UNIVERSITY OF AUCKLAND

Vector Generalized Linear Models for Zipf-Mandelbrot Distributions

by

Mu-Jou (Miriam) Chou

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Science

Department of Statistics

February 2020

Declaration of Authorship

I, Mu-Jou (Miriam) Chou, declare that this thesis titled, 'Vector Generalized Linear Models for Zipf-Mandelbrot Distributions' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: ________________________________

“We have complex, messy models, yet reality is startlingly neat and simple.”

Paul Krugman

UNIVERSITY OF AUCKLAND

Abstract

Faculty of Science
Department of Statistics

Master of Science

by Mu-Jou (Miriam) Chou

The aim of this thesis is to implement an additional Zipf-MandelbrotII distribution (Mandelbrot 1961), as a piece of software, on top of the existing Vector Generalized Linear and Additive Models (VGLMs/VGAMs) framework. The implemented model follows Zipf's law, and is particularly suitable for word frequency analysis, taking into account of the occurring frequency and occurring rank of words in a corpus. Through the VGAM R package, one can develop the Maximum Likelihood Estimate (MLE) of the scaling parameter on empirical data sets by passing in the implemented model. Along with the model, associate `dpqr`-type functions are also implemented. All functions are used and validated using real world data sets from the Gutenberg Database. Upon the completion of this work, a greater flexibility on analysing linguistics data is provided.

Acknowledgements

I would like to express my deepest appreciation to my mentor, Dr. Thomas William Yee, over the past year. He is witty and talented, with the most devoted heart in teaching and crafting the beautiful VGAM. His meticulous guidance in research and academics is also impeccable. Having been a student of Thomas is an honour, also a challenge for the high standard he has.

I am indebted to my partner, Yin-Han Chung. He has selflessly given me all his time during this journey. Not only I have learnt from him how to style the thesis or debug LaTeX, but also to be gritty and determined. I wish one day I would be as strong-hearted as he is, to support him the same way he has supported me!

My two beloved sisters, Angel Chou and Ariel Chou, are my source of energy. Although they may not have contributed directly to this work—and possibly have no idea what I am writing—it would not have been possible without them. They bring me joy and laughter, every night and day.

Last but foremost, this work is dedicated to my parents. They are the two greatest people in my world, who have lent me wisdom and love. They are my motivation and my refuge. *I love you forever!*

Contents

Declaration of Authorship **i**

Abstract **iii**

Acknowledgements **iv**

List of Figures **viii**

List of Tables **ix**

Acronyms **x**

1 Introduction **1**
- 1.1 Vector Generalized Linear Models 1
- 1.2 The Zipf's Law 2
- 1.3 Goals 3
- 1.4 Outline 4

2 Zipf's Law **5**
- 2.1 Definitions 5
  - 2.1.1 Variant 1 6
  - 2.1.2 Variant 2 7
  - 2.1.3 Variant 3 9
- 2.2 Recent Research 10
- 2.3 Fitting to Empirical Data 12
  - 2.3.1 Parameter estimation 14
- 2.4 Gutenberg Database 15
- 2.5 Hypothesis Testing 16
  - 2.5.1 Goodness-of-fit procedure 16
  - 2.5.2 Likelihood Ratio 17
  - 2.5.3 Likelihood Ratios $L_{1,3}$ and $L_{2,3}$ 18

3 VGLMs **19**
- 3.1 Basics 19

3.1.1 IRLS . . . 20
3.2 The VGAM Package . . . 22
3.2.1 Link Functions . . . 22
3.2.2 The `summary` Generic Function . . . 23
3.3 Examples . . . 24
3.3.1 Example 1: `zetaff()` . . . 24
3.3.2 Example 2: `diffzeta()` . . . 26

4 The New zipfmbrot() Family Function 28
4.1 Overview . . . 28
4.2 Family Function `zipfmbrot()` . . . 30
4.2.1 Log-likelihood . . . 30
4.2.2 Score Vector . . . 31
4.2.3 Working Weight Matrix . . . 31
4.2.3.1 Algorithm 1: `EIM.zipfmb.specilp()` . . . 32
4.2.3.2 Algorithm 2: Simulated Fisher Scoring (SFS) . . . 33
4.2.4 Half Stepping . . . 34
4.3 dpqr-functions . . . 34
4.3.1 Density Function `dzipfmb()` . . . 35
4.3.2 Cumulative Function `pzipfmb()` . . . 36
4.3.3 Inversed CDF `qzipfmb()` . . . 37
4.3.4 Random Deviates `rzipfmb()` . . . 38
4.4 Writing R package . . . 39

5 Uses and Applications 40
5.1 Basic Use of `zipfmbrot()` . . . 40
5.2 Comparisons over 3 Variants . . . 45

6 Conclusions and Future Work 48
6.1 Theoretical Contribution . . . 48
6.2 Practical Application . . . 48
6.3 Limitations and Future Work . . . 49

A Source Code 50
A.1 Density Function `dzipfmb()` . . . 50
A.2 Cumulative Function `pzipfmb()` . . . 51
A.3 Quantile Function `qzipfmb()` . . . 53
A.4 Random Deviates Generator `rzipfmb()` . . . 54
A.5 Family function `zipfmbrot()` . . . 55
A.5.1 `@blurb` . . . 55
A.5.2 `@info` . . . 55
A.5.3 `@linkinv` . . . 56
A.5.4 `@constraints` . . . 56
A.5.5 `@last` . . . 57
A.5.6 `@loglikelihood` . . . 58

