \( \hat{\gamma} \). The first-order term disappears since \( \hat{\gamma} \) is the solution to

\[
\partial \ell(\Psi; \hat{\gamma}; y)/\partial \gamma = 0.
\]
2. THEORY

Using that
\[ \ell(\Psi; \gamma, y) = \log \{ \exp (\ell(\Psi; \gamma, y)) \} \]
combined with (2.12), the marginal likelihood function becomes
\[
\ell(\Psi; y) = \int_\gamma \ell(\Psi; \gamma, y) \, d\gamma \\
= \log \int_\gamma \exp \{ \ell(\Psi; \gamma, y) \} \, d\gamma \\
\approx \log \int_\gamma \exp \left\{ \ell(\Psi; \gamma, y) - \frac{1}{2} (\gamma - \hat{\gamma})^t \mathbf{h}(\hat{\gamma}) (\gamma - \hat{\gamma}) \right\} \, d\gamma \\
= \ell(\Psi; \hat{\gamma}, y) + \log \int_\gamma \exp \left\{ -\frac{1}{2} (\gamma - \hat{\gamma})^t \mathbf{h}(\hat{\gamma}) (\gamma - \hat{\gamma}) \right\} \, d\gamma \\
= \ell(\Psi; \hat{\gamma}, y) + \log \left\{ \frac{(2\pi)^q}{|\mathbf{h}(\hat{\gamma})|} \times \int_\gamma \exp \left( -\frac{1}{2} (\gamma - \hat{\gamma})^t \mathbf{h}(\hat{\gamma}) (\gamma - \hat{\gamma}) \right) \, d\gamma \right\} \\
= \ell(\Psi; \hat{\gamma}, y) + \log \left( \frac{(2\pi)^{\frac{q}{2}}}{|\mathbf{h}(\hat{\gamma})|} \right) \\
= \ell(\Psi; \hat{\gamma}, y) - \frac{1}{2} \log \left( \frac{|\mathbf{h}(\hat{\gamma})|}{(2\pi)^q} \right),
\]
where \( q \) is the number of random effects. The last integral is eliminated since we are integrating over a multivariate density with mean \( \hat{\gamma} \) and variance-covariance \( \mathbf{h}^{-1}(\hat{\gamma}) \).

Adaptive Gauss-Hermite quadrature

The non-adaptive Gauss-Hermite quadrature (GHQ) method is based on Gaussian quadrature rules which approximate integrals of the form
\[
\int r(t) \, dt = \int f(t) v(t) \, dt
\]
by a finite weighted sum. Different quadrature methods perform better for different types of integrands. In the case \( v(t) = \exp(-t^2) \), the quadrature rule is referred to as the GHQ. The form is popular because of its relation to the Gaussian densities. The integral approximation is then
\[
\int f(t) \exp(-t^2) \, dt \approx \sum_{i=1}^{Q} w_i f(t_i), \quad (2.13)
\]
where \( Q \) is the number of quadrature points, the nodes (or quadrature points) \( t_i \) are the zeroes of the \( Q \)-th order Hermite polynomial and \( w_i \) are suitable corresponding weights (Liu and Pierce, 1994). A more detailed description of the Hermite polynomial can be found in e.g. Greenwood and Miller (1948). It turns out that for specified quadrature points and weights, the formula (2.13) is exact if \( f(t) \) is a polynomial of degree \( 2Q - 1 \).
or less. The quadrature points are symmetric and centered around zero for the GHQ. Now, assume that we have a smooth function $g(x)$ and that $g(x) > 0$. The goal is to approximate the integral $\int g(x)dx$ by applying the GHQ. Whenever $g(x)$ is not centered around zero, many quadrature points may fall outside the region where the variation in $g(x)$ is concentrated. A solution for this problem is to use the AGHQ. The method transforms the quadrature points, centring and spreading them around the mode of the integrand. This has been shown to dramatically reduce the number of quadrature points needed to approximate integrals effectively. Assume the integral

$$\int_{-\infty}^{\infty} f(x) \phi(x; \mu, \sigma) dx,$$

where $\phi(x; \mu, \sigma)$ is an arbitrary Gaussian density. A variable transformation

$$u = \frac{x - \mu}{\sqrt{2\sigma^2}} \iff x = \sqrt{2\sigma^2}u + \mu, \quad dx = \sqrt{2\sigma^2}du$$

gives the form

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} f(\sqrt{2\sigma^2}u + \mu) \exp \left( -u^2 \right) du.$$

Hence, by (2.13) we have the approximation

$$\int_{-\infty}^{\infty} f(x) \phi(x; \mu, \sigma) dx \approx \sum_{i=1}^{Q} \frac{w_i}{\sqrt{\pi}} f(\sqrt{2\sigma^2}u_i + \mu),$$

where $w_i/\sqrt{\pi}$ are modified weights and $x_i = \sqrt{2\sigma^2}u_i + \mu$ the sampling nodes. $\mu$ and $\sigma$ are then chosen such that the integrand $g(x)$ is sampled in a suitable range. In particular, we let

$$\hat{\mu} = \text{mode} \left[ f(x) \phi(x; \mu, \sigma) \right] \quad \text{and} \quad \hat{\sigma}^2 = \left[ -\frac{\partial^2}{\partial x^2} \log f(x) \phi(x; \mu, \sigma) \bigg|_{x=\hat{\mu}} \right]^{-1}.$$

If we now define

$$h(x) = \frac{g(x)}{\phi(x; \hat{\mu}, \hat{\sigma})}$$

we can write

$$\int_{-\infty}^{\infty} g(x)dx = \int_{-\infty}^{\infty} h(x) \phi(x; \hat{\mu}, \hat{\sigma}) dx.$$

Replacing the integrand $f(x) \phi(x; \mu, \sigma)$ in (2.14) with $h(x) \phi(x; \hat{\mu}, \hat{\sigma})$ and applying the GHQ from (2.13), we have that

$$\int_{-\infty}^{\infty} g(x)dx \approx \sum_{i=1}^{Q} \frac{w_i}{\sqrt{\pi}} h(\sqrt{2\hat{\sigma}^2}u_i + \hat{\mu})$$

$$= \sqrt{2\hat{\sigma}^2} \sum_{i=1}^{Q} w_i^* g(\sqrt{2\hat{\sigma}^2}u_i + \hat{\mu}),$$

19
2. THEORY

where \( w_i^* = w_i \exp(x_i^2) \). Recall from (2.11) that the marginal likelihood for GLMMs is given by

\[
L(\beta, D; y) = \int_\gamma f(y|\gamma; \beta) f(\gamma; D) \; d\gamma.
\]

Since the random effects \( \gamma \) are assumed to be normally distributed, AGHQ is a commonly used approximation. The accuracy is high when \( f(y|\gamma; \beta) \) can be well approximated by a polynomial.
3. Models for insurance data

In Chapter 2 we introduced different classes of linear models. The LM is broadly used in applied statistic. However, the normality assumption makes the model unsuitable for insurance modeling. Claim counts follow a discrete distribution, are non-negative and typically right-skewed. Claim sizes on the other hand, follow a continuous distribution, are positive, and like claim counts, often right-skewed. GLMs are more appropriate for insurance data since they constitute a broader class and can handle the mentioned properties. GLMMs allow random effects which take the within-group correlation structure into account and may provide a better choice for clustered data. This chapter specifies the chosen distributions and the regression models used in our modeling part.

3.1 Generalized linear models for claim frequency

Poisson

A fundamental distribution for modeling count data is the Poisson. De Jong et al. (2008) and Ohlsson and Johansson (2010) for instance, consider that number of claims in non-life insurance follows this distribution. This motivates the use of the Poisson distribution for claim frequency in our simulation. A discrete random variable $Y$ is Poisson distributed if the pmf is given by

$$f(y; \lambda) = \frac{\lambda^y}{y!} e^{-\lambda}, \quad y = 0, 1, \ldots$$

with the mean and variance $E[Y] = var[Y] = \lambda$. In short, we use the expression $Y \sim \text{Poisson}(\lambda)$.

The claim count is the number of claims that occur during a policy period. The claim frequency is obtained by dividing the claim counts by the exposure. In our case, the exposure is the duration of the policy. It is often more appropriate to use claim frequency for modeling, since the duration can vary for different policies. With a log-link, the Poisson GLM becomes

$$\log \left( \frac{\lambda_i}{t_i} \right) = x_i \beta \quad \iff \quad \lambda_i = t_i \cdot \exp(x_i \beta), \quad Y_i \sim \text{Poisson}(\lambda_i)$$

where $\lambda_i = E[Y_i | x_i]$ and the term $t_i$ is known as an offset.

Negative binomial

In addition to the Poisson, the negative binomial is utilized for the claim frequency. The negative binomial has also been used by e.g. De Jong et al. (2008) and Boucher et al. (2008) for insurance modeling. The Poisson model requires the mean to be equal to the variance, which is not satisfied for many datasets of interest. When the variance is greater than the mean, the data are said to be overdispersed. In the opposite case, they are said to be underdispersed. The negative binomial distribution has two parameters, and hence, it is more flexible for fitting data compared to the Poisson. It can therefore be a better
3. MODELS FOR INSURANCE DATA

alternative when data are over- or underdispersed.

Assume that $Y|\lambda \sim \text{Poisson}(\lambda)$ and $\lambda$ is gamma distributed with mean $\mu$ and a shape parameter $k$. It can be shown that $Y$ follows the negative binomial distribution

\[
f(y; \mu, \tau) = \frac{\Gamma(y + 1/\tau)}{\Gamma(1/\tau)\Gamma(y + 1)} \left( \frac{\mu}{\mu + 1/\tau} \right)^{\frac{1}{\tau}} \left( \frac{1/\tau}{\mu + 1/\tau} \right)^y, \quad y = 0, 1, ...
\]

where $\tau = 1/k$ is called a dispersion parameter. The mean and variance of $Y$ are given by $E[Y] = \mu$ and $\text{var}[Y] = \mu + \tau \mu^2$. The pmf can be expressed in slightly different ways. Here, we use the same as Agresti (2015). Greater $\tau$ means greater overdispersion relative to the Poisson. Also note that when $\tau \to 0$, the distribution of $Y$ converges to the Poisson. In short, we use the notation $Y \sim \text{NB}(\mu, \tau)$. It is also known as a Poisson-gamma mixture distribution.

As for the Poisson GLM, the log-link is used for the negative binomial GLM. The model becomes

\[
\mu_i = E[Y_i|x_i] = t_i \cdot \exp(x_i \beta), \quad Y_i \sim \text{NB}(\mu_i, \tau).
\]

In the modeling part, we simplify the models by taking the exposure $t_i$ to be 1 for all policies.

3.2 Generalized linear models for claim sizes

**Gamma**

The claim size is simply the cost associated with each claim that occurred. Both De Jong et al. (2008), and Ohlsson and Johansson (2010) use the gamma distribution for claim sizes in their insurance applications. It will also be utilized in our simulation. A continuous random variable $Y$ follows a gamma distribution with a mean parameter $\mu$ and shape parameter $k$ if the pdf is of the form

\[
f(y; \mu, k) = \frac{(k/\mu)^k}{\Gamma(k)} e^{-k y/\mu} y^{k-1}, \quad y > 0
\]

or $Y \sim \text{gamma}(\mu, k)$ in short hand notation. The distribution has the mean $\mu$ and variance $\mu^2/k$. There are other parameterizations, but we use the same as Agresti (2015).

With a log-link, the gamma GLM becomes

\[
\mu_i = E[Y_i|x_i] = \exp(x_i \beta), \quad Y_i \sim \text{gamma}(\mu_i, k).
\]

**Inverse Gaussian GLM**

Another distribution we use for claim sizes is the inverse Gaussian. Similar to the gamma, it is a right-skewed distribution but is heavier tailed. De Jong et al. (2008) mention that the inverse Gaussian is a good alternative in situations of extreme skewness, and illustrate
the use of the distribution in some examples. A continuous random variable \( Y \) follows
the inverse Gaussian distribution if the pdf can be given by

\[
f(y; \mu, \lambda_{IG}) = \left( \frac{\lambda_{IG}}{2\pi y^3} \right)^{1/2} \exp \left\{ -\frac{\lambda_{IG}(y - \mu)^2}{2\mu^2 y} \right\}, \quad y > 0.
\]

The mean and variance of the distribution are \( \mu \) and \( \mu^3/\lambda_{IG} \), respectively, and \( \lambda_{IG} \) is the scale (also called precision) parameter of the distribution. For any random variable \( Y \) with the pdf given above, we use the short hand notation \( Y \sim IG(\mu, \lambda_{IG}) \).

As for the gamma GLM, we choose to use a log-link. The inverse Gaussian can hence be
expressed by

\[
\mu_i = E[Y_i|x_i] = \exp(x_i\beta), \quad Y_i \sim IG(\mu_i, \lambda_{IG}).
\]

### 3.3 Generalized linear mixed models for insurance data

For longitudinal data where insurance policies are followed over time, it is natural to
assume that the claims from the same policyholder are somehow related. Some of the
policyholders have a higher claim frequency than the average. This can be caused by
individual behavior characteristics such as driving skills, swiftness of reflexes, power of
concentration, aggressiveness behind the wheel, etc. Such characteristics are not captured
by the fixed effects in the regression models. Random effects are cluster-specific parameters and are used to resolve cluster-dependency.

Recall the random intercept model from (2.10). With a log-link, the model is expressed
by

\[
\mu_{ic} = \exp(x_i\beta + \gamma_i).
\]

Regarding our dataset, \( i = 1, \ldots, n \) denotes the insured over \( c = 1, \ldots, m \) years. The random
effect \( \gamma_i \) is an individual heterogeneity term that is constant over time. We will, in the
modeling part of the thesis, compare GLMs and random intercept models for simulated
datasets. We will evaluate whether the benefit is worth introducing a more complicated
model and if it is preferable for predicting future claims. Although the random intercept
model may be suitable for our research, it is not inconceivable that a more general case
of GLMMs from (2.9) provides a better model. Therefore we will also look into GLMMs
that allow several random effects and see if they provide even more accurate results than
GLMs.
4 Simulation study

Goal of simulations
Our purpose is to see how well insurance data can be estimated by GLMs and GLMMs, both when affected by random effects and not, in terms of the premium. Especially, we wish to get an insight into how much estimates of the pure premium are affected when using GLMs in the case of random effects. Another interesting part is to compare the models in Table 3 and see how well they fit our data. Since the data are drawn from specific models, the true model should give the estimate that is closest to the true value. Use of "wrong" models may provide bad predictions for future claims. An important question to ask is whether the estimates of the pure premium can be good even when we are using the "wrong" models. To answer this question, we simulate datasets that reflect insurance data in reality, but with some simplifications. The claim counts and sizes are then fitted by different models for comparison. An advantage of these simulations is that the true premium is known. It enables us to see how much the estimates deviate from the true value.

In the first part of the study, we simulated datasets based on the models from Table 3. For the mixed models, we let only the intercept term be affected by a random effect. In the second part, we simulated data from a more general form of mixed model. The goal of the second part is to see if GLMMs give noticeably better results when the true models are affected by several random effects.

Software and packages
All simulations in the study are done with the R software (R Core Team, 2017). Estimating parameters for GLMs is done by use of the glm function from the stats package and the glm.nb function from the mass package (Venables and Ripley, 2002). For GLMMs, the functions glmer and glmer.nb from the lme4 package (Bates et al., 2015) are used. The function glmer.nb uses the function theta.ml from the mass package to estimate the initial value of the negative binomial parameter theta, which is required. Several R packages contain functions that fit GLMMs, but few have all of Poisson, negative binomial, gamma and inverse Gaussian GLMM included. Hence, the lme4 package was chosen.