A.5.7 `@vfamily` 58
A.5.8 `@deriv` 58
A.5.9 `@weight` 59

B Background Material **60**
B.1 On Zipf and Zeta Distributions 60
B.2 The Riemann and Hurwitz Zeta Functions 61
B.3 The Hurwitz Zeta Function 61

List of Figures

2.1 Zeta Distribution PMF and CDF; Shape = 1 and Shape = 2 7
2.2 Differenced Zeta Distribution PMF and CDF; Shape = 1 and Shape = 2 8
2.3 Mandelbrot Zipf Distribution PMF and CDF; Shape = 1 and Shape = 2 10
2.4 Expression of Zipf's Law on 12 books 13

3.1 Pearson residuals of fit by `zetaff()` 26

5.1 Comparisons of exponent β estimates between three variants over 20 books 47

List of Tables

2.1 Three Variants with PMF, CDF and the parameter β 11
2.2 Word frequency of *The Duke's Children* by Anthony Trollope 15

3.1 VGAM link functions for Zipf's models 23

4.1 Family function slots 29
4.2 internal functions for VGAM link function. 32
4.3 Summary of four `dpqr-zipfmb()` functions 35

5.1 Comparisons of exponent β estimates between three variants over 20 books 47

Acronyms

EIM	**E**xpected **I**nformation **M**atrix
GAM	**G**eneralized **A**dditive **M**odels
GLM	**G**eneralized **L**inear **M**odels
IRLS	**I**teratively **R**eweighted **L**east **S**quares
LR	**L**ikelihood **R**atio
MLE	**M**aximum **L**ikelihood **E**stimate
VGAM	**V**ector **G**eneralized **A**dditive **M**odels
VGLM	**V**ector **G**eneralized **L**inear **M**odels
PDF	**P**robability **D**ensity **F**unction
PMF	**P**robability **M**ass **F**unction
SFS	**S**imulated **F**isher **S**coring
CDF	**C**umulative **D**ensity **F**unction
DF	**D**egrees of **F**reedom

Dedicated to Amos and Alice. . .

Chapter 1

Introduction

In the early 1970s, the *generalized linear model* (GLM) class of statistical models was proposed to the statistical landscape, notably for its unified framework for several frequently used regression models. Nelder & Wedderburn (1972) has shown that the linear model, Poisson regression, logistic regression, probit analysis and several others could be treated as special cases of GLM and with one algorithm, *iteratively reweighted least squares* (IRLS), to estimate them all. Prior to the unification, these methods were treated as unrelated. The GLM framework also provides a theoretical structure encompassing streamlining inference, diagnostics, computation, software interface, etc. Since then GLMs have gained universal acceptance by the statistical community.

1.1 Vector Generalized Linear Models

In spite of the great advantages, GLMs are in large restricted to one-parameter distributions within the classical exponential family. This results in a great limitation in the digital era—information at all levels is experiencing exponential growth and a broad variety of data has become accessible. Standard GLM regression could no longer satisfy the demand of the diverse environment. To expand the usability, a more flexible framework was proposed in the form of *vector generalized linear models* (VGLMs). They are described by Yee (2015). Additional to VGLMs, the class is extended in various directions to offer a suite of variants including *vector generalized additive models* (VGAMs) for smoothing, *reduced-rank VGLMs* (RR-VGLMs) for dimension reduction, and afew other classes. The new framework provides the same benefits that GLMs gave, but on a much larger scale beyond the exponential family. VGLMs can be loosely thought of as a multivariate GLMs and over 150 distributions and regression models have been implemented in the VGAM R package (Yee 2019). This is a large (1.3 MB source code after compression) substantial piece of software developed over the past two decades. With the software as a convenient vehicle, these models could be easily fit by full maximum likelihood estimation through IRLS. VGLMs/VGAMs have been successfully proposed for the analysis of extremes data (Gilleland et al. 2013), categorical responses (Yee 2010) and other areas such as univariate distributions and aspects of quantile regression.

1.2 The Zipf's Law

In terms of other VGLM application areas, one significant and currently undeveloped field is the quantitative linguistics. Written text received active research focus over the past 70 years, for the purpose of understanding the cornerstone of human natural language. The access to huge amounts of digital documents has become a powerful source that can be analysed for signatures of human communication. Statistical patterns in written text can be detected as a trace of our mental processes. The study and understanding of text structure have been extended to a broad range of critical applications such as Web search (Chakrabarti 2002), literature mining (Ananiadou & Mcnaught 2005), topic detection (Allan et al. 1998), and security (Newman et al. 2006). Thus, it is not surprising that researchers in linguistics, information theory and cognitive science, statistics, and complex systems are collaborating together to model how universal text properties emerge.

From the statistics perspective, the frequency distribution of word occurrence has been a key object of study. It is commonly believed that this distribution approximately follows a simple mathematical form known as *Zipf's law*. According to the law, the frequency of words in a piece of text collection decreases inversely to the rank of the words, detailed in Chapter 2. Zipf's law has empirically been found to apply to collections of written documents in virtually all languages (Yu et al. 2018). More fascinatingly, the same law has been claimed in other forms of communication such as music (Serra et al. 2012) and the timbres of sounds (Haro et al. 2012). Various formulations of the law have been derived since the first proposal of Zipf's. However, pointed out by many, most research have soley focused on seeking the "true form" of the law, by deriving the formulation based on different principles. Little of them have tried to assess the underlying assumptions of the mass hypothesis. Thus, to make progress at understanding the full relationship between linguistics and Zipf's law, studies must seek evidence beyond the law itself. Validity and assumptions of the law needs rigorous statistical testings with novel predictions evaluated by new and independent data.