Models
The Poisson model is widely used to fit claim counts in non-life insurance, but for overdispersed data the negative binomial model may provide better fit. The most common choices for claim sizes are the gamma or the inverse Gaussian model. Different combinations of those models for claim counts and sizes, either with or without random effects, give us the 8 models in Table 3, which are the ones we are using in this research. For the case of random effects, both models for claim counts and sizes include random effects.
4. SIMULATION STUDY

Table 3: Models for the insurance data

<table>
<thead>
<tr>
<th>#</th>
<th>models for claim counts</th>
<th>models for sizes</th>
<th>random effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Poisson</td>
<td>gamma</td>
<td>no</td>
</tr>
<tr>
<td>(2)</td>
<td>Poisson</td>
<td>inverse Gaussian</td>
<td>no</td>
</tr>
<tr>
<td>(3)</td>
<td>negative binomial</td>
<td>gamma</td>
<td>no</td>
</tr>
<tr>
<td>(4)</td>
<td>negative binomial</td>
<td>inverse Gaussian</td>
<td>no</td>
</tr>
<tr>
<td>(5)</td>
<td>Poisson</td>
<td>gamma</td>
<td>yes</td>
</tr>
<tr>
<td>(6)</td>
<td>Poisson</td>
<td>inverse Gaussian</td>
<td>yes</td>
</tr>
<tr>
<td>(7)</td>
<td>negative binomial</td>
<td>gamma</td>
<td>yes</td>
</tr>
<tr>
<td>(8)</td>
<td>negative binomial</td>
<td>inverse Gaussian</td>
<td>yes</td>
</tr>
</tbody>
</table>

Dataset
Each dataset consists of simulated claim counts, sizes and a design matrix. The first column in the design matrix is a vector of ones corresponding to the intercept term in our model. Columns 2 to 11 are covariates in the regression models. The design matrix is simulated by Algorithm 2. Claim counts and sizes are simulated from the models given in Table 3.

Algorithm 2 Design matrix

Input: $n$ (number of policyholders), $m$ (number of years)

1: $X_0 = (1, 1, \ldots, 1)^t$ \hspace{1cm} \% vector of length $n \times m$
2: Draw $X_1, \ldots, X_7 \sim N(1, 0.04)$ \hspace{1cm} \% vector of length $n \times m$
3: Draw $X_8 \sim binom(0.5)$ \hspace{1cm} \% vector of length $n \times m$
4: Draw $X_9 \sim binom(0.5)$ \hspace{1cm} \% vector of length $n$
5: Draw $X_{10} \sim multinom(0.2, 0.3, 0.5)$ \hspace{1cm} \% vector of length $n \times m$
6: Draw $X_{11} \sim multinom(0.2, 0.05, 0.35, 0.1, 0.3)$ \hspace{1cm} \% vector of length $n$
7: Repeat $X_9$ and $X_{11}$ $m$ times \hspace{1cm} \% remain the same each year
8: \textbf{Return} design matrix $X = (X_0, \ldots, X_{11})$ \hspace{1cm} \% $n \times m$ rows and 12 columns

For real insurance data, covariates can be either continuous or discrete variables. Examples of continuous variables are weight of vehicle, price of vehicle, horse power, driving length and age of driver. In practice, it is common to transform some of the continuous variables into categorical (discrete) variables to characterize a non-linear effect of the covariates. Take for example the age of the driver. When drivers become more experienced, the accident rate goes down. At the same time, the rate will typically go up again for very old drivers. Age can then be divided into several groups, where each group is related to the response with its own coefficient $\beta$. Examples of discrete variables are gender, nationality and number of children. Some of the variables often remain the same for the policyholder over years, e.g. gender, geographical area and so on. To get synthetic data that resemble real data, we let some of the covariates be continuous and some categorical. We also let $X_9$ (could be gender) and $X_{11}$ (could be geographic area) remain the same
4. SIMULATION STUDY

over all years for the policyholders. The exposure, i.e. the duration of the policy for the response variables, is taken to be 1 in the simulations. Turning the categorical covariates into \( p - 1 \) vectors with values 0 or 1 where \( p \) is the number of categories, we have a design matrix with 16 columns.

**Model effects**
Simulations are based on 3 different model forms

(i)  \( \mu = \exp(x\beta) \)

(ii) \( \mu = \exp(x\beta + \gamma) \)

(iii) \( \mu = \exp(x\beta + u\gamma) \)

The first one is a model without random effects, the second has a random intercept term and the third is a more general case of a mixed model with several random effects. Adding more random effects to the model increases the complexity and requires more computation time. For this research, we settle with two random variables for the general case of the mixed model. We let \( u = [u_0, u_1] = [1, x_1] \) be a subset of \( x \) and \( \gamma = [\gamma_0, \gamma_1]^t \) a column vector with two random normal variables.

In order to simulate claim counts and sizes, we need the fixed effects \( \beta^N \) and \( \beta^Z \), and the random effects \( \gamma^N = [\gamma_0^N, \gamma_1^N]^t \) and \( \gamma^Z = [\gamma_0^Z, \gamma_1^Z]^t \). They are also needed when calculating the true premium. \( \beta^N \) and \( \gamma^N \) apply to the models for claim counts, \( \beta^Z \) and \( \gamma^Z \) apply to the models for claim sizes. Since there are 16 columns in our design matrix, \( \beta^N \) and \( \beta^Z \) have to be vectors of length 16. The fixed parameters are chosen randomly and shown in Table 4.

Recall that GLMs with a log-link are on the following form

\[
\mu = \exp (\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k) = \exp(\beta_0) \cdot \exp(\beta_1 x_1) \cdot \ldots \cdot \exp(\beta_k x_k).
\]

Large values of \( \beta_i \) can increase \( \mu \) dramatically, so we choose "reasonable" values for \( \beta_0, \ldots, \beta_{15} \) such that none of them have too much effect on \( \mu \).

**Table 4:** Fixed effects for the model

| \( \beta^N \) | \( [-2.26, 0.32, 0.34, 0.24, 0.71, -0.52, 0.22, -0.13, 0.12, -0.13, 0.49, 0.61, -0.42, -0.62, 1.20, 0.21]^t \) |
| \( \beta^Z \) | \( [-2.32, 0.31, -0.19, 0.27, 0.31, -0.22, 0.43, 0.23, 0.28, 0.26, 0.23, 0.31, -0.21, 0.12, 0.64, 0.22]^t \) |
The random effects are generated independently from a normal distribution
\[ \gamma_0^N \sim N(0, v_1^2), \quad \gamma_1^N \sim N(0, v_2^2), \quad \gamma_0^Z \sim N(0, v_3^2), \quad \gamma_1^Z \sim N(0, v_4^2) \]
and are hence uncorrelated. The parameters \( v_1, v_2, v_3, v_4 \) are chosen standard deviations for the random effects. To see how much the estimates of the pure premium are affected by random effects, we let the values for \( v_1, v_2, v_3 \) and \( v_4 \) vary between 0 (for fixed effects only), 0.1, 0.2 and 0.4.

**Claim counts and claim sizes**

The mixed model in our simulations takes the form
\[ \lambda = \exp \left( \beta_0^N + \beta_1^N X_1 + \ldots + \beta_{15}^N X_{15} + \gamma_0^N + u_1 \gamma_1^N \right), \quad \lambda = E[N|x, \gamma^N] \quad (4.1) \]
for the claim count \( N \), and
\[ \mu = \exp \left( \beta_0^Z + \beta_1^Z X_1 + \ldots + \beta_{15}^Z X_{15} + \gamma_0^Z + u_1 \gamma_1^Z \right), \quad \mu = E[Z|x, \gamma^Z] \quad (4.2) \]
for the claim size \( Z \). Based on \( \lambda \) and \( \mu \), we simulated claim counts and sizes by Algorithm 3. The models that were used to simulate the data are referred to as the true models.

---

**Algorithm 3 Generating claim counts and sizes**

**Input:** \( n, m, \beta^N \) and \( \beta^Z \) from Table 4, \( v_1, v_2, v_3, v_4 \)

1: Draw \( X \) % Algorithm 2
2: if model without random effects then % model (1),(2),(3) or (4)
3: \( \lambda = \exp(X \beta^N) \)
4: \( \mu = \exp(X \beta^Z) \)
5: if mixed model then % model (5),(6),(7) or (8)
6: Draw \( \gamma_0^N = N(0, v_1^2), \gamma_1^N = N(0, v_2^2) \) % vectors of length \( n \)
7: Draw \( \gamma_0^Z = N(0, v_3^2), \gamma_1^Z = N(0, v_4^2) \) % vectors of length \( n \)
8: Repeat \( \gamma_0^N, \gamma_1^N, \gamma_0^Z, \gamma_1^Z \) in times % remain the same each year
9: Set \( U_0 = X_0, \quad U_1 = X_1 \) % \( U \) subset of \( X \)
10: \( \lambda = \exp(X \beta^N + U_0 \gamma_0^N + U_1 \gamma_1^N) \) % follows from (4.1)
11: \( \mu = \exp(X \beta^Z + U_0 \gamma_0^Z + U_1 \gamma_1^Z) \) % follows fom (4.2)
12: Draw \( N = (N_1, \ldots, N_{n \times m}) \) % \( Poisson(\lambda) \) or \( NB(\lambda, \tau) \)
13: for \( (i \text{ in } n \times m) \) do
14: if \( N_i > 0 \) then
15: Draw \( Z_{N_i} = (Z_1, \ldots, Z_{N_i}) \) % \( gamma(\lambda, k) \) or \( IG(\lambda_{IG}) \)
16: Return \( N, Z \) % simulated claim counts and sizes

For the random intercept model, \( \gamma_1^N \) and \( \gamma_1^Z \) are set equal to 0. The algorithm returns one set with claim counts and one with claim sizes. This is done for all combinations in Table 3.
4. SIMULATION STUDY

**True premium**

Each insurance policy constitutes a possibility that the insurer has to pay a compensation to the insured when covered damages occur during the policy period. The expected payout is represented by the pure premium. Since the math in the calculations for the premium is similar for all policyholders, we simplify the pure premium $\Pi_{pu}$ (conditional expected payout for policy $i$ in cluster $c$) with $\Pi_{pu}$ throughout this chapter to avoid cumbersome notation. The pure premium for a policyholder is given by $\Pi_{pu} = E[N|x] \cdot E[Z|x]$, where $E[N|x]$ is the conditional expected number of claims during the policy period and $E[Z|x]$ the conditional expected cost per claim during the same period. This expression comes directly from the calculation

$$
\Pi_{pu} = \mathbb{E} \left[ \sum_{i=1}^{N} Z_i | x \right] = \mathbb{E} \left\{ \mathbb{E} \left[ \sum_{i=1}^{N} Z_i | N, x \right] | x \right\} \overset{\text{indep.}}{=} \mathbb{E} \left\{ \sum_{i=1}^{N} E[Z_i | x] | x \right\} = E[N] \cdot E[Z|x].
$$

(4.3)

To determine the true premium, we need $E[N|x]$ and $E[Z|x]$. They are calculated separately. For models without random effects, we have that

$$
E[N|x] = \exp(x \beta^N) \quad \text{and} \quad E[Z|x] = \exp(x \beta^Z)
$$

(4.4)

For mixed models with $\lambda$ and $\mu$ from (4.1) and (4.2) we have that

$$
E[N|x] = E[\lambda|x] = \mathbb{E} \left[ \exp(x \beta^N + \gamma_0^N + u_1 \gamma_1^N) | x \right] \\
= \exp(x \beta^N) \cdot \mathbb{E} \left[ \exp(\gamma_0^N + u_1 \gamma_1^N) | x \right] \\
= \exp(x \beta^N) \cdot \exp(v_1^2/2) \cdot \exp\left\{\left(u_1^2 v_1^2\right)/2\right\}
$$

(4.5)

and

$$
E[Z|x] = E[\mu|x] = \mathbb{E} \left[ \exp(x \beta^Z + \gamma_0^Z + u_1 \gamma_1^Z) | x \right] \\
= \exp(x \beta^Z) \cdot \mathbb{E} \left[ \exp(\gamma_0^Z + u_1 \gamma_1^Z) | x \right] \\
= \exp(x \beta^Z) \cdot \exp(v_2^2/2) \cdot \exp\left\{\left(u_1^2 v_2^2\right)/2\right\}
$$

(4.6)

With known parameters $\beta^N, \beta^Z, v_1, v_2, v_3, v_4$ and the covariates $x$ applied to each policy simulated by Algorithm 2, the true premium can be computed by (4.3). The last exponential factors in (4.5) and (4.6) come from the moment generating function, $E[\exp(tv)] = \exp\left\{(t^2v^2)/2\right\}$ of a random normal variable with mean zero and standard deviation $v$. Algorithm 4 is used to calculate the true premium. For the random intercept model, $v_2$ and $v_4$ are equal to zero.
## 4. SIMULATION STUDY

**Algorithm 4** True pure premium

**Input:** $\beta^N$ and $\beta^Z$ from Table 4, $v_1, v_2, v_3, v_4, n, m$

1. Draw $X$  
2. if model without random effects then  
   3. $\lambda_{\text{true}} = \exp(X\beta^N)$  
   4. $\mu_{\text{true}} = \exp(X\beta^Z)$  
   5. if mixed model then  
      6. $\lambda_{\text{true}} = \exp\{X\beta^N + v_2^2/2 + (U_1^2 v_4^2)/2\}$  
      7. $\mu_{\text{true}} = \exp\{X\beta^Z + v_3^2/2 + (U_1^2 v_4^2)/2\}$  
   8. $\Pi_{\text{true}}^{pu} = \lambda_{\text{true}} \times \mu_{\text{true}}$  
9. Return $\Pi_{\text{true}}^{pu}$  

**Algorithm 5** Estimated pure premiums

**Input:** $n, m, \beta^N$ and $\beta^Z$ from Table 4, $v_1, v_2, v_3, v_4$

1. Generate $N, Z$  
2. fit $N, Z$  
3. if fitted with GLM then  
   4. $\hat{\lambda} = \exp(X\hat{\beta}^N), \quad \hat{\mu} = \exp(X\hat{\beta}^Z)$  
5. if fitted with GLMM then  
   6. $\lambda = \exp\{X\beta^N + \hat{v}_1^2/2 + U_1^2 \hat{v}_3^2/2\}$  
   7. $\mu = \exp\{X\beta^Z + \hat{v}_3^2/2 + U_1^2 \hat{v}_4^2/2\}$  
8. $\bar{\Pi} = \hat{\lambda} \times \hat{\mu}^{pu}$  
9. Return $\Pi^{pu}$

---

**Estimated premium**

In order to estimate the premium, we need estimates of the parameters $\beta^N, \beta^Z, v_1, v_2, v_3$ and $v_4$. Both GLMs and GLMMs are fitted by the maximum likelihood method in R. In Section 2.2.3 and 2.3.1 we mentioned that there is, in general, no closed form solution for the MLE for GLMs and GLMMs so numerical methods are required. The default method for the functions glm and glm.nb is the Fisher scoring algorithm. The functions glmer and glmer.nb use the Laplace approximation as default. Another option for glmer and glmer.nb is to use the AGHQ approximation method by setting the quadrature points $nAGQ$ as an argument. Setting $nAGQ = 1$ corresponds to the Laplace approximation (Liu and Pierce, 1994). More quadrature points provide higher accuracy, but demands more computations and are much more time-consuming. By running the models a few times, we notice that five or more quadrature points gave relatively similar estimate for our datasets. With a large number of simulations, we chose to use $nAGQ=5$ for the glmer function and the default method for the glm function. The estimated pure premium is simulated by Algorithm 5.
4. SIMULATION STUDY

The algorithm returns the estimate from one fitted model. By the combinations in Table 3 we end up with eight different estimates for each true model. When data are generated from a random intercept model, we do not have estimates for \( \hat{\nu}_2 \) and \( \hat{\nu}_4 \).

Stochastic simulations should always be run enough times to provide a stable pattern. With expensive simulation (much computer time for each run) we chose to run the simulation (only) 500 times. A list of the parameter specifications is given in Table 5.

**Table 5: Parameter specifications**

<table>
<thead>
<tr>
<th>Simulation runs</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policyholders</td>
<td>( n = 1000 )</td>
</tr>
<tr>
<td>Years</td>
<td>( m = 5 )</td>
</tr>
<tr>
<td>Models</td>
<td>given in Table 3</td>
</tr>
<tr>
<td>( Y \sim \text{Poisson}(\lambda) )</td>
<td>( \lambda ) is a rate parameter</td>
</tr>
<tr>
<td>( Y \sim \text{NB}(\lambda, \tau) )</td>
<td>( \lambda ) is a mean and ( \tau = 0.2 ) is a dispersion parameter</td>
</tr>
<tr>
<td>( Y \sim \text{G}(\mu, k) )</td>
<td>( \mu ) is a mean and ( k = 6 ) is a shape parameter</td>
</tr>
<tr>
<td>( Y \sim \text{IG}(\mu, \lambda) )</td>
<td>( \mu ) is a mean and ( \lambda_{IG} = 8 ) is a scale parameter</td>
</tr>
<tr>
<td>( \gamma_0^N \sim N(0, \nu_1^2) )</td>
<td>mean 0 and standard deviation ( \nu_1 = 0.1, 0.2 ) or 0.4</td>
</tr>
<tr>
<td>( \gamma_1^N \sim N(0, \nu_2^2) )</td>
<td>mean 0 and standard deviation ( \nu_2 = 0.1, 0.2 ) or 0.4</td>
</tr>
<tr>
<td>( \gamma_0^Z \sim N(0, \nu_3^2) )</td>
<td>mean 0 and standard deviation ( \nu_3 = 0.1, 0.2 ) or 0.4</td>
</tr>
<tr>
<td>( \gamma_1^Z \sim N(0, \nu_4^2) )</td>
<td>mean 0 and standard deviation ( \nu_4 = 0.1, 0.2 ) or 0.4</td>
</tr>
</tbody>
</table>

**Model assessment and selection**

Good models for claim counts and sizes are necessary for predicting insurance premiums well. The evaluation of the model performance regarding the pure premium is done in term of bias, root-mean-squared error (RMSE) and mean absolute error (MAE)

\[
\text{bias} = \frac{1}{J} \sum_{j=1}^{J} \left( \hat{\Pi}^{pu}_j - \Pi^{pu}_j \right)
\]

\[
\text{RMSE} = \sqrt{\frac{1}{J} \sum_{j=1}^{J} \left( \hat{\Pi}^{pu}_j - \Pi^{pu}_j \right)^2}
\]

\[
\text{MAE} = \frac{1}{J} \sum_{j=1}^{J} |\hat{\Pi}^{pu}_j - \Pi^{pu}_j|.
\]

The bias is used as a metric when we are interested in how close our estimate is to the true value on average, i.e. the mean error. Negative bias means that the true value is underestimated on average. In the opposite case, when the bias is positive, the true value is overestimated on average. Since positive and negative values cancel out, the bias is not a good measure for the spread of the estimates. RMSE square the differences and MAE take the absolute value of them, so the direction of the estimate is eliminated in both
4. SIMULATION STUDY

cases. The spread is hence, better measured by RMSE and MAE. As the RMSE squares
the differences before averaging, it is more sensitive than MAE to outliers. In our study,
the combination of the three metrics is used for model comparison.

Averaging over the metrics from 500 simulated datasets yields

\[
\text{mean bias} = \frac{1}{500} (bias_1 + .. + bias_{500})
\]

\[
\text{mean RMSE} = \frac{1}{500} (RMSE_1 + .. + RMSE_{500})
\]

\[
\text{mean MAE} = \frac{1}{500} (MAE_1 + .. + MAE_{500}).
\]

The claim sizes are larger than the claim counts in real situations. In our simulation, we
chose the parameters $\beta^N$ and $\beta^Z$ such that the expectation of claim sizes was close to the
expectation of claim numbers. This is done so we can use the same standard deviation for
the random effects for claim counts and sizes when simulating data. One can for example
think that the claim sizes are divided by 1000 or 10 000 for a more realistic situation.
In this study the metrics were multiplied by a factor of $10^4$ to make the results more
presentable.
5 Results and discussion

5.1 Part 1: Random intercept

In the first part we simulated data without random effects and data where a random effect was included in the intercept term. The models from Table 3 were used. Each true model was then fitted by the models from Table 3. The value of the standard deviations for the random effects was varied between 0.1, 0.2 and 0.4 in order to see how the random effects influence the premium.

Table 6: Notation in the presentation of the results

| $v^N$ | standard deviation for the random effects for count models |
| $v^Z$ | standard deviation for the random effects for size models |
| $PG$ | Poisson - gamma (GLMs) |
| $PIG$ | Poisson - inverse Gaussian (GLMs) |
| $NBG$ | negative binomial - gamma (GLMs) |
| $NBIG$ | negative binomial - inverse Gaussian (GLMs) |

The results are presented below, with notation listed in Table 6. Fitted GLMMs are symbolized by an uppercase letter "M" (for mixed). For example, the fitted Poisson-gamma GLMM is denoted by $PG_M$. $P$ represents the Poisson, $NB$ the negative binomial, $G$ the gamma and $IG$ the inverse Gaussian.

5.1.1 Looking at the models separately

Poisson-gamma

Results from simulations where claim counts are Poisson distributed and claim sizes are gamma distributed are shown in Tables 7 and 8. The former shows results from simulations without random effects, while the latter shows results from simulations with a random intercept term.

Table 7: Model comparison Part 1: Poisson-gamma fixed effects

<table>
<thead>
<tr>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PG$</td>
<td>$PIG$</td>
</tr>
<tr>
<td>Bias</td>
<td>6.1</td>
</tr>
<tr>
<td>RMSE</td>
<td>539.8</td>
</tr>
<tr>
<td>MAE</td>
<td>276.3</td>
</tr>
</tbody>
</table>
5. RESULTS AND DISCUSSION

For simulations of the Poisson-gamma model without random effects, there are small differences in bias, RMSE and MAE for the models. However, GLMs are slightly better than GLMMs, which is expected since the data are simulated without random effects. The results from $PG_M$ are relatively similar to $PG$, $PIG_M$ to $PIG$, $NBG_M$ to $NBG$, and $NBIG_M$ to $NBIG$. This is reasonable since GLMs are just GLMMs without random effects. $NBIG_M$ gives the worst estimates, which is reasonable since it is the model that differs the most from the true model. Overall $PG$ seems to be the best model for Poisson-gamma data without random effects, which is expected since it is the model used to simulate data.

Table 8: Model comparison Part 1: Poisson-gamma mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$PG$</td>
<td>$PIG$</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.1$, $v^Z = 0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>4.4</td>
<td>7.1</td>
</tr>
<tr>
<td>RMSE</td>
<td>555.6</td>
<td>562.1</td>
</tr>
<tr>
<td>MAE</td>
<td>283.5</td>
<td>285.9</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.2$, $v^Z = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>0.7</td>
<td>3.3</td>
</tr>
<tr>
<td>RMSE</td>
<td>612.0</td>
<td>616.8</td>
</tr>
<tr>
<td>MAE</td>
<td>310.1</td>
<td>311.8</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.4$, $v^Z = 0.4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>−4.2</td>
<td>1.2</td>
</tr>
<tr>
<td>RMSE</td>
<td>885.7</td>
<td>887.3</td>
</tr>
<tr>
<td>MAE</td>
<td>438.4</td>
<td>439.6</td>
</tr>
</tbody>
</table>

Including random effects with standard deviations $v^N$ and $v^Z$ equal to 0.1, the differences in the results are still small for the models. In this case, $PG_M$ is better than $PG$ and $PIG_M$ better than $PIG$. At the same time, $NBG$ is better than $NBG_M$ and $NBIG$ better than $NBIG_M$. This could be because the random effect is so small that it does not make a clear impact on the models, and hence, GLM sometimes gives a better estimate than GLMM, even when the data are affected by random effects. The worst model is $NBIG_M$, but it still provides almost the same results as the other models. The best results are presented by $PG_M$ which is reasonable since it is the true model.

In the case of random effects with standard deviations 0.2, all GLMMs display slightly better results than GLMs, although the differences are still small. This is not unexpected since we are dealing with random effects. For the GLMMs used, $P$ works slightly better than $NB$. This is not surprising since the claim counts are Poisson distributed and affected by random effects. For GLMs, $NB$ provides approximately the same results as
5. RESULTS AND DISCUSSION

$P$. In fact, $NB$ shows a slightly smaller spread than $P$. This could be a result of the dispersion parameter in the negative binomial that makes the $NB$ GLM more flexible than $P$ GLM for the variation caused by the random effects. $PIG$ differs from the true model and does not take random effects into account, and is probably the reason why it shows the worst results. $PG_M$ is the true model and shows the best results as expected.

Increasing $v^N$ and $v^Z$ to 0.4, all GLMMs clearly underestimate the true premium on average. When fitting the simulated data with GLMMs, warnings due to convergence problems sometimes occurred. Unfortunately, it was hard to find a general solution for the convergence issues. Section 5.1.3 shows how the warnings were dealt with. However, the warnings seem to increase when increasing the standard deviations for the random effects. The large difference in bias for GLMMs and GLMs is assumed to be caused by convergence problems. On the other hand, the RMSE and MAE are lower for the GLMMs than for the GLMs. This provides less variation in estimates for GLMMs even though the models may have convergence issues. $NBIG$ gives the estimate that is closest to the true value on average, but it also has a wider spread compared to GLMMs. With GLMMs the uncertainty for the estimate is lower, but in return, they provide worse bias than the GLMs. The best or worst model in this case is hard to point out due to the trade-off between the bias and the spread.

$NB$ appears to give almost the same results as $P$ in Tables 7 and 8, both for GLM and GLMM. This could be because the Poisson can be seen as a limiting case of the negative binomial, since the negative distribution converges to the Poisson as the dispersion parameter goes to zero. $NB$ is therefore a good alternative for Poisson data. The models containing $G$ show better results than the ones containing $IG$ for data simulated without random effects and data with random effects with standard deviation 0.1 and 0.2. This could be because the claim sizes are gamma distributed. In the case with standard deviation for the random effects increased to 0.4, $IG$ shows slightly better bias than $G$ on average for GLM but also a wider spread. The opposite is the case for GLMM, i.e. worse bias on average for $IG$, but with a smaller spread. The odd behavior can be caused by the random effect being so high that some model parameters may not be estimated well, and hence, the convergence warnings. RMSE and MAE increase as the standard deviation for the random effects increase, which is reasonable since we add more sources of variation to our data.

**Poisson-inverse Gaussian**

In Tables 9 and 10, the model for the number of claims is still Poisson, but the sizes are now simulated by the inverse Gaussian model. The set up is similar to Poisson-gamma where the first table shows results from simulations without random effects and the second from simulations with a random intercept term.
5. RESULTS AND DISCUSSION

Table 9: Model comparison Part 1: Poisson-Inverse G. fixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>-6.1</td>
<td>-5.9</td>
</tr>
<tr>
<td>RMSE</td>
<td>525.4</td>
<td>524.4</td>
</tr>
<tr>
<td>MAE</td>
<td>268.9</td>
<td>268.3</td>
</tr>
</tbody>
</table>

For simulations without random effects, the results are approximately the same for all models, which means that GLMMs practically work as well as GLMs. The GLMs are just as good or slightly better than GLMMs in this case. NBG_M differs most from the true model and is also the model with the worst results. With data simulated from the Poisson-inverse Gaussian model, it is not unexpected that the PIG and PIG_M give the best results.

Table 10: Model comparison Part 1: Poisson-inverse G. mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Standard deviations for the random effects: (v^N = 0.1, v^Z = 0.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-0.2</td>
<td>-0.3</td>
</tr>
<tr>
<td>RMSE</td>
<td>540.3</td>
<td>538.6</td>
</tr>
<tr>
<td>MAE</td>
<td>274.0</td>
<td>273.1</td>
</tr>
<tr>
<td>Standard deviations for the random effects: (v^N = 0.2, v^Z = 0.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>606.8</td>
<td>604.6</td>
</tr>
<tr>
<td>MAE</td>
<td>306.2</td>
<td>304.8</td>
</tr>
<tr>
<td>Standard deviations for the random effects: (v^N = 0.4, v^Z = 0.4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-13.3</td>
<td>-11.4</td>
</tr>
<tr>
<td>RMSE</td>
<td>881.6</td>
<td>871.3</td>
</tr>
<tr>
<td>MAE</td>
<td>431.4</td>
<td>427.6</td>
</tr>
</tbody>
</table>

With a standard deviation of 0.1 for the random effects, there are no significant differences between the results from the different models. The GLMs work slightly better than GLMMs. The random effects might be too small to affect the models to the extent that GLMMs are better than GLMs. NBG_M can be taken to be the worst, and PIG the best model according to the spread. This is expected from the model used to simulate data if we assume that GLMs could be preferable for small values of \(v^N\) and \(v^Z\).
5. RESULTS AND DISCUSSION

When $v^N$ and $v^Z$ are increased to 0.2 for the random effects, the results are still almost the same for the compared models. GLMMs now provide better estimates than GLMs, so increasing the standard deviations to 0.2 seems to have a more significant impact on the models. $PG$ is the worst model although the claim counts are Poisson distributed. $NBG$ gives approximately the same results as $PG$, but with a slightly lower spread. This is assumed to be because the negative binomial contains a dispersion parameter which enables it to take some of the random effects into account. $PIG_M$ and $NBIG_M$ give better estimates than the other models. This is not surprising since data were simulated from the Poisson-inverse Gaussian model with random effects included.

Increasing the standard deviations for the random effects to 0.4 tends to result in the same problems as the Poisson-gamma case. All models underestimate the premium, especially the GLMMs which is assumed to be caused by convergence issues. For GLMMs, models containing $IG$ show worse bias than the models containing $G$, but smaller spread. The reason can be that the model parameters sometimes are not estimated well when the standard deviation for the random effects is high (0.4 in our study). Regarding the spread, $PIG_M$ and $NBIG_M$ are the best models which is expected, but they also have the worst bias on average. $PIG$ gives the best bias on average but has a wider spread than the GLMMs. Choosing the best and worst model seems to be difficult in this case due to the trade-off between bias and spread.

Fitting Poisson distributed claim counts with $NB$ still yields approximately the same results as with $P$ which shows again that $NB$ is a good alternative for Poisson distributed data. In Tables 9 and 10 the models containing $IG$ are slightly better than the ones with $G$. This is expected as the claim sizes are now inverse Gaussian distributed. The exception is the case of GLMMs where the standard deviation for the random effects is 0.4, which is assumed to be caused by the convergence issues. The gamma model seems to work better for inverse Gaussian distributed data than the other way around in our simulations, since the results from $G$ and $IG$ are closer to each other in this case. Increasing the standards deviation for the random effects continues to show more variation in the estimates.