1.3 Goals

Previously, Moreno-Sanchez et al. (2016) surveys Zipf's Law and described three variants that satisfy it. Goodness-of-fit of the models has been tested against text data sets from *Project Gutenberg* (2019). Of the three models, two have been implemented in the VGAM R package. The aim of this work is to implement the third model proposed by Mandelbrot (1961). *Maximum Likelihood Estimation* (MLE) of the model will be developed, and further validated with the same word frequency data sets collected in the paper. To achieve this, more specific goals include the following:

1. The Zipf's formulation proposed by Mandelbrot (1961) will be written as a VGAM family function as `zipfmbrot()`. The *Probability Mass Function* (PMF) of the original distribution may be found at equation (2.2), and a reparameterised version as the equation (4.2) of which the implementation will be based on.

2. Along the family function, its `dpqr`-type functions [i.e., density, CDF, inverse-CDF and generation of random variates] will also be written for R.

3. All functions and the help files, once written and tested, will be developed into an R package called VGAMzm (VGAM Zipf-Mandelbrot). Data sets collected in Moreno-Sanchez et al. (2016) will be further processed into suitable format for analysis and included in the package.

4. The new family function `zipfmbrot()`, along with two pre-existing ones, `zetaff()`, `diffzeta()` will be compared, by fitting to the word frequency data sets followed by the *likelihood ratio* (LR) test.

1.4 Outline

The thesis comprises six chapters as outlined below:

Chapter 2 gives an introduction to the theoretical aspects of Zipf's law, and an overview of the existing literature. Three different variants the law are presented and discussed.

Chapter 3 covers the underlying theory of the VGAMs/VGLMs framework, and the practical functionality of the VGAM R package centering Zipf's distribution.

Chapter 4 discusses the methodology of implementing the family function `zipfmbrot()` and the four `dpqr-zipfmb()` functions. Essential details of the implementation such as the *Fisher Information* of the distribution and the *Expected Information Matrix* (EIM) will be provided.

Chapter 5 contains analysis and applications of the new functions, using the word frequency data sets processed. LR test will be carried out for comparisons between efficiencies of three Zipf's models.

Chapter 6 summarizes the thesis with its theoretical and practical contributions to the field. Limitations of the new function will be discussed for possible future works.

Appendix A includes the source code and comments of the implementation as a reference for future developers.

Appendix B includes other background materials needed throughout the thesis.

Through this work, we wish to show how VGLMs in general can provide a convenient vehicle for the analysis of such data.

Chapter 2

Zipf's Law

While studying the use of vocabulary in *Ulysses* by James Joyce, the Harvard linguist George Kingsley Zipf made a startling discovery in the 1930s about the statistics of language (Zipf 1932). Whether it is in speech or written text, a relatively small number of words are very frequently used, while the remainder fall on a very long tail. As the discovery has become known as Zipf's law, scholars began to investigate the mechanisms underlying this phenomenon. This chapter gives a brief overview to the existing literature surrounding the topic: Section 2.1 begins with the basic definitions of the quantities involved, followed by three variants derived from the law. Section 2.2 reviews relevant literatures and research of statistical linguistics. Section 2.3 reviews the latest methodologies used in parameter estimation. Section 2.4 gives an introduction to *Project Gutenberg* (2019) from which the word frequency data sets are collected and used throughout the work. Section 2.5 describes the likelihood ratio test for determining the best fitting variant to any empirical data set. All results from the tests will be detailed in Chapter 5 along with the new implementation done in Chapter 4.

2.1 Definitions

Zipf's law states that, given a large corpus of natural language—whether it be speech or written text—the frequency y of any word and its rank r of occurrence follow the power law: the relationship of y and r are inversely proportional in an approximately hyperbolic manner. The most frequently occurring word is assigned $r = 1$, the second $r = 2$, and so on. To be slightly more general, the formulation may include a scaling parameter in the form of an exponent α. Mathematically, the quantity r would obey the power law when drawn from the frequency distribution,

$$y(r) \ \propto \ \frac{1}{r^{\alpha}}, \tag{2.1}$$

with α typically close to 1. This, we call it a *frequency-rank relationship*. Note that in this work, we consider only the discrete distributions, thus $\{y \in \mathbb{R} : y \geq 0\}$. This expression will further be derived into three different variants.

2.1.1 Variant 1

Now let y represents the quantity, the *rank*, whose distribution we are interested in. The variant, in the very basic form of Zipf's distribution, is described by a *Probability Mass Function* (PMF) of

$$\Pr(Y = y; \beta) \ = \ C \cdot \frac{1}{y^{\beta}} \tag{2.2}$$

where Y is the observed value and C is a normalized constant. Sometimes, the power-law behaviour is seen only in the tail of the distribution. In such cases, the empirical phenomena does not obey the law for all values of y but only for values greater than some minimum y_{min}—which we call the *lower bound*. As y only takes in a discrete set of non-negative integers, we have $y_{min} > 1$. Thus $y = y_{min}, y_{min} + 1, \ldots$ up to infinity. Generalized by Mandelbrot (1961), one gains

$$f(y; \beta) \ = \ \frac{1/(y+q)^{\beta}}{H_{y_min,\beta,q}} \tag{2.3}$$

where $H_{y_min,\beta,q}$ is a generalisation of a harmonic number given by $H_{n,s,p} = \zeta(z,\ q) = \sum_{i=1}^{n} (i+p)^{-s}$. As y_{min} approaches infinity, the harmonic number becomes a Hurwitz zeta function $\zeta(z,\ q) = \sum_{n=0}^{\infty} (n+q)^{-z}$. For $q = 0$ we gain our first variant, the Zeta distribution, whose PMF is given by

$$f_1(y) \ \equiv \ \Pr(Y = y; \beta) \ = \ \frac{y^{-\beta}}{\zeta(\beta, y_{min})} \tag{2.4}$$

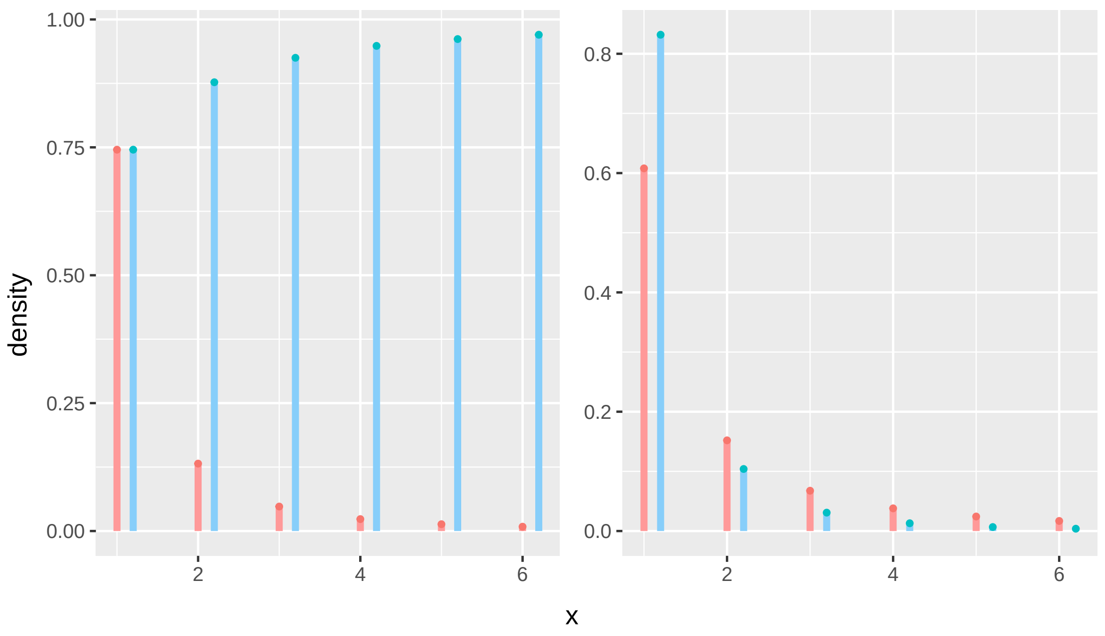

FIGURE 2.1: Left hand side: this illustrates the relationship of the mass (pink) and the cumulative density (blue) with `x` from 1:6; Right hand side: this compares the two different shape parameters, 1 (pink) and 2 (blue), on its PMF.

where $\beta \geq 1$ and $\zeta(\beta, y_{min})$ is the Hurwitz zeta function. Details of zeta functions may be found in Appendix B. For this work, the lower bound will be fixed at $y_{min} = 1$, in order to include the entire distribution from the very first value of y. Such assumption is made in works such as Goldstein et al. (2004) and Seal (1952). It may be useful to also consider the complementary CDF of the distribution, which we denote as $F(y)$ and is defined to be $F(y) = \Pr(Y \leq y)$. Thus one may obtain

$$F_1(y) \;\equiv\; \Pr(Y \leq y; \beta) \;=\; \frac{\zeta(\beta, y)}{\zeta(\beta, y_{min})}. \tag{2.5}$$

This distribution has been implemented in the VGAM R package as a family function `zetaff()`. Visualisation of both PMF and CDF is illustrated in Figure 2.1 using the associate `dpqr`-functions, `dzeta()` and `pzeta()`. An example of the family function `zetaff()` will be carried out in Section 3.3.1.

2.1.2 Variant 2

In the discrete case, a power law in the PMF does not lead to a power law in its CDF, and vice-versa. Although for large y we may approximate $f(y)$ with $-\frac{dF(y)}{dy}$ which implies that a power law in $f(y)$ does lead to a power law in $F(y)$–this