Negative binomial-gamma
Another model used for the simulations is the negative binomial-gamma model. The results are given in Tables 11 and 12.


5. RESULTS AND DISCUSSION

Table 11: Model comparison Part 1: negative binomial-gamma fixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>$-2.0$</td>
<td>$-0.0$</td>
</tr>
<tr>
<td>RMSE</td>
<td>601.9</td>
<td>606.4</td>
</tr>
<tr>
<td>MAE</td>
<td>302.5</td>
<td>304.4</td>
</tr>
</tbody>
</table>

Firstly we look at simulations without random effects. Except the $NBG_M$ and the $NBIG_M$, Table 11 provide fairly similar results. $NBG_M$ and $NBIG_M$ display a larger spread than the other models. When fitting data with GLMMs, the function `glmer.nb` is noticeably slower compared to the other distributions used. It may indicate that the estimation of parameters is more challenging with `glmer.nb` and can cause more uncertainty around the estimate. Bolker et al. (2017) mention that `glmer.nb` is somewhat slow and fragile compared to the other methods that are available in the same packages, which reinforces our suspicion. However, the noticeably greater spread for $NBG_M$ and $NBIG_M$ does not seem to arise when using `glmer.nb` for Poisson distributed claim counts. The `glmer.nb` function is therefore assumed to be more fragile for negative binomial distributed data which are more overdispersed than Poisson distributed data in general. One may anticipate that $PIG_M$ gives the worst results since it differs most from the true model. However, it is not the case. It turns out that $NBIG_M$ gives the worst estimate in our simulation study. Since both $NBG_M$ and $NBIG_M$ show higher RMSE and MAE than the other models, we have reason to believe that the function `glmer.nb` is less reliable than the other functions used, at least in our study. $NBG$ should provide the best results since it is the true model. However, it shows the lowest spread as expected, but not the best average bias.
## RESULTS AND DISCUSSION

Table 12: Model comparison Part 1: negative binomial-gamma mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$PG$</td>
<td>$PIG$</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.1$, $v^Z = 0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>$-3.3$</td>
<td>$-1.4$</td>
</tr>
<tr>
<td>RMSE</td>
<td>625.4</td>
<td>630.0</td>
</tr>
<tr>
<td>MAE</td>
<td>313.1</td>
<td>315.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.2$, $v^Z = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>$-5.8$</td>
<td>$-2.6$</td>
</tr>
<tr>
<td>RMSE</td>
<td>682.0</td>
<td>683.4</td>
</tr>
<tr>
<td>MAE</td>
<td>337.4</td>
<td>338.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.4$, $v^Z = 0.4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>$-10.7$</td>
<td>$-5.5$</td>
</tr>
<tr>
<td>RMSE</td>
<td>955.7</td>
<td>958.0</td>
</tr>
<tr>
<td>MAE</td>
<td>463.3</td>
<td>464.7</td>
</tr>
</tbody>
</table>

Including random effects with standard deviations 0.1, GLMMs shows slightly better bias on average, although the differences are small. This is expected since data were simulated with random effects. Except for the $NBG_M$ and the $NBIG_M$, the spread is also slightly lower for GLMMs than GLMs. The wide spread in $NBG_M$ and $NBIG_M$ is assumed to be related to the estimation issues when using `glmer.nb` for negative binomial distributed data. $NBG_M$ and $NBIG_M$ give the closest estimates to the true premium on average, but also show higher RMSE and MAE. The models with the best bias on average tend to have a wider spread in this case. The choice of model therefore depends on the trade-off between the bias and the spread.

For simulations with standard deviation 0.2 for the random effects, there are similarities to the case with standard deviation 0.1. For instance, there are small differences in the results for the compared models, the bias is better for GLMMs on average and that $NBG_M$ and $NBIG_M$ have slightly wider spread than the other models. Overall, $PG$ may be chosen as the worst model, which is reasonable since it differs the most from the true model. It has the worst bias on average, and the spread is not so different from the compared models, except the $NBG_M$ and the $NBIG_M$. $NBIG_M$ is the worst model in this case according to the spread, but it also shows the best bias on average. Identifying the best model again depends on whether we put emphasis on the bias or the spread.

For the case of standard deviations of the random effect equal to 0.4, GLMMs clearly underestimate the true value on average, but the estimates have a smaller spread than GLMs. The same trend was also obtained for the Poisson-gamma and Poisson-inverse Gaussian case. This reinforces the assumption that the convergence problem is challeng-
ing for the GLMMs when the values of $v^N$ and $v^Z$ are high.

Greater standard deviation for the random effects adds more variation to the data. As earlier, it results in higher RMSE and MAE. It is worth mentioning that RMSE and MAE are higher in general when claim counts are simulated from the negative binomial distribution. Recall that the variance of the negative binomial model can be expressed by $\mu + \tau \mu^2$ rather than $\mu$ as in the Poisson model. In our simulations, we choose $\tau$ to be positive such that the variance of the outcome is larger for negative binomial data than the poisson data. More variation in the data can make them more challenging to fit, and it is assumed to be the reason that RMSE and MAE are in general higher when claim counts are simulated from the negative binomial distribution.

**Negative binomial-inverse Gaussian**

Combining the negative binomial distributed claim counts and inverse Gaussian claim sizes without and with random effects yields the results in Tables 13 and 14 respectively.

**Table 13:** Model comparison Part 1: negative binomial-inverse G. fixed effects

<table>
<thead>
<tr>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PG$</td>
<td>$PIG$</td>
</tr>
<tr>
<td>Bias</td>
<td>-1.6</td>
</tr>
<tr>
<td>RMSE</td>
<td>573.5</td>
</tr>
<tr>
<td>MAE</td>
<td>288.7</td>
</tr>
</tbody>
</table>

When simulating data from the negative binomial-inverse Gaussian model without random effects, the differences in the results are small. What stands out are the results for $NBG_M$ and $NBIG_M$. The reason is assumed to be the estimation uncertainty with the `glmer.nb` function. For GLMMs $P$ works better than $NB$, but we are not focusing on that point since we assumed that the `glmer.nb` function does not give us a reliable estimate. Excluding $NBG_M$ and $NBIG_M$, the results are almost the same for all models, but $PG$ and $PG_M$ yield slightly worse results. This is expected since the claim counts are negative binomial distributed and the claim sizes are inverse Gaussian distributed. $NBIG$ is the true model and as expected, also the best model.

We now include random effects with standard deviations of 0.1. Once again, the function `glmer.nb` seems to give more uncertainty around the estimates since the spread for $NBG_M$ and $NBIG_M$ is larger than the spread for the compared models. In return, they show a better bias on average. Otherwise, there are small differences in the results, so GLM practically works as well as GLMM. $NBG$ shows slightly worse bias on average, but has a somewhat smaller spread than $PG$, $PG_M$, $NBG_M$ and $NBIG_M$. $NBIG_M$ is the best model according to the bias, which is reasonable since it is the true model. However, the spread is larger than most of the models. This is assumed to be caused by the uncertainty
5. RESULTS AND DISCUSSION

Table 14: Model comparison Part 1: negative binomial-inverse G. mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG  PIG NBG NBIG</td>
<td>PG M PIG M NBG M NBIG M</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.1$, $v^Z = 0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-3.3 -2.8 -3.8 -3.3</td>
<td>-3.4 -3.0 -1.3 -0.8</td>
</tr>
<tr>
<td>RMSE</td>
<td>593.8 590.0 591.0 587.2</td>
<td>593.7 589.8 614.1 610.7</td>
</tr>
<tr>
<td>MAE</td>
<td>299.4 298.0 298.1 296.7</td>
<td>299.3 297.8 307.9 306.7</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.2$, $v^Z = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-9.2 -8.4 -9.4 -8.6</td>
<td>-9.1 -8.3 -6.2 -5.5</td>
</tr>
<tr>
<td>RMSE</td>
<td>661.3 657.3 657.7 653.8</td>
<td>658.8 654.9 674.2 670.6</td>
</tr>
<tr>
<td>MAE</td>
<td>329.0 327.1 327.4 325.5</td>
<td>328.1 326.2 334.7 332.9</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v^N = 0.4$, $v^Z = 0.4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-2.1 1.1 -1.7 1.5</td>
<td>-51.2 -79.6 -49.1 -77.6</td>
</tr>
<tr>
<td>RMSE</td>
<td>952.3 945.7 941.8 934.8</td>
<td>894.7 881.7 905.3 891.4</td>
</tr>
<tr>
<td>MAE</td>
<td>467.5 464.0 463.0 459.4</td>
<td>444.1 437.2 448.9 441.6</td>
</tr>
</tbody>
</table>

around the estimate when using the \texttt{glmer.nb} function. With approximately the same results, the model with the best average bias does not have the smallest spread, and the model with the smallest spread does not have the best bias. Therefore the choice of the best and the worst model depends on whether we put emphasis on the bias or the spread.

For random effects with standard deviations of 0.2, the differences are still small between the models. The data are now simulated with random effects so GLMMs are expected to give better estimates. GLMMs overall work slightly better than GLMs. Neither the worst nor the best models are easy to point out in this case. One may favor the $NBIG_M$ since it provides the best bias on average. In that case, the assumption that the true model is the best model is met. Both $NBG_M$ and $NBIG_M$ show larger spread than the other models, so one should not exclude the possibility that the higher spread of those models is related to the estimation issues for the \texttt{glmer.nb} function.

Changing the standard deviations to 0.4, the GLMMs have significantly lower RMSE and MAE than GLMs. The premium is, however, underestimated by GLMMs on average. We again believe that the GLMMs used have convergence issues for a high value of the standard deviation for the random effects. For GLMs, models containing $IG$ provides slightly better results than models with $G$, but for GLMMs, models containing $IG$ show worse bias than models containing $G$ on average. This observation differs from the cases where lower $v_N$ and $v_Z$ were used for the negative binomial-inverse Gaussian model. The behavior can be caused by the random effect being so high that some of the models struggle to give plausible estimates, hence, the convergence warnings.
5. RESULTS AND DISCUSSION

Except the case with the standard deviation for the random effects equal to 0.4, Table 13 and 14 show that models that include IG give slightly better estimates than the ones with G. This is expected since the claim sizes are inverse Gaussian distributed. Overall, P provides approximately the same results as NB for negative binomial distributed claim counts, which shows that P can be a good model for negative binomial distributed data in our study. However, the spread between P GLM and NB GLM are slightly more similar for Poisson data than for negative binomial data, which means that NB is somewhat more flexible than P. It is not implausible that another chosen value for the dispersion parameter used when generating data could make a more clear difference. Increasing the standard deviation for the random effects also increases the spread of the estimates. Like for the negative binomial-gamma model, RMSE and MAE tend to be higher compared to the models with Poisson distributed claim counts. This is assumed to be caused by the higher variance of the outcome of the negative binomial data than the Poisson data that makes the data slightly more difficult to estimate.

5.1.2 General discussion of Part 1

In Section 5.1.1, data were simulated without random effects and with a random intercept term. Increasing the standard deviation for the random effects also increases the uncertainty around the estimates, which is reasonable since more variation is added to the model. Overall, GLM gives slightly better results than GLMM when data are simulated without random effects. With standard deviations of 0.1 for the random effects, it is difficult to say whether the GLM or the GLMM yields the best results. Increasing $v^N$ and $v^Z$ to 0.2 results in the GLMM giving slightly better results than the GLM.

GLMMs underestimate the true premium on average for all cases with 0.4 for $v^N$ and $v^Z$. Since convergence warnings seem to occur more frequently when increasing the standard deviation for the random effects, we assume that GLMMs have convergence issues when the standard deviation for random effects becomes high. Recall that a random intercept model can be expressed by

$$\mu = x\beta + \gamma, \quad \gamma \sim N(0, v^2).$$

A large value for $v^2$ compared to $x\beta$ can make the random variation so large that GLMMs struggle to identify the model and some model parameters may not be estimated well (they are not identifiable). For all cases with 0.4 for $v^N$ and $v^Z$, GLM provides worse bias than GLM on average, but has a lower spread. It is therefore often difficult to determine which model is the best since we often end up with a trade-off between the bias and the variation of the estimate.

Another noticeable issue with GLMMs is the glmer.nb function from the lme4 package. It is mentioned in the GLMM FAQ (Bolker et al., 2017) that glmer.nb is slower and more fragile than the glmer function from the same package, which coincides with our experience. In our study, the glmer.nb function seems to have more of a problem for negative binomial distributed data than Poisson distributed data. This is assumed to
be caused by higher variance of outcome of the negative binomial data than the Poisson data, which could make the data slightly more challenging to fit.

There are small differences between the results from $P$ and $NB$. Also the differences in the results for $G$ and $IG$ are small. In fact, GLM and GLMM also yield approximately the same results. From a practical perspective, choosing the 'wrong' model for the data seems to not have considerable consequences as long as the choice of model is sensible (for example that one should not use a normal distribution for claim size modeling). GLMM is more computationally expensive and demands more computing time than GLM. The only case where GLMM is preferable, according to the results, is the case of random effects with standard deviations 0.2, and only for simulation of the Poisson-gamma and the Poisson-inverse Gaussian model. Even then, the results are approximately the same. From an insurance company's perspective, a good estimate of the premium is important. To keep themselves solvent, the premium needs to meet the claims obligations. A lower spread means that there is less uncertainty in the estimate, but the fact that the bias is worse on average makes the GLMMs less preferable than GLMs.

At the moment, the functions `glmer` and `glmer.nb` from the `lme4` packages seem to not be as robust as we need them to be, at least for our purpose. Convergence problems seem to be a challenge in our study. We do not exclude that GLMMs from other packages may perform better, but few have all Poisson, negative binomial, gamma and inverse Gaussian GLMM included. Hence, the `lme4` package was chosen.

### 5.1.3 Troubleshooting warnings and errors

It is worth mentioning that warnings of models not converging may occur when using the `glmer` and the `glmer.nb` function to fit the data. After testing several simulations, it seems as if the higher the standard deviations, the more warnings. Changing the optimization method can remove the warnings in many cases, but the best method to do this is dependent on the data. This means that some methods work better for different datasets. Since the simulation is over 500 sets, it is difficult to find an optimal method. The 'bobyqa' optimizer from the `minqa` package was chosen, as it works well in many cases, but can still result in warnings. However, if many optimizing methods converge to values that are almost the same, one may consider the convergence warnings to be false positives (Bates et al., 2017). Ben Bolker, author of the `glmer` function, provides more detailed descriptions of troubleshooting due to convergence problems for the `lme4` package with a specific example in Bolker (2014).