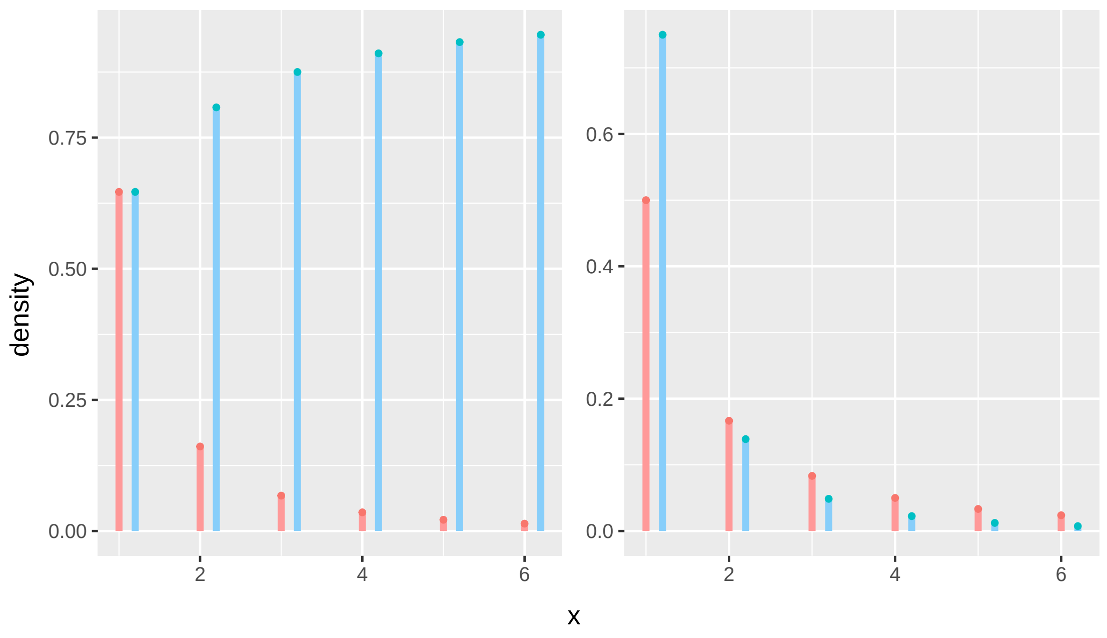

FIGURE 2.2: Left hand side: this illustrates the relationship of the mass (pink) and the cumulative density (blue) with `x` from 1:6; Right hand side: this compares the two different shape parameters, 1 (pink) and 2 (blue), on its PMF.

simplification is clearly wrong for small y in discrete distribution. Thus when the power-law nature exists in the CDF, we have

$$F_2(y) \equiv \Pr(Y = y; \beta) = \left(\frac{y_{min}}{y}\right)^{\beta-1}. \tag{2.6}$$

Based on the relationship $f(x) = F(x) - F(x+1)$ and $F(x) = \sum_{n'=n}^{\infty} f(n')$, we may further obtain the PDF

$$f_2(y) \equiv \Pr(Y = y; \beta) = \left(\frac{y_{min}}{y}\right)^{\beta-1} - \left(\frac{y_{min}}{y+1}\right)^{\beta-1} \tag{2.7}$$

where $\beta > 1$ and $y = y_{min}, y_{min} + 1, \ldots$. According to Moreno-Sanchez et al. (2016), this model accommodates over 40% of text data sets in Gutenberg database, detailed in Section 2.4. The VGAM R package also fits this distribution by the family function `diffzeta()`. Figure 2.2 again illustrates the PMF with its CDF using `ddiffzeta()` and `pdiffzeta()`. Comparisons are made between two shape parameters, 1 and 2, on the PMF. Refer to Section 3.3.2 for applications.

2.1.3 Variant 3

Apart from the first variant, Mandelbrot (1961) also derived another formulation, taking into account of the sample size of occurring words in a corpus. To not confuse this variant with the first one, which is also derived by Mandelbrot, this variant is named Zipf-Mandelbrot II distribution. This variant will be the target model to be implemented in Chapter 4. We first consider event (k, y) that the word comes out y times in k samples. Here r follows the rank-frequency relationship defined at (2.1) stated as a CDF, denoted $p(r) = r^{\beta-1}$ the probability of occurrence. The formulation thus involves a binomial distribution, resulting in a probability function

$$f(y,k) = \binom{y}{k} p(r)^y [1-p(r)]^{k-y}, \tag{2.8}$$

which has expected value

$$E[f(y,k)] = \binom{y}{k} \sum_{r=1}^{\infty} p(r)^y [1-p(r)]^{k-y}. \tag{2.9}$$

Let $p(r) = x$, with change of variables technique we gain $r = x^{-A}$, where $A = \beta-1$, and compare the sum to the integral

$$\int_0^1 x^y (1-x)^{k-y}\, dr(x) = A \int_0^1 (1-x)^{k-y}\, x^{y-1-A}\, dx. \tag{2.10}$$

For large k, the sum differs little from the sum restricted to some range such as $(10, \infty)$, and the integral differs little from the integral restricted to some range such as $(0, p(10))$. As the difference is small, we may approximate

$$\begin{aligned} E[f(y,k)] &\approx A \cdot \frac{\Gamma(k-y+1)\Gamma(y-A)}{\Gamma(k+1-A)} \cdot \frac{k!}{y!(k-y)!} \\ &= A \cdot \frac{\Gamma(y-A)}{\Gamma(y+1)} \cdot \frac{\Gamma(k+1)}{\Gamma(k+1-A)}. \end{aligned} \tag{2.11}$$

Substituting back $A = \beta - 1$, and let $k = y_{min} - 1$ for k being $1 \leq k < y_{min}$, one gains the third variant

$$f_3(y) \equiv \Pr(Y = y; \beta) = (\beta-1) \cdot \frac{\Gamma(y+1-\beta)}{\Gamma(y+1)} \cdot \frac{\Gamma(y_{min})}{\Gamma(y_{min}+1-\beta)} \tag{2.12}$$

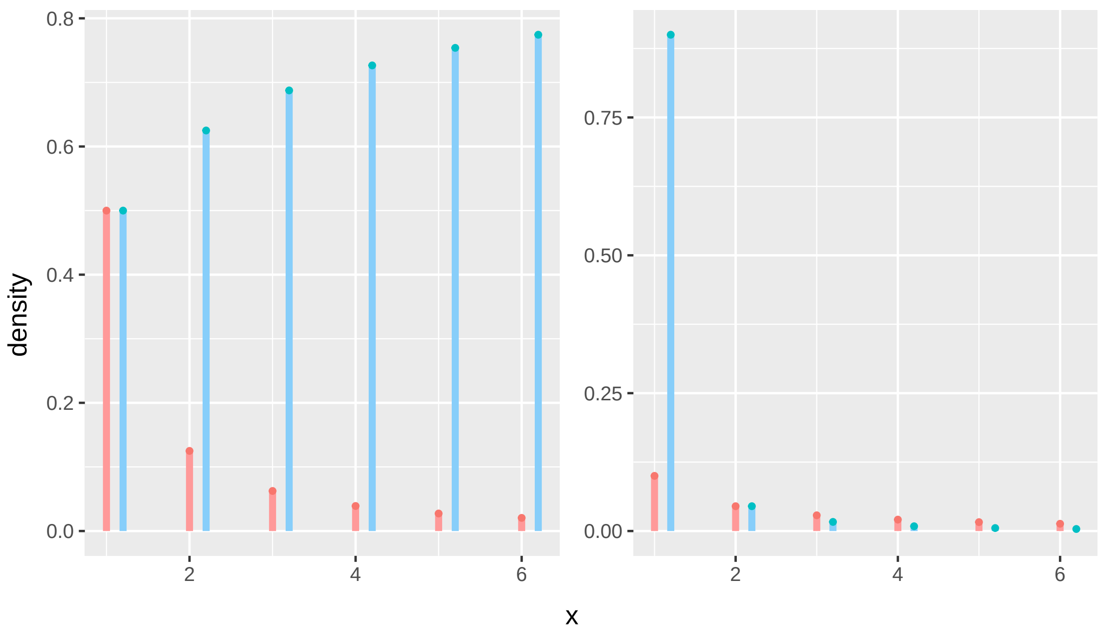

FIGURE 2.3: Left hand side: this illustrates the relationship of the mass (pink) and the cumulative density (blue) with `x` from 1:6; Right hand side: this compares the two different shape parameters, 1 (pink) and 2 (blue), on its PMF.

with the CDF being

$$F_3(y) \equiv Pr(Y = y; \beta) = \frac{\Gamma(y_{min})\ \Gamma(y + 1 - \beta)}{\Gamma(y)\ \Gamma(y_{min} + 1 - \beta)} \tag{2.13}$$

where $1 < \beta < 2$ is the parameter to be estimated, and $\Gamma(\gamma) = \int_0^\infty q^{\gamma-1} e^{-q} dq$ denotes the gamma function. This variant is unimplemented in VGAM Rprior to this work, and is developed in Chapter 4. Figure 2.4 makes use of the newly implemented `dpqr`-function, `dzipfmb()` and `pzipfmb()`, to again illustrate the relationship between the PMF with its CDF. Comparisons are made between two shape parameters, 1.1 and 1.9, on the PMF. Summary of three variants mentioned above can be found in Table 2.1.

2.2 Recent Research

Since Zipf's law was put forth, it has attracted particular attentions for its mathematical properties, and its appearance in a diverse range of natural and man-made phenomena. The pattern not only has been found across different spoken and written languages, it has also been claimed in other codes of communications. For example, the number of calls received by customers service during a day (Ebel

Variant	PMF	CDF	parameter
(1)	$\frac{y^{-\beta}}{\zeta(\beta,y_{min})}$	$\frac{\zeta(\beta,y)}{\zeta(\beta,y_{min})}$	$\beta \geq 1$
(2)	$(\frac{y_{min}}{y})^{\beta-1} - (\frac{y_{min}}{y+1})^{\beta-1}$	$(\frac{y_{min}}{y})^{\beta-1}$	$\beta \geq 1$
(3)	$(\beta-1)\frac{\Gamma(y+1-\beta)}{\Gamma(y+1)}\frac{\Gamma(y_{min})}{\Gamma(y_{min}+1-\beta)}$	$\frac{\Gamma(y_{min})}{\Gamma(y)}\frac{\Gamma(y+1-\beta)}{\Gamma(y_{min}+1-\beta)}$	$1 > \beta > 2$

TABLE 2.1: Summary of three Variants with PMF, CDF and the parameter β. First two variants are already implemented in VGAM R, while the third is the target model to be developed in this work.

et al. 2002), the number of bytes of data received as the result of individual web (HTTP) requests from computer users (Sani & Daman 2014) or the number of hits received by web sites (Ali & Scarr 2007); and also in disparate discrete systems where individual units or agents gather into different classes (Li 2002), for example, employees into firms (Axtell 2001), believers into religions (Clauset et al. 2009), insects in to plants (Pueyo & Jovani 2006), units of mass into animals present in ecosystems (Camacho & Sole 2001) or abundance of proteins in a single cell (Furusawa & Kaneko 2003).