This section illustrates a case where warnings occurred and how different optimizers can be used to overcome the failures. The seed of R's random number generator was set to 9, by setting `set.seed(9)`, so the simulation can be reproduced. The standard deviations for the random effects that was used for this example is 0.2. The data used are simulated from the negative binomial-gamma model with a random intercept term. The R-code can be found in Appendix B. Convergence warnings are often similar to
5. RESULTS AND DISCUSSION

Warning message:
In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.791889 (tol = 0.001, component 1)

When we fit the claim sizes with the gamma GLMM and the inverse Gaussian GLMM we get two warnings that indicate convergence problems, one for gamma and one for inverse Gaussian. Changing the optimization method to "nlminb" from the optimx package makes the warning from the gamma GLMM disappear. Using the optimization method "bobyqa" on the other hand removes both warnings. The "bobyqa" optimizer is therefore the best alternative for this case. Note that it is not necessarily always the case. Another problem when fitting data was that the error

Error in pwrssUpdate(pp, resp, tol = tolPwrss, GQmat = GQmat, compDev = compDev,: pwrssUpdate did not converge in (maxit) iterations

sometimes occurred. It was only experienced for random effects with standard deviations 0.4. A general solution for this error was hard to find, so to get around it, we used the try function and started over when the error occurred. A sketch of how the function was used is

for(i in 1:sim){
  repeat{
    if(is.character(func)){
      .. #try again!
    }else{
      .. #do something!
      break #break the repeat
    }
  }
}

The code repeats the simulation if an error occurs. If not, it works as if an error did not occur. The whole code can be found in Appendix B.

5.2 Part 2: Two random effects

For this part, we simulate a new set of data with multiple random effects. In this case not only the random intercept term is influenced by a random effect, but also one of the covariates. More specifically, we let the intercept term and $x_1$ be affected. Due to time limitations, we choose to only simulate data from the Poisson-gamma and the Poisson-inverse Gaussian models. In Part 2 we are interested in if GLMMs give significantly better estimates than GLMs for data affected by several random effects. We also simulate data without random effects to see the consequences of using GLMMs which take several random effects into account for data without random effects.
5. RESULTS AND DISCUSSION

5.2.1 Looking at the models separately

Poisson-gamma

Table 15: Model comparison Part 2: Poisson-gamma fixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>4.4</td>
<td>6.7</td>
</tr>
<tr>
<td>RMSE</td>
<td>541.7</td>
<td>546.6</td>
</tr>
<tr>
<td>MAE</td>
<td>276.2</td>
<td>278.3</td>
</tr>
</tbody>
</table>

For simulated data without random effects, GLM proves to give better results than GLMM. The differences in spread are small, but the GLMMs underestimate the true premium on average. Comparing it to Part 1 of the simulation study, GLMM works poorly when taking more random effects into account.

Table 16: Model comparison Part 2: Poisson-gamma mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>−3.8</td>
<td>−1.8</td>
</tr>
<tr>
<td>RMSE</td>
<td>578.0</td>
<td>583.2</td>
</tr>
<tr>
<td>MAE</td>
<td>294.8</td>
<td>297.1</td>
</tr>
</tbody>
</table>

Standard deviations for the random effects: $v_1 = v_2 = v_3 = v_4 = 0.1$

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>7.9</td>
<td>10.4</td>
</tr>
<tr>
<td>RMSE</td>
<td>710.2</td>
<td>714.5</td>
</tr>
<tr>
<td>MAE</td>
<td>353.6</td>
<td>355.7</td>
</tr>
</tbody>
</table>

Standard deviations for the random effects: $v_1 = v_2 = v_3 = v_4 = 0.2$

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>0.8</td>
<td>9.8</td>
</tr>
<tr>
<td>RMSE</td>
<td>1398.5</td>
<td>1376.1</td>
</tr>
<tr>
<td>MAE</td>
<td>662.9</td>
<td>655.1</td>
</tr>
</tbody>
</table>

For data affected by two random effects, GLM yields better results than GLMM. When the standard deviations of the random effects increase, the differences regarding bias between GLM and GLMM increase. Higher standard deviations of the random effects result in worse results for GLMMs compared to GLMs. This is surprising since GLMMs take the random effects into account and should therefore, theoretically, perform better than
5. RESULTS AND DISCUSSION

GLMs. On the other hand we experience similar results in Part 1, where GLMMs in some cases struggled when data were simulated with random effects. It is possible that more random effects make the models more complicated and result in difficulties such as convergence issues. Compared to Part 1, we see that the spread generally is higher when we are dealing with two random effects. This is expected since there is more variation in the data. Increasing the standard deviation of the random effects also increases the spread for the case of two random effects.

**Poisson-inverse Gaussian**

**Table 17:** Model comparison Part 2: Poisson-inverse G. fixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td>Bias</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>RMSE</td>
<td>526.1</td>
<td>522.8</td>
</tr>
<tr>
<td>MAE</td>
<td>266.3</td>
<td>264.8</td>
</tr>
</tbody>
</table>

With data generated from the Poisson-inverse Gaussian model without random effects the results do not differ much from the case of Poisson-gamma without random effects. GLM yields better results than GLMM and the GLMMs underestimate the true premium on average.

**Table 18:** Model comparison Part 2: Poisson-inverse G. mixed effects

<table>
<thead>
<tr>
<th></th>
<th>Generalized linear models</th>
<th>Generalized linear mixed models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG</td>
<td>PIG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>3.1</td>
<td>3.8</td>
</tr>
<tr>
<td>RMSE</td>
<td>574.1</td>
<td>572.0</td>
</tr>
<tr>
<td>MAE</td>
<td>288.4</td>
<td>287.4</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v_1 = v_2 = v_3 = v_4 = 0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>0.6</td>
<td>2.0</td>
</tr>
<tr>
<td>RMSE</td>
<td>720.7</td>
<td>714.3</td>
</tr>
<tr>
<td>MAE</td>
<td>354.1</td>
<td>351.2</td>
</tr>
<tr>
<td>Standard deviations for the random effects: $v_1 = v_2 = v_3 = v_4 = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>-19.8</td>
<td>-15.5</td>
</tr>
<tr>
<td>RMSE</td>
<td>1440.7</td>
<td>1400.6</td>
</tr>
<tr>
<td>MAE</td>
<td>682.7</td>
<td>666.3</td>
</tr>
</tbody>
</table>

45
For data generated from the Poisson-inverse Gaussian model with multiple random effects, the trend is the same as in the Poisson-gamma case. We see that the GLM shows better results than GLMM, and that the distance between them regarding bias increases when the standard deviations of the random effects increase. This is assumed to be due to the fact that the models become more complicated and therefore struggle with convergence issues.

5.2.2 General discussion of Part 2

More random effects should favor GLMMs over GLMs in theory, but our experience shows the opposite. GLMs provide better results than GLMMs throughout Part 2. In Part 1 we saw that GLMMs did not perform noticeably better than GLMs for data affected by random effects, probably because of convergence problems. With more random effects added to the data, the model might be too complicated for GLMMs in practice, at least with the \textit{lme4} package. For data with a random intercept term, we used five quadrature points, specified by \texttt{nAGQ=5}, for all GLMMs. At present, only \texttt{nAGQ=1}, i.e. the Laplace approximation is available for GLMMs with several random effects with the \textit{lme4} package. Since more quadrature points in general provide higher accuracy, using the Laplace approximation might affect the estimates.
6 Conclusion

GLMs are commonly used for insurance applications. GLMMs are more flexible and powerful, but are also more complicated, and parameter estimation can be more challenging. The purpose of the study was to see how well insurance data can be estimated by GLMs and GLMMs, both when affected by random effects and not, regarding the premium. We wanted an insight into how much estimates of the pure premium are affected when using GLMs in the case of random effects, and to answer whether the estimates of the pure premium can be good even when we are using the wrong models. We also wanted to see if GLMMs give noticeably better results when the true model is affected by several random effects. It was solved by a comparison of models to see how well they fit our data.

The true pure premium can be expressed by the product of the expected claim counts and the expected cost per claim. To generate the claim counts, the Poisson and the negative binomial were used. This is because they are common count models for insurance data. For claim sizes, the most common models are the gamma and the inverse Gaussian model, which are also the ones that were used in our study. Combinations of those models, either with or without random effects, gave eight models. Each model that was used to generate the data was then fitted by the true model and the seven other models. This was done 500 times to provide a stable pattern.

The results from Part 1 of the simulation study displayed a small bias on average for GLMM and GLM, which showed that GLMM and GLM are appropriate tools for insurance data when the total premium is of interest. The uncertainty around the estimates increased when the random effect in the data increased. This was inferred from the increasing spread.

In this study the value of the average bias tends to decrease when increasing the standard deviations for the random effects. This applies to both GLMs and GLMMs. The changes, however, are so small that we can only say that the estimates of the premium are slightly affected by the random effects. In fact the random effects influence the estimate for GLMMs more than they influence the estimate for GLMs.

Looking into the different models, the Poisson and the negative binomial model have many similarities, and give good fit to insurance data. They are discrete models, have non-negative values and are right-skewed. The negative binomial includes a dispersion parameter and is slightly more flexible than the Poisson. Gamma and inverse Gaussian also have many similarities. They are right-skewed, non-negative and are continuous models. Because of the similarities, there were very small differences in the results. So using the wrong model did not have a great impact on the results in our study.

To see if GLMM gives noticeably better results when the data are affected by several random effects, we generated data with two random effects. The data were fitted with GLMs, which do not take the random effects into account, and GLMMs, which take the
random effects into account. Surprisingly, GLM did a better job than GLMM. For further research, it could be interesting to explore different \texttt{R}-packages.

The comparison of results showed that GLMMs are a lot more computationally expensive than GLMs. At the same time they yielded very similar results. From an insurer’s point of view what is important is the pure premium, the individual premium and time. One wants accurate, simple models that are robust. The study showed that GLM mostly gave better estimates for the total premium on average than GLMM, but the spread was larger in some cases. With data affected by several random effects, the study favors GLM. Taking the long computational time of the GLMMs into account and knowing that they do not stand out regarding results, GLMs are preferable. In addition the study demonstrated the challenges of GLMM, with convergence issues, at least with the \textit{lme4} package in \texttt{R}. In particular cases it is possible to find methods to overcome these issues. However, it is difficult to find a universal solution that works for all data sets. GLMMs also struggle with insurance data when the data are affected by random effects, especially in the cases of several random effects. GLMM therefore seems to be unreliable and should be used carefully.
References


REFERENCES


Appendices

A Notations and acronyms

A.1 Notations

- small letters are scalars
- small bold letters are vectors
- capital bold letters are matrices

- $c$ index of observations within clusters $c = 1, \ldots, m$
- $d$ variance-covariance matrix for the random effects for a single policy
- $D$ block diagonal matrix with entries $d$
- $i$ index of observations, $i = 1, \ldots, n$
- $k$ number of fixed effects
- $m$ sample size within clusters
- $n$ sample size
- $N$ claim counts
- $Q$ quadrature points for the AGHQ
- $t$ offset
- $u$ row vector of covariates whose effects are assumed to be random across clusters
- $v$ standard deviation for the random effects
- $x$ row vector of covariates whose effects are assumed to be fixed
- $X$ design matrix
- $y$ observed value of $Y$
- $y$ observed vector of $Y$
- $Y$ response variable
- $Y$ column vector of response variables
- $Z$ cost of claims

- $\beta$ single model coefficient
- $\beta$ column vector of model coefficients
- $\gamma$ single random effect
- $\gamma$ column vector of random effects
- $\eta$ linear predictor
- $\eta$ column vector of linear predictor
- $\Pi^{pu}$ insurance pure premium
- $\Psi$ model parameters to be estimated
A. NOTATIONS AND ACRONYMS

\( g(\cdot) \)  link function in GLM and GLMM
\( L(\cdot) \)  likelihood function
\( \ell(\cdot) \)  log-likelihood function
\( s(\cdot) \)  score function
\( I(\cdot) \)  Fisher information
\( S(\cdot) \)  score vector
\( I(\cdot) \)  Fisher information matrix
\( H(\cdot) \)  Hessian matrix

\[ \begin{bmatrix} \cdot \end{bmatrix}^t \]  transpose of a vector or matrix
\( E[Y] \)  expectation of \( Y \)
\( E[Y|X] \)  conditional expectation of \( Y \) given \( X \)
\( \text{var}[Y] \)  variance of \( Y \)
\( \text{var}[Y|X] \)  conditional variance of \( Y \) given \( X \)

\( \text{gamma}(\cdot) \)  gamma distribution
\( \text{IG}(\cdot) \)  inverse Gaussian distribution
\( N(\cdot) \)  normal distribution
\( \text{NB}(\cdot) \)  negative binomial distribution
\( \text{Poisson}(\cdot) \)  Poisson distribution

A.2 Acronyms

AGHQ  adaptive Gauss-Hermite quadrature
GHQ  Gauss-Hermite quadrature
GLM  generalized linear model
GLMM  generalized linear mixed model
HGLM  hierarchical generalized linear model
iid  independent and identically distributed
LM  classical linear model
LSE  least squares estimation
MCMC  Markov chain Monte Carlo
MLE  maximum likelihood estimate
pdf  probability density function
pmf  probability mass function
PQL  penalized quasi-likelihood
B. R-CODE

B. R-code

The packages, the model specifications and the function used for simulations of covariates are identical for both Part 1 and Part 2 of the simulation study. The standard deviations for the random effects were varied between 0 (for fixed effects only), 0.1, 0.2 and 0.4.

```r
#install.packages("MASS")
#install.packages("statmod")
#install.packages("lme4")

library(MASS) #for use of negbin
library(nlme) #for use of mgcv
library(statmod) #for use of inverse gaussian
library(lme4)

########################################
#### model specifications ####
########################################
sim = 500 #500 simulations
years=5 #policies over 5 years
J=1000*years #1000 policies over 5 years
id=seq(1:(J/years)) #id for i=1,...,1000 policyholders
v_n = 0.2 #standard deviation for random effect in claim counts
v_z = 0.2 #standard deviation for random effect in claim sizes
tau = 0.2 #choose a tau for negbin
k_gamma=6 #choose a shape parameter k for gamma sizes
lambda_ig=8 #choose a lambda for inverse gaussian sizes

#choose beta_n for claim counts and beta_z for claim sizes
beta_n = rbind(-2.26,0.32,0.34,0.24,0.71,-0.52,0.22,-0.13,0.12,-0.13,0.49,0.61,-0.42,-0.62,1.20,0.21)
beta_z = rbind(-2.32,0.31,-0.19,0.27,0.31,-0.22,0.43,0.23,0.28,0.26,0.23,0.31,-0.21,0.12,0.64,0.22)

########################################
#### function (simulate covariates) ####
########################################
cov = function() {

#covariates x9 and x11 remain the same for the policyholders each year
x0=rep(1,J)
x1=rnorm(J,1,0.2)
x2=rnorm(J,1,0.2)
x3=rnorm(J,1,0.2)
x4=rnorm(J,1,0.2)
x5=rnorm(J,1,0.2)
x6=rnorm(J,1,0.2)
```

53
B. R-CODE

```r
x7=rnorm(J,1,0.2)
x8=rbinom(J,1,0.5)
x9=rep(rbinom(J/years,1,0.5),years) #e.g gender, same for policies over 5 years
#multinomial with 3 categorial
x10=t(rmultinom(J,1,prob=c(0.2,0.3,0.5)))[,-1]
#multinomial with 5 categorial
x11=t(rmultinom(J/years,1,prob=c(0.2,0.05,0.35,0.1,0.3)))[,-1]
x11=matrix(rep(t(x11),years),ncol=ncol(x11),byrow=TRUE)

cov16=cbind(x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11) #matrix with 16 columns

return(cov16)
}
```