Derivations of Zipf's law from the basic assumptions have been numerous. Scholars in literature have devoted in seeking the most precise form of the law, that best fits the word frequency distribution observed in natural language. Examples include the variants derived by Mandelbrot (1961) and Mandelbrot & Wallis (1968) in Section 2.1. A highly recognised comparison between different proposed models are carried out in Baayen (2001). Quantitative methods are used and validated by statistical tests. Baayen concludes that there is no one true model that fits all corpus–different variants capture different aspects of the full distribution of natural word frequencies. For example, For the book `Hound of the Baskervilles` is best fit by the log-normal, while `Alice in the Wonderland` is best fit by the Yule-Simon model.

However, most work in language research has devoted solely in seeking the true form of the law. Very little work has attempted to validate the underlying assumptions of the phenomena. Thus the validity of any forms Zipf's law has yet to be tested rom the statistical point of view. Over the last few years, there has been increasing rigor in testing Zipf's law claims. Most of which quantify the law with large sample of empirical data (Moreno-Sanchez et al. 2016), combining statistical testing procedure to validate different forms of Zipf's law. With growing maturity of statistical and technological amenity, large data sets of text are more accessible for rigorous testing. To this end, this work picks up from here, to test the variant not yet validated through the development of MLE under the VGAM R framework.

2.3 Fitting to Empirical Data

As mentioned, the mass literature has been seeking the correct fitting of Zipf's law forms to empirical distribution of word frequency data. In order to do so, the estimation of scaling parameter has often been the research question in many studies. Regardless of which formulation it is based on, parameter estimations of Zipf's law have traditionally been done through qualitative methods such as data visualization. In literature we have seen methods such as i) fitting full raw histogram on the double logarithmic plot (Li et al. 2010) or ii) fitting binned histograms on the double logarithmic plot (Newman & Park 2003). Observations such as having a linear regressive slope very close to -1, obeying $\log y(r) = \alpha \log r + C$, which gives an initial indication of the distribution obeying the Power-law. Examples of such approach can be seen on Figure 2.1 illustrating the rank-frequency relationship of 12 books from the Gutenberg database.

However, authors such as Goldstein et al. (2004) have pointed out that simple graphical methods suffer from severe systematic errors under relatively common conditions. Experiments are done in justification of more sophisticated approaches through quantitative measurements. An asymptotically unbiased and minimum-variance procedure based on CDF and MLE was proposed in White et al. (2008) that is shown to outperform other methods in both accuracy and precision. The solutions are moreover invariant under reparameterisations (Deluca & Corral 2013). Therefore MLE should be considered as the most reliable procedure for parametric estimation, given that a global maxima of likelihood does exist with a sufficient size of observations. Thus in Chapter 4, the distribution based on Zipf's law is implemented under the VGAM R framework suitable for solving MLEs.