B.1 Computer program for Part 1 of the simulation study

```r
###########################################
#### function 1 (poisson+gamma part 1) ####
###########################################

claims_pg=function(){

#J, id, years, k_gamma from program
#exposure t_i = 1 for all policies for simplicity

cov16= cov() #draw covariates

#simulate claim counts
r = rep(rnorm(J/years,0,v_n),years) #same effect for same policy over 5 years
lambda = exp(cov16%*%beta_n + r) #mean value for claim counts with log-link
claim_counts = rpois(J,lambda) #true model for claim counts
mat = cbind(id,cov16,claim_counts) #matrix: id, covariates, claim counts
mat = data.frame(mat) #make data frame

#GLMs and GLMMs for claim counts
mod_p = glm(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat,family=poisson(log))

mod_pr = glmer(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat,family=poisson(log),nAGQ=5,control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_nb = glm.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,data=mat,family=poisson(log))

mod_nbr = glmer.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat,link=log,nAGQ=5,control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
```
# Simulate claim sizes

```r
#same effect for same policy over 5 years
r = rep(rnorm(J/years, 0, v_z), years) # mean value for claim sizes with log-link
mu = exp(cov16%*%beta_z + r) # claim = 1 size
claim_sizes = matrix(NaN, J, max(claim_counts)) # row=J, col=max number of claims
```

#Simulate gamma claim sizes for each policy N_ij

```r
for(i in 1:J){
  if(claim_counts[i] > 0){ # when claim_counts[i] = 0, go to next loop
    z_i = mu[i]*rgamma(claim_counts[i], k_gamma)/k_gamma # 1 claim = 1 size
    claim_sizes[i, 1:length(z_i)] = z_i # insert z_i in matrix claim_sizes
  }
}
```

claim_sizes = as.vector(t(claim_sizes)) # convert matrix to vector
claim_counts = claim_sizes[!is.nan(claim_sizes)] # remove nan values
mat_new = mat[mat$claim_counts>0,] # remove all rows where claim numbers is zero

# Insert new line for policy with more than one claim
```r
j=1 # new variable
for (i in mat_new$claim_counts){ # i is size of claim numbers for policy i
  if (i > 1){ # if test whenever claim numbers larger than one
    for (a in 1:(i-1)){ # only when claim numbers larger than one
      # insert new line for same policy
      mat_new = rbind(mat_new[1:j,], mat_new[j,], mat_new[-(1:j),])
    }
    j = j+i # plus i lines of same policy
  } else{
    j = j+1 # jump to next policy
  }
}
```

mat_new$claim_counts = claim_sizes # don’t need claim numbers
```r
colnames(mat_new)[18] = "claim_sizes" # change column name in mat_new
mat_new = data.frame(mat_new) # make data frame
```

# GLMs and GLMMs for claim sizes
```r
mod_g = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=Gamma(log))
```

```r
mod_gr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+
  V17+(1|id),data=mat_new,family=Gamma(log),nAGQ=5,control=glmerControl(
  optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
```

```r
mod_ig = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=inverse.gaussian(log))
```

```r
mod_igr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+
  V17+(1|id),data=mat_new,family=inverse.gaussian(log),nAGQ=5,control=
```
B. R-CODE

glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5))

var_pr=VarCorr(mod_pr); var_nbr=VarCorr(mod_nbr)
var_gr=VarCorr(mod_gr); var_igr=VarCorr(mod_igr)
var_sim=cbind(var_pr,var_nbr,var_gr,var_igr) #estimated var. random effects
var_sim=as.numeric(var_sim) #numerical vector

#return cov, parameters, true premium, estimated variance of random effects
count_sim=c(coef(mod_p),coef(mod_nb),fixef(mod_pr),fixef(mod_nbr))
size_sim=c(coef(mod_g),coef(mod_ig),fixef(mod_gr),fixef(mod_igr))
pi_true = exp(cov16%*%beta_n+v_n^2/2)*exp(cov16%*%beta_z+v_z^2/2) #true premium

return(list(cov16=cov16,beta_sim=c(count_sim,size_sim), pi_true=pi_true, var_sim=var_sim))

#####################################################
#### function 2 (poisson+inverse gaussian part 1) ####
#####################################################
claims_pig=function(){

#J, id, years, lambda_ig from program
#exposure t_i = 1 for all policies for simplicity.

cov16=cov() #draw covariates

#simulate claim counts
r = rep(rnorm(J/years,0,v_n),years) #same effect for same policy over 5 years
lambda = exp(cov16%*%beta_n + r) #mean value for claim counts with log-link
claim_counts = rpois(J,lambda) #true model for claim counts
mat = cbind(id,cov16,claim_counts) #matrix: id, covariates, claim counts
mat = data.frame(mat) #make data frame

#GLMs and GLMMs for claim counts
mod_p = glm(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17, data=mat,family=poisson(log))

mod_pr = glmer(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat,family=poisson(log),nAGQ=5,control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_nb = glm.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,data=mat,link=log)

mod_nbr = glmer.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat,link=log,nAGQ=5,control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
#simulate claim sizes
r = rep(rnorm(J/years,0,v_z),years) #same effect for same policy over 5 years
mu = exp(cov16%*%beta_z + r) #mean value for claim sizes with log-link
claim_sizes = matrix(NaN,J,max(claim_counts))#row=J, col=maximum number of claims

#simulate inverse gaussian claim sizes for each policy N_ij
for(i in 1:J){
  if(claim_counts[i] >0){ #when claim_counts[i] = 0, go to next loop
    z_i = rinvgauss(claim_counts[i],mu[i],shape=lambda_ig)#1 claim = 1 size
    claim_sizes[i,1:length(z_i)] = z_i #insert z_i in matrix claim_sizes
  }
}
claim_sizes=as.vector(t(claim_sizes)) #convert matrix to vector
claim_sizes = claim_sizes[!is.nan(claim_sizes)] #remove nan values
mat_new=mat[mat$claim_counts>0,] #remove all rows where claim numbers is zero

#insert new line for policy with more than one claim
j=1 #new variable
for (i in 1:mat$claim_counts){ #i is size of claim numbers for policy i
  if (i > 1){ #if test whenever claim numbers larger than one
    for (a in 1:(i-1)) { #only when claim numbers larger than one
      #insert new line for same policy
      mat_new = rbind(mat_new[1:j,],mat_new[j,],mat_new[-(1:j),])
    }
    j = j+i #plus i lines of same policy
  } else{
    j = j+1 #jump to next policy
  }
}
mat_new$claim_counts = claim_sizes #don't need claim numbers
colnames(mat_new)[18]="claim_sizes" #change column name in mat_new
mat_new=data.frame(mat_new) #make data frame

#GLMs and GLMMs for claim sizes
mod_g = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=Gamma(log))
mod_gr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat_new,family=Gamma(log),nAGQ=5,control=glmerControl(
  optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
mod_ig = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=inverse.gaussian(log))
mod_igr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id),data=mat_new,family=inverse.gaussian(log),nAGQ=5,control= glmerControl( optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
B. R-CODE

```r
# R-code for simulating claim counts and premium calculations

glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

var_pr=VarCorr(mod_pr); var_nbr=VarCorr(mod_nbr)
var_gr=VarCorr(mod_gr); var_igr=VarCorr(mod_igr)
var_sim = cbind(var_pr,var_nbr,var_gr,var_igr) #estimated var. random effects
var_sim = as.numeric(var_sim) #numerical vector

#return cov, parameters, true premium, estimated variance of random effects
count_sim=c(coef(mod_p),coef(mod_nb),fixef(mod_pr),fixef(mod_nbr))
size_sim=c(coef(mod_g),coef(mod_ig),fixef(mod_gr),fixef(mod_igr))
pi_true = exp(cov16%*%beta_n+v_n^2/2)*exp(cov16%*%beta_z+v_z^2/2) #true premium

return(list(cov16=cov16,beta_sim=c(count_sim,size_sim), pi_true=pi_true,
var_sim=var_sim))

#################################################
#### function 3 (negbin+gamma+random part 1) ####
#################################################

claims_nbg=function(){

##J, id, years, tau, k_gamma from program
#exposure t_i = 1 for all policies for simplicity

cov16=cov() #draw covariates

#simulate claim counts
r = rep(rnorm(J/years,0,v_n),years) #same effect for same policy over 5 years
lambda = exp(cov16%*%beta_n + r) #mean value for claim counts with log-link
claim_counts = rnegbin(J,lambda,1/tau) #true model for claim counts
mat = cbind(id,cov16,claim_counts) #matrix: id, covariates, claim counts
mat = data.frame(mat) #make data frame

#GLMs and GLMMs for claim counts
mod_p = glm(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat,family=poisson(log))

mod_pr = glmer(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+
V17+(1|id),data=mat,family=poisson(log),nAGQ=5,control=glmerControl(
optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_nb = glm.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+
V17,data=mat,link=log)

mod_nbr = glmer.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+
V17+(1|id),data=mat,link=log,nAGQ=5,control=glmerControl(
optimizer="bobyqa",optCtrl=list(maxfun=10e5)))
```

58
# simulate claim sizes
\[ r = \text{rep}(\text{rnorm}(J/\text{years}, 0, v_z), \text{years}) \] # same effect for same policy over 5 years
\[ \mu = \exp(\text{cov16} \times \beta_z + r) \] # mean value for claim sizes with log-link

claim_sizes = matrix(NaN, J, max(claim_counts)) # row=J, col=max number of claims

# simulate gamma claim sizes for each policy N_ij
for (i in 1:J) {
  if (claim_counts[i] > 0) {
    \[ z_i = \frac{\mu[i] \times \text{rgamma}(\text{claim_counts}[i], k_{\text{gamma}})}{k_{\text{gamma}}} \] # 1 claim = 1 size
    claim_sizes[i, 1:length(z_i)] = z_i # insert z_i in matrix claim_sizes
  }
}

claim_sizes = as.vector(t(claim_sizes)) # convert matrix to vector
claim_sizes = claim_sizes[!is.na(claim_sizes)] # remove nan values
mat_new = mat[mat$claim_counts > 0, ] # remove all rows where claim numbers is zero

# insert new line for policy with more than one claim
j = 1 # new variable
for (i in mat_new$claim_counts) {
  if (i > 1) {
    \[ j = j + i \] # plus i lines of same policy
    mat_new = rbind(mat_new[1:j, ], mat_new[j, ], mat_new[-(1:j), ])
  } else {
    j = j + 1 # jump to next policy
  }
}

mat_new$claim_counts = claim_sizes # don't need claim numbers
colnames(mat_new)[18] = "claim_sizes" # change column name in mat_new
mat_new = data.frame(mat_new) # make data frame

# GLMs and GLMMs for claim sizes
mod_g = glm(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17, data = mat_new, family = Gamma(log))

mod_gr = glmer(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1 | id), data = mat_new, family = Gamma(log), nAGQ = 5, control = glmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 10e5)))

mod_ig = glm(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17, data = mat_new, family = inverse.gaussian(log))

mod_igr = glmer(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1 | id), data = mat_new, family = inverse.gaussian(log))
B. R-CODE

```r
V17+(1|id), data = mat_new, family = inverse.gaussian(log), nAGQ = 5, control = glmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 1e5)))

var_pr = VarCorr(mod_pr); var_nbr = VarCorr(mod_nbr)
var_gr = VarCorr(mod_gr); var_igr = VarCorr(mod_igr)
var_sim = cbind(var_pr, var_nbr, var_gr, var_igr)  # estimated var. random effects
var_sim = as.numeric(var_sim)  # numerical vector

# return cov, parameters, true premium, estimated variance of random effects
count_sim = c(coef(mod_p), coef(mod_nb), fixef(mod_pr), fixef(mod_nbr))
size_sim = c(coef(mod_g), coef(mod_ig), fixef(mod_gr), fixef(mod_igr))
pi_true = exp(cov16%*%beta_n + v_n^2/2) * exp(cov16%*%beta_z + v_z^2/2)  # true premium

return(list(cov16 = cov16, beta_sim = c(count_sim, size_sim), pi_true = pi_true, var_sim = var_sim))
```}

```
# function 4 (negbin+inverse gaussian part 1)
claims_nbig=function(){
  # J, id, years, tau, lambda_ig from program
  # exposure t_i = 1 for all policies for simplicity.

  cov16 = cov()  # draw covariates

  # simulate claim counts
  r = rep(rnorm(J/years, 0, v_n), years)  # same effect for same policy over 5 years
  lambda = exp(cov16%*%beta_n + r)  # mean value for claim counts with log-links
  claim_counts = rnegbin(J, lambda, 1/tau)  # true model for claim counts
  mat = cbind(id, cov16, claim_counts)  # matrix: id, covariates, claim counts
  mat = data.frame(mat)  # make data frame

  # GLMs and GLMMs for claim counts
  mod_p = glm(claim_counts ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17,
              data = mat, family = poisson(log))

  mod_pr = glmer(claim_counts ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1|id),
                 data = mat, family = poisson(log), nAGQ = 5, control = glmerControl(
                 optimizer = "bobyqa", optCtrl = list(maxfun = 1e5)))

  mod_nb = glm.nb(claim_counts ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17,
                  data = mat, link = log)

  mod_nbr = glmer.nb(claim_counts ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1|id),
                     data = mat, link = log, nAGQ = 5, control = glmerControl(
```
#simulate claim sizes
\[ r = \text{rep}(\text{rnorm}(J/\text{years}, 0, v_z), \text{years}) \] #same effect for same policy over 5 years
\[ \mu = \exp(\text{cov16} %*% \beta_z + r) \] #mean value for claim sizes with log-link
\[ \text{claim Sizes} = \text{matrix}(\text{NaN}, J, \text{max(claim_counts)}) \] #row=J, col=max number of claims

#simulate inverse gaussian claim sizes for each policy \( N_{ij} \)

\[
\text{for}(i \in 1:J)\{ \\
\text{if}(\text{claim_counts}[i] > 0) \{ #\text{when claim_counts}[i] = 0, \text{go to next loop} \\
\text{z}_i = \text{rinvgauss(claim_counts[i], mu[i], shape=lambda_ig)} #1 \text{ claim} = 1 \text{ size} \\
\text{claim_sizes}[i,1:length(z_i)] = z_i \} \text{ #insert z_i in matrix claim_sizes} \\
\}
\]
\[ \text{claim_sizes} = \text{as.vector(t(claim_sizes))} \] #convert matrix to vector
\[ \text{claim_sizes} = \text{claim_sizes}[[\text{is.nan(claim_sizes)}]] \] #remove nan values
\[ \text{mat_new=mat[mat$claim_counts>0,]} \] #remove all rows where claim counts is zero

#insert new line for policy with more than one claim
\[ j=1 \] #new variable
\[
\text{for} \ (i \ \text{in} \ \text{mat_new$claim_counts}) \{ \ #i \text{ is size of claim counts for policy i} \\
\text{if} \ (i > 1) \{ \ #\text{if test whenever claim counts larger than one} \\
\text{for} \ (a \ \text{in} \ 1:(i-1)) \{ \ #\text{only when claim counts larger than one} \\
\text{#insert new line for same policy} \\
\text{mat_new} = \text{rbind(mat_new[1:j,], mat_new[j,], mat_new[-(1:j),])} \\
\}
\]
\[
\text{j} = \text{j+i} \ #\text{plus i lines of same policy} \\
\text{else} \\
\text{j} = \text{j+1} \ #\text{jump to next policy} \\
\}
\]
\[ \text{mat_new$claim_counts} = \text{claim_sizes} \] #don’t need claim numbers
\[
\text{colnames(mat_new)[18]="claim_sizes"} \ #\text{change column name in mat_new} \\
\text{mat_new=data.frame(mat_new)} \ #\text{make data frame}
\]

#GLMs and GLMMs for claim sizes
\[ \text{mod_g} = \text{glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,} \]
\[ \text{data=mat_new,family=Gamma(log))} \]
\[ \text{mod_gr} = \text{glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+} \]
\[ \text{V17+(1|id),data=mat_new,family=Gamma(log),nAGQ=5,control=glmerControl(} \]
\[ \text{optimizer="bobyqa",optCtrl=list(maxfun=10e5))} \]
\[ \text{mod_ig} = \text{glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,} \]
\[ \text{data=mat_new,family=inverse.gaussian(log))} \]
\[ \text{mod_igr} = \text{glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+} \]
B. R-CODE

V17+(1|id),data=mat_new,family=inverse.gaussian(log),nAGQ=5,control=
glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

var_pr=VarCorr(mod_pr); var_nbr=VarCorr(mod_nbr)
var_gr=VarCorr(mod_gr); var_igr=VarCorr(mod_igr)
var_sim = cbind(var_pr,var_nbr,var_gr,var_igr) # estimated var. random effects
var_sim = as.numeric(var_sim) # numerical vector

# return cov, parameters, true premium, estimated variance of random effects
count_sim=c(coef(mod_p),coef(mod_nb),fixef(mod_pr),fixef(mod_nbr))
size_sim=c(coef(mod_g),coef(mod_ig),fixef(mod_gr),fixef(mod_igr))
pi_true = exp(cov16%*%beta_n+v_n^2/2)*exp(cov16%*%beta_z+v_z^2/2) # true premium

return(list(cov16=cov16,beta_sim=c(count_sim,size_sim), pi_true=pi_true,
var_sim=var_sim))

}  

####################################
#### function (premium part 1) #####
####################################

premium = function(cov16,beta_sim,pi_true,var_r.e){

beta_hat_p = beta_sim[,1:16] # row 1, col 1:16
beta_hat_nb = beta_sim[,17:32] # row 2, col 17:32
beta_hat_pr = beta_sim[,33:48]
beta_hat_nbr = beta_sim[,49:64]
beta_hat_g = beta_sim[,65:80]
beta_hat_ig = beta_sim[,81:96]
beta_hat_gr = beta_sim[,97:112]
beta_hat_igr = beta_sim[,113:128]

lambdaHat_p=lambdaHat_nb=lambdaHat_pr=lambdaHat_nbr = matrix(NaN,J,sim)
muHat_g=muHat_ig=muHat_gr=muHat_igr = matrix(NaN,J,sim)

var_r.e_pr=var_r.e[,1]; var_r.e_nbr=var_r.e[,2]
var_r.e_gr=var_r.e[,3]; var_r.e_igr=var_r.e[,4]

# Expected premium for GLMs and GLMMs
j=1
for (i in 1:sim){
  mat=cov16[j:(i*J),]
  lambdaHat_p[,i] = exp(mat%*%beta_hat_p[,i]) # dim(row=J, col=sim)
  lambdaHat_nb[,i] = exp(mat%*%beta_hat_nb[,i])
  lambdaHat_pr[,i] = exp(mat%*%beta_hat_pr[,i] + (var_r.e_pr[i]/2 )
  lambdaHat_nbr[,i] = exp(mat%*%beta_hat_nbr[,i] + (var_r.e_nbr[i]/2 )
  muHat_g[,i] = exp(mat%*%beta_hat_g[,i])
  muHat_ig[,i] = exp(mat%*%beta_hat_ig[,i])
}

62
B. R-CODE

```r
def beta_hat_gr[i], beta_hat_igr[i] = \exp(\text{mat} \times \beta_{\text{hat-gr}}[i] + \frac{\text{var_r.e.gr}[i]}{2})

def beta_hat_igr[i] = \exp(\text{mat} \times \beta_{\text{hat-igr}}[i] + \frac{\text{var_r.e.igr}[i]}{2})

j = j + J
```

```r
#bias
diff_pi_pg = colMeans(lambda_hat_p * mu_hat_g - pi_true)
diff_pi_pig = colMeans(lambda_hat_p * mu_hat_ig - pi_true)
diff_pi_nbg = colMeans(lambda_hat_nb * mu_hat_g - pi_true)
diff_pi_nbig = colMeans(lambda_hat_nb * mu_hat_ig - pi_true)
diff_pi_pgr = colMeans(lambda_hat_pr * mu_hat_gr - pi_true)
diff_pi_pigr = colMeans(lambda_hat_pr * mu_hat_igr - pi_true)
diff_pi_nbgr = colMeans(lambda_hat_nbr * mu_hat_gr - pi_true)
diff_pi_nbigr = colMeans(lambda_hat_nbr * mu_hat_igr - pi_true)

#root-mean-squared error
rmse_pi_pg = sqrt(colMeans((lambda_hat_p * mu_hat_g - pi_true)^2))
rmse_pi_pig = sqrt(colMeans((lambda_hat_p * mu_hat_ig - pi_true)^2))
rmse_pi_nbg = sqrt(colMeans((lambda_hat_nb * mu_hat_g - pi_true)^2))
rmse_pi_nbig = sqrt(colMeans((lambda_hat_nb * mu_hat_ig - pi_true)^2))
rmse_pi_pgr = sqrt(colMeans((lambda_hat_pr * mu_hat_gr - pi_true)^2))
rmse_pi_pigr = sqrt(colMeans((lambda_hat_pr * mu_hat_igr - pi_true)^2))
rmse_pi_nbgr = sqrt(colMeans((lambda_hat_nbr * mu_hat_gr - pi_true)^2))
rmse_pi_nbigr = sqrt(colMeans((lambda_hat_nbr * mu_hat_igr - pi_true)^2))

#mean absolute error
mae_pi_pg = colMeans(abs(lambda_hat_p * mu_hat_g - pi_true))
mae_pi_pig = colMeans(abs(lambda_hat_p * mu_hat_ig - pi_true))
mae_pi_nbg = colMeans(abs(lambda_hat_nb * mu_hat_g - pi_true))
mae_pi_nbig = colMeans(abs(lambda_hat_nb * mu_hat_ig - pi_true))
mae_pi_pgr = colMeans(abs(lambda_hat_pr * mu_hat_gr - pi_true))
mae_pi_pigr = colMeans(abs(lambda_hat_pr * mu_hat_igr - pi_true))
mae_pi_nbgr = colMeans(abs(lambda_hat_nbr * mu_hat_gr - pi_true))
mae_pi_nbigr = colMeans(abs(lambda_hat_nbr * mu_hat_igr - pi_true))

diff = cbind(diff_pi_pg, diff_pi_pig, diff_pi_nbg, diff_pi_nbig,
diff_pi_pgr, diff_pi_pigr, diff_pi_nbgr, diff_pi_nbigr)
rmse = cbind(rmse_pi_pg, rmse_pi_pig, rmse_pi_nbg, rmse_pi_nbig,
rmse_pi_pgr, rmse_pi_pigr, rmse_pi_nbgr, rmse_pi_nbigr)
mae = cbind(mae_pi_pg, mae_pi_pig, mae_pi_nbg, mae_pi_nbig,
mae_pi_pgr, mae_pi_pigr, mae_pi_nbgr, mae_pi_nbigr)

return(list(diff=diff, rmse=rmse, mae=mae))
```
#### function (write to file part 1) ####

```r
write_to_file = function(){
sink('output.txt', append=TRUE, split=TRUE)
cat('================================================================
')
cat(sprintf("v_n = %0.1f and v_z = %0.1f \n", v_n, v_z) )
cat("================================================================
")
cat("pg diff",pg_mean_diff,"\n")
cat("pg rmse",pg_mean_rmse,"\n")
cat("pg mae ",pg_mean_mae, "\n")
cat("pg std ",pg_mean_std, "\n")
cat("\n")
cat("pig diff",pig_mean_diff,"\n")
cat("pig rmse",pig_mean_rmse,"\n")
cat("pig mae ",pig_mean_mae, "\n")
cat("pig std ",pig_mean_std, "\n")
cat("\n")
cat("nbg diff",nbg_mean_diff,"\n")
cat("nbg rmse",nbg_mean_rmse,"\n")
cat("nbg mae ",nbg_mean_mae, "\n")
cat("nbg std ",nbg_mean_std, "\n")
cat("\n")
cat("nbig diff",nbig_mean_diff,"\n")
cat("nbig rmse",nbig_mean_rmse,"\n")
cat("nbig mae ",nbig_mean_mae, "\n")
cat("nbig std ",nbig_mean_std, "\n")
cat("\n")
sink()
}
```

#### (main program part 1) ####

```r
#----------------------
# (main program part 1)
#----------------------

beta_pg = matrix(NaN,sim,(8*16)) #estimates of beta: 8 models*16 beta
cov_pg = matrix(NaN,sim*J,16) #covariates from different simulations
pi_true_pg = matrix(NaN,J,sim) #pure premium vector
variance_pg = matrix(NaN,sim,4) #est. var. from GLMMs, need for premium
j=1
error_pg = 0
for(i in 1:sim){
  repeat{ #repeat simulation if error occurred
    func=try(claims_pg(), silent = TRUE)
    if(is.character(func)){ #if error occurred, "error in func="character")
      error_pg = error_pg + 1 #counting number of error occured
    }
  }
}
```
B. R-CODE

```r
print("error occurred")
} else { # when func returned a list (no error occurred)
    print(i)
    beta_pg[i,] = func$beta_sim
    cov_pg[j:(i*J),] = func$cov16
    pi_true_pg[,i] = func$pi_true
    variance_pg[i,] = func$var_sim
    j=j+J
    break #break the repeat
}
}
prem_pg = premium(cov_pg, beta_pg, pi_true_pg, variance_pg)

pg_mean_diff = colMeans(prem_pg$diff) * 1e4
pg_mean_rmse = colMeans(prem_pg$rmse) * 1e4
pg_mean_mae = colMeans(prem_pg$mae) * 1e4
pg_mean_std = colMeans(sqrt(variance_pg))

beta_pig = matrix(NaN, sim, (8*16)) # estimates of beta: 8 models*16 beta
cov_pig = matrix(NaN, sim*J, 16) # covariates from different simulations
pi_true_pig = matrix(NaN, J, sim) # pure premium vector
variance_pig = matrix(NaN, sim, 4) # est. var. from GLMMs, need for premium
j=1
error_pig = 0
for(i in 1:sim){
    repeat{ # repeat simulation if error occurred
        func=try(claims_pig(), silent = TRUE)
        if(is.character(func)){ # if error occurred, "error in func='character"
            error_pig = error_pig + 1 # counting number of error occurred
            print("error occurred")
        } else { # when func returned a list (no error occurred)
            print(i)
            beta_pig[i,] = func$beta_sim
            cov_pig[j:(i*J),] = func$cov16
            pi_true_pig[,i] = func$pi_true
            variance_pig[i,] = func$var_sim
            j=j+J
            break #break the repeat
        }
    }
    prem_pig = premium(cov_pig, beta_pig, pi_true_pig, variance_pig)

    pig_mean_diff = colMeans(prem_pig$diff) * 1e4
    pig_mean_rmse = colMeans(prem_pig$rmse) * 1e4
    pig_mean_mae = colMeans(prem_pig$mae) * 1e4
}```
pig_mean_std = colMeans(sqrt(variance_pig))

beta_nbg = matrix(NaN,sim,(8*16)) #estimates of beta: 8 models*16 beta
cov_nbg = matrix(NaN,sim*J,16) #covariates from different simulations
pi_true_nbg = matrix(NaN,J,sim) #pure premium vector
variance_nbg = matrix(NaN,sim,4) #est. var. from GLMMs, need for premium
j=1
error_nbg = 0
for(i in 1:sim){
    repeat{ #repeat simulation if error occured
        func=try(claims_nbg(), silent = TRUE)
        if(is.character(func)){ #if error occured, "error in func="character"
            error_nbg = error_nbg + 1 #counting number of error occured
            print("error occurred")
        }else{ # when func returned a list (no error occurred)
            print(i)
            beta_nbg[i,] = func$beta_sim
            cov_nbg[j:(i*J),] = func$cov16
            pi_true_nbg[,i] = func$pi_true
            variance_nbg[i,] = func$var_sim
            j=j+J
            break #break the repeat
        }
    }
}
prem_nbg = premium(cov_nbg,beta_nbg,pi_true_nbg,variance_nbg)

nbg_mean_diff = colMeans(prem_nbg$diff)*1e4
nbg_mean_rmse = colMeans(prem_nbg$rmse)*1e4
nbg_mean_mae = colMeans(prem_nbg$mae) *1e4
nbg_mean_std = colMeans(sqrt(variance_nbg))

beta_nbig = matrix(NaN,sim,(8*16)) #estimates of beta: 8 models*16 beta
cov_nbig = matrix(NaN,sim*J,16) #covariates from different simulations
pi_true_nbig = matrix(NaN,J,sim) #pure premium vector
variance_nbig = matrix(NaN,sim,4) #est. var. from GLMMs, need for premium
j=1
error_nbig = 0
for(i in 1:sim){
    repeat{ #repeat simulation if error occured
        func=try(claims_nbig(), silent = TRUE)
        if(is.character(func)){ #if error occured, "error in func="character"
            error_nbig = error_nbig + 1 #counting number of error occured
            print("error occurred")
        }else{ # when func returned a list (no error occurred)
            print(i)
            beta_nbig[i,] = func$beta_sim
        }
    }
}
B. R-CODE

```r
cov_nbig[j:(i*J),] = func$cov16
pi_true_nbig[,i] = func$pi_true
variance_nbig[i,] = func$var_sim
j=j+J
break #break the repeat
}
}
prem_nbig = premium(cov_nbig,beta_nbig,pi_true_nbig,variance_nbig)

nbig_mean_diff = colMeans(prem_nbig$diff)*1e4
nbig_mean_rmse = colMeans(prem_nbig$rmse)*1e4
nbig_mean_mae = colMeans(prem_nbig$mae) *1e4
nbig_mean_std = colMeans(sqrt(variance_nbig))

write_to_file()
```

B.2 Computer program for Part 2 of the simulation study

# uncorrelated random effects: (x1+x2||id) = (1|id)+(0+x1|id)+(0+x2|id)

#-----------------------------
# function 1 (poisson+gamma part 2) #
#-----------------------------

```r
claims_pg=function(){
#J, id, years, k_gamma from program
#exposure t_i = 1 for all policies for simplicity

cov16=cov() #draw covariates

r0 = rep(rnorm(J/years,0,v_n),years) #first random effect for counts
r1 = rep(rnorm(J/years,0,v_n),years) #second random effect for counts

#covariates Z = (z0,z1) = (x0,x1) Z is a subset of X
z0 = cov16[,1]
z1 = cov16[,2]

lambda = exp(cov16%*%beta_n + z0*r0 + z1*r1 ) #mean value for claim counts
claim_counts = rpois(J,lambda) #true model for claim counts
mat = data.frame(cbind(id,cov16,claim_counts)) #matrix: id, cov, claim counts

#GLMs and GLMMs for claim counts
mod_p = glm(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat,family=poisson(log))
```
mod_pr = glmer(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id)+(0+x1|id),data=mat,family=poisson(log),control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_nb = glm.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,data=mat,link=log)

mod_nbr = glmer.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id)+(0+x1|id),data=mat,link=log,control=glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

r0 = rep(rnorm(J/years,0,v_z),years)
r1 = rep(rnorm(J/years,0,v_z),years)

mu = exp(cov16%*%beta_z + z0*r0 + z1*r1) #mean value for claim sizes

claim_sizes = matrix(NaN,J,max(claim_counts)) #row=J, col=max number of claims

#simulate gamma claim sizes for each policy N_ij
for(i in 1:J){
  if(claim_counts[i] >0){ #when claim_counts[i] = 0, go to next loop
    z_i = mu[i]*rgamma(claim_counts[i],k_gamma)/k_gamma
    claim_sizes[i,1:length(z_i)] = z_i #insert z_i in matrix
  }
}

claim_sizes=as.vector(t(claim_sizes)) #convert matrix to vector
claim_sizes = claim_sizes[!is.nan(claim_sizes)]

mat_new=mat[mat$claim_counts>0,] #remove all rows where claim numbers is zero

#insert new line for policy with more than one claim
j=1 #new variable
for (i in mat_new$claim_counts){ #i is size of claim numbers for policy i
  if (i > 1){ #if test whenever claim numbers larger than one
    for (a in 1:(i-1)){ #only when claim numbers larger than one
      mat_new = rbind(mat_new[1:j,],mat_new[j,],mat_new[-(1:j),])
    }
    j = j+i #plus i lines of same policy
  } else{
    j = j+1 #jump to next policy
  }
}

mat_new$claim_counts = claim_sizes
colnames(mat_new)[18]="claim_sizes"
mat_new=data.frame(mat_new)

#GLMs and GLMMs for claim sizes
mod_g = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
B. R-CODE

data = mat_new, family = Gamma(log))

mod_gr = glmer(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1 | id) + (0 + x1 | id), data = mat_new, family = Gamma(log), control = glmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 10e5)))

mod_ig = glm(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17, data = mat_new, family = inverse.gaussian(log))

mod_igr = glmer(claim_sizes ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + V12 + V13 + V14 + V15 + V16 + V17 + (1 | id) + (0 + x1 | id), data = mat_new, family = inverse.gaussian(log), control = glmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 10e5)))

v_pr = c(VarCorr(mod_pr)[1], VarCorr(mod_pr)[2])

v_nbr = c(VarCorr(mod_nbr)[1], VarCorr(mod_nbr)[2])

v_gr = c(VarCorr(mod_gr)[1], VarCorr(mod_gr)[2])

v_igr = c(VarCorr(mod_igr)[1], VarCorr(mod_igr)[2])

var_sim = cbind(v_pr, v_nbr, v_gr, v_igr)

var_sim = as.numeric(var_sim) # make the list to be a numeric vector

# return estimate, true premium and variance of the random effects

count_sim = c(coef(mod_p), coef(mod_nb), fixef(mod_pr), fixef(mod_nbr))

size_sim = c(coef(mod_g), coef(mod_ig), fixef(mod_gr), fixef(mod_igr))

lambda_true = exp( cov16 %*% beta_n + (z0^2 * v_n^2/2) + (z1^2 * v_z^2/2) )

mu_true = exp( cov16 %*% beta_z + (z0^2 * v_z^2/2) + (z1^2 * v_z^2/2) )

pi_true = lambda_true * mu_true # true pure premium

return(list(cov16 = cov16, beta_sim = c(count_sim, size_sim), pi_true = pi_true, var_sim = var_sim))

}
z1 = cov16[,2]

lambda = exp(cov16%*%beta_n + z0*r0 + z1*r1 ) #mean value for claim counts

claim_counts = rpois(J,lambda) #true model for claim counts

mat = data.frame(cbind(id,cov16,claim_counts))

#GLMs and GLMMs for claim counts
mod_p = glm(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17, 
data=mat,family=poisson(log))

mod_pr = glmer(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+ 
V17+(1|id)+(0+x1|id),data=mat,family=poisson(log),control=glmerControl( 
optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_nb = glm.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+ 
V17, data=mat,link=log)

mod_nbr = glmer.nb(claim_counts~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+ 
V16+V17+(1|id)+(0+x1|id),data=mat,link=log,control=glmerControl( 
optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

r0 = rep(rnorm(J/years,0,v_z),years)
r1 = rep(rnorm(J/years,0,v_z),years)

mu = exp(cov16%*%beta_z + z0*r0 + z1*r1 ) #mean value for claim sizes

claim_sizes =matrix(NaN,J,max(claim_counts))

#simulate gamma claim sizes for each policy N_ij
for(i in 1:J){
if(claim_counts[i] >0){ #when claim_counts[i] = 0, go to next loop
z_i = rinvgauss(claim_counts[i],mu[i],shape=lambda_ig)
claim_sizes[i,1:length(z_i)] = z_i #insert z_i in matrix
}
}

claim_sizes=as.vector(t(claim_sizes)) #convert matrix to vector
claim_sizes = claim_sizes[!is.nan(claim_sizes)]

mat_new=mat[mat$claim_counts>0,] #remove all rows where claim numbers is zero

#insert new line for policy with more than one claim
j=1 #new variable
for (i in mat_new$claim_counts){ #i is size of claim numbers for policy i
if (i >1){ #if test whenever claim numbers larger than one
for (a in 1:(i-1)) { #only when claim numbers larger than one
mat_new = rbind(mat_new[1:j,],mat_new[j,],mat_new[-(1:j),])
}
j = j+i# #plus i lines of same policy
} else{
B. R-CODE

j = j+1 #jump to next policy
}
}
mat_new$claim_counts = claim_sizes
colnames(mat_new)[18]="claim_sizes"
mat_new=data.frame(mat_new)

#GLMs and GLMMs for claim sizes
mod_g = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=Gamma(log))

mod_gr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id)+(0+x1|id),data=mat_new,family=Gamma(log),control=glmerControl(
  optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

mod_ig = glm(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17,
data=mat_new,family=inverse.gaussian(log))

mod_igr = glmer(claim_sizes~x1+x2+x3+x4+x5+x6+x7+x8+x9+V12+V13+V14+V15+V16+V17+(1|id)+(0+x1|id),data=mat_new,family=inverse.gaussian(log),control=
  glmerControl(optimizer="bobyqa",optCtrl=list(maxfun=10e5)))

v_pr = c( VarCorr(mod_pr)[1], VarCorr(mod_pr)[2])
v_nbr= c( VarCorr(mod_nbr)[1],VarCorr(mod_nbr)[2])
v_gr = c( VarCorr(mod_gr)[1], VarCorr(mod_gr)[2])
v_igr= c( VarCorr(mod_igr)[1],VarCorr(mod_igr)[2])

var_sim = cbind(v_pr,v_nbr,v_gr,v_igr)
var_sim = as.numeric(var_sim)

#return estimate,true premium and variance of the random effects
count_sim=c(coef(mod_p),coef(mod_nb),fixef(mod_pr),fixef(mod_nbr))
size_sim=c(coef(mod_g),coef(mod_ig),fixef(mod_gr),fixef(mod_igr))

lambda_true = exp( cov16%*%beta_n + (z0^2*v_n^2/2) + (z1^2*v_n^2/2) )
mu_true = exp( cov16%*%beta_z + (z0^2*v_z^2/2) + (z1^2*v_z^2/2) )
pi_true = lambda_true*mu_true #true pure premium

return(list(cov16=cov16,beta_sim=c(count_sim,size_sim),pi_true=pi_true,
  var_sim=var_sim))
}

####################################
#### function (premium part 2) #####
####################################

premium = function(cov16,beta_sim,pi_true,var_r.e){


B. R-CODE

```
beta_hat_p = beta_sim[,1:16]  #row 1, col 1:16
beta_hat_nb = beta_sim[,17:32]  #row 2, col 17:32
beta_hat_pr = beta_sim[,33:48]
beta_hat_g = beta_sim[,65:80]
beta_hat_ig = beta_sim[,81:96]
beta_hat_gr = beta_sim[,97:112]
beta_hat_igr = beta_sim[,]113:128

lambda_hat_p=lambda_hat_nb=lambda_hat_pr=lambda_hat_nbr=matrix(NaN,J,sim)
mu_hat_g = mu_hat_ig = mu_hat_gr = mu_hat_igr = matrix(NaN,J,sim)

var_r.e_pr = cbind(var_r.e[,1], var_r.e[,2]);
var_r.e_nbr = cbind(var_r.e[,3], var_r.e[,4]);
var_r.e_gr = cbind(var_r.e[,5], var_r.e[,6]);
var_r.e_igr = cbind(var_r.e[,7], var_r.e[,8]);

#Expected premium for GLMs and GLMMs
j=1
i = 1
for (i in 1:sim){
  mat=cov16[j:(i*J),]
  z0 = mat[,1]
  z1 = mat[,2]
  lambda_hat_p[,i] = exp(mat%*%beta_hat_p[i,])  #dim(row=J, col=sim)
  lambda_hat_nb[,i] = exp(mat%*%beta_hat_nb[i,])
  lambda_hat_pr[,i] = exp(mat%*%beta_hat_pr[i,] + (z0^2*var_r.e_pr[i,1]/2) +
                        (z1^2*var_r.e_pr[i,2]/2))
  lambda_hat_nbr[,i]= exp(mat%*%beta_hat_nbr[i,] + (z0^2*var_r.e_nbr[i,1]/2) +
                        (z1^2*var_r.e_nbr[i,2]/2))
  lambda_hat_g[,i] = exp(mat%*%beta_hat_g[i,])
  lambda_hat_ig[,i] = exp(mat%*%beta_hat_ig[i,])
  lambda_hat_gr[,i] = exp(mat%*%beta_hat_gr[i,] + (z0^2*var_r.e_gr[i,1]/2) +
                        (z1^2*var_r.e_gr[i,2]/2))
  lambda_hat_igr[,i] = exp(mat%*%beta_hat_igr[i,] + (z0^2*var_r.e_igr[i,1]/2) +
                        (z1^2*var_r.e_igr[i,2]/2))
  j = j+J
}

#bias
diff_pi_pg= colMeans(lambda_hat_p*mu_hat_g - pi_true)
diff_pi_pig= colMeans(lambda_hat_p*mu_hat_ig - pi_true)
diff_pi_nb= colMeans(lambda_hat_nb*mu_hat_g - pi_true)
diff_pi_nbig= colMeans(lambda_hat_nb*mu_hat_ig - pi_true)
```

72
B. R-CODE

diff_pi_pgr = colMeans(lambda_hat_pr*mu_hat_gr - pi_true)
diff_pi_pigr = colMeans(lambda_hat_pr*mu_hat_igr - pi_true)
diff_pi_nbgr = colMeans(lambda_hat_nbr*mu_hat_gr - pi_true)
diff_pi_nbigr = colMeans(lambda_hat_nbr*mu_hat_igr - pi_true)

#root-mean-squared error
rmse_pi_pg = sqrt(colMeans((lambda_hat_p*mu_hat_g - pi_true)^2 ))
rmse_pi_pig = sqrt(colMeans((lambda_hat_p*mu_hat_ig - pi_true)^2 ))
rmse_pi_nbg = sqrt(colMeans((lambda_hat_nb*mu_hat_g - pi_true)^2 ))
rmse_pi_nbig = sqrt(colMeans((lambda_hat_nb*mu_hat_ig - pi_true)^2 ))
rmse_pi_pgr = sqrt(colMeans((lambda_hat_pr*mu_hat_gr - pi_true)^2 ))
rmse_pi_pigr = sqrt(colMeans((lambda_hat_pr*mu_hat_igr - pi_true)^2 ))
rmse_pi_nbgr = sqrt(colMeans((lambda_hat_nbr*mu_hat_gr - pi_true)^2 ))
rmse_pi_nbigr = sqrt(colMeans((lambda_hat_nbr*mu_hat_igr - pi_true)^2 ))

diff = cbind(diff_pi_pg, diff_pi_pig, diff_pi_nbg, diff_pi_nbig,
diff_pi_pgr, diff_pi_pigr, diff_pi_nbgr, diff_pi_nbigr)
rmse = cbind(rmse_pi_pg, rmse_pi_pig, rmse_pi_nbg, rmse_pi_nbig,
rmse_pi_pgr, rmse_pi_pigr, rmse_pi_nbgr, rmse_pi_nbigr)
mae = cbind(mae_pi_pg, mae_pi_pig, mae_pi_nbg, mae_pi_nbig,
mae_pi_pgr, mae_pi_pigr, mae_pi_nbgr, mae_pi_nbigr)

return(list(diff=diff, rmse=rmse, mae=mae))

###########################################
#### function (write to file part 2) ####
###########################################

write_to_file = function(){
sink('output_part2.txt', append=TRUE, split=TRUE)
cat("=============================\n")
cat(sprintf("kode del 2: v_n = %0.1f and v_z = %0.1f \n", v_n, v_z) )
cat("=============================\n")
cat("pg diff", pg_mean_diff, "\n")
cat("pg rmse", pg_mean_rmse, "\n")
cat("pg mae ", pg_mean_mae, "\n")
cat("pg std ", pg_mean_std, "\n")
}
B. R-CODE

cat("\n")
cat("pig diff", pig_mean_diff, "\n")
cat("pig rmse", pig_mean_rmse, "\n")
cat("pig mae ", pig_mean_mae, "\n")
cat("pig std ", pig_mean_std, "\n")
sink()
}

########################################################################
#### main program part 2 ####
########################################################################
#--------------------------------------------------1
beta_pg = matrix(NaN, sim, (8*16))
cov_pg = matrix(NaN, sim*J, 16)
pi_true_pg = matrix(NaN, J, sim)
variance_pg = matrix(NaN, sim, 2*4)
j=1
error_pg = 0
for(i in 1:sim){
  repeat{ #repeat simulation if error occurred
    func=try(claims_pg(), silent = TRUE)
    if(is.character(func)){ #if error occurred, "error in func = "character"
      error_pg = error_pg + 1 #counting number of error occurred
      print("error occurred")
    }else{ # when func returned a list (no error occurred)
      print(i)
      beta_pg[i,] = func$beta_sim
      cov_pg[j:(i*J),] = func$cov16
      pi_true_pg[,i] = func$pi_true
      variance_pg[i,] = func$var_sim
      j=j+J
      break #break the repeat
    }
  }
}
prem_pg = premium(cov_pg, beta_pg, pi_true_pg, variance_pg)

gg_mean_diff = colMeans(prem_pg$diff)*1e4
gg_mean_rmse = colMeans(prem_pg$rmse)*1e4
gg_mean_mae = colMeans(prem_pg$mae) *1e4
gg_mean_std = colMeans(sqrt(variance_pg))
#--------------------------------------------------2
beta_pig = matrix(NaN, sim, (8*16))
cov_pig = matrix(NaN, sim*J, 16)
pi_true_pig = matrix(NaN, J, sim)
variance_pig = matrix(NaN, sim, 2*4)
j=1
error_pig = 0
for(i in 1:sim){
  repeat{ #repeat simulation if error occurred
    func=try(claims_pig(), silent = TRUE)
    if(is.character(func)){ #if error occurred, "error in func = "character"
      error_pig = error_pig + 1 #counting number of error occurred
      print("error occurred")
    }else{ # when func returned a list (no error occurred)
      print(i)
      beta_pig[i,] = func$beta_sim
      cov_pig[j:(i*J),] = func$cov16
      pi_true_pig[,i] = func$pi_true
      variance_pig[i,] = func$var_sim
      j=j+J
      break #break the repeat
    }
  }
}
pig_mean_diff = colMeans(prem_pig$diff)*1e4
pig_mean_rmse = colMeans(prem_pig$rmse)*1e4
pig_mean_mae = colMeans(prem_pig$mae)*1e4
pig_mean_std = colMeans(sqrt(variance_pig))
write_to_file()}