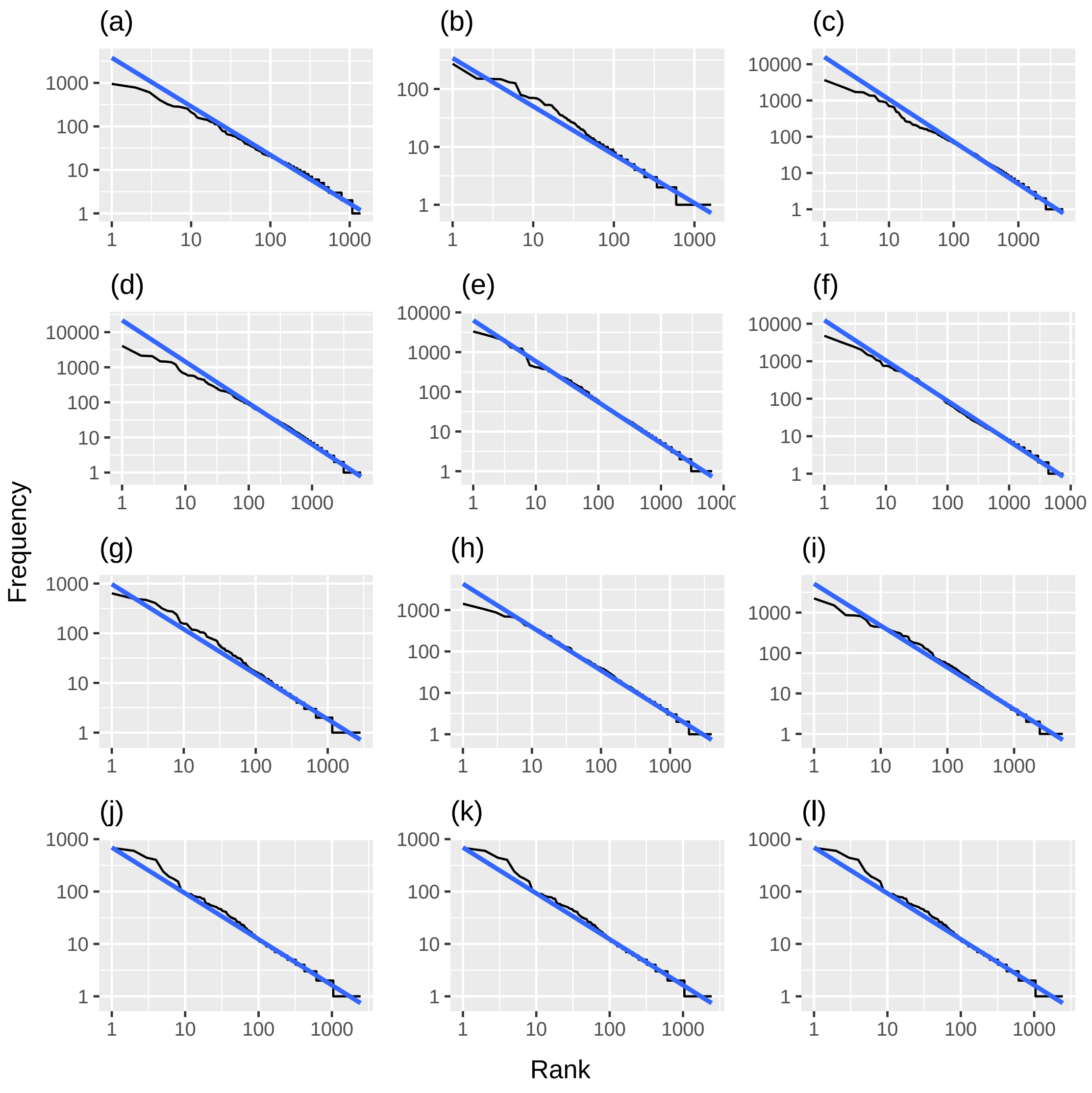

FIGURE 2.4: Zipf's law expressed by 12 books from the Gutenberg database, on a double-logarithmic scale with frequency rank on the x-axis and frequency on the y-axis. Observations of the straight slope very close to 1 indicate the tendency of distribution following the power law. 12 books are: (a) *On the Significance of Science and Art* by Leo Tolstoy (b) *What To Do?* by Leo Tolstoy (c) *The Titan* by Theodore Dreiser (d) *The Kingdom of Love, and Other Poems* by Ella Wheeler Wilcox (e) *Life of Bunyan* by James Hamilton (f) *Quotations from Abraham Lincoln's Writings* by David Widger (g) *Honore de Balzac* by Albert Keim and Louis Lumet (h) *Chaucer* by Adolphus William Ward (i) *The Golden Bough, abridged, in one volume* by James George Frazer (j) *The Duke's Children* by Anthony Trollope (k) *Peg O' My Heart* by J. Hartley Manners (l) *Geological Observations On South America* by Charles Darwin.

2.3.1 Parameter estimation

Estimating the parameter β requires a value for the lower bound y_{min} of power-law behavior in the data. For this work, the value is taken to be 1, to include the whole distribution from the very first value. Assuming that out data are drawn from a distribution that follows the power law exactly for $y > y_{min}$, we can derive MLE of the scaling parameter β. The estimation may be done by direct numerical maximisation of the likelihood function itself, or equivalently the log-likelihood function that is usually used for its simplicity. Given a set of data $x_1, \dots, x_n$ and the PMF parameterized by β, one may obtain the log-likelihood function

$$\ell(\beta) = \sum_{i=1}^{n} \log f(y_i; \beta). \tag{2.14}$$

The MLE of β is denoted as $\widehat{\beta}$ which maximizes $\ell(\beta)$. In the case of the first variant (2.1), the log-likelihood takes the form

$$\ell_1(y; \beta) = -\log \zeta(\beta, a) - \beta \log n; \tag{2.15}$$

the second variant, in the form of its CDF, gives

$$\ell_2(y; \beta) = (\beta - 1) \log \frac{y_{min}}{y} \tag{2.16}$$

while the PMF does not have an existing close form of log-likelihood. The third variant, which is derived from its form of the reparameterised version (4.1) has log-likelihood

$$\begin{aligned} \ell_3(y) &= \log \Pr(Y = y; \beta) \\ &= \log(\beta - 1) + \log \Gamma(a) + \log \Gamma(y + 1 - \beta) - \log \Gamma(a + 1 - \beta) - \log \Gamma(y + 1). \end{aligned} \tag{2.17}$$

Using VGAM, one could easily maximise the parameter β using IRLS through Fisher Scoring, detailed in Section 3.1.1. The first two forms of Zipf's distribution are already implemented, while the third is developed in Chapter 4.

Rank	Frequency	Rank	Frequency	Rank	Frequency	Rank	Frequency
1	1525	11	277	21	155	31	132
2	1313	12	270	22	151	32	131
3	1001	13	253	23	147	33	129
4	880	14	245	24	144	34	127
5	825	15	223	25	144	35	125
6	488	16	214	26	142	36	124
7	434	17	206	27	142	37	121
8	351	18	202	28	140	38	116
9	312	19	195	29	137	39	112
10	311	20	185	30	133	40	110

TABLE 2.2: Word frequency of *The Duke's Children, by Anthony Trollope*: Two variables are available in each data set—rank r and frequency y of each word.

2.4 Gutenberg Database

In order to test the model with empirical observations, word frequency data sets collected in Moreno-Sanchez et al. (2016) from the Project Gutenberg database is considered. The database consists of books and articles from different languages, times and periods. The collected data sets consist of articles over 100 word tokens pre-processed with removals off selected parts, such as notes of copyright, headers, punctuation sign and numbers etc. To be clearer, a *word token* refers to one individual occurrence of a *word type*. Each data set contains the word frequency y of each word type, sorting from the greatest to the smallest. In this word, each data set is further processed with additional variable of the rank of r, from $1, 2, \ldots$. A sample from one of the data set *The Duke's children, by Anthony Trollope* is shown in Table 2.1 with first 40 frequent words taken. The values of these frequencies, for each text, are available at
`http://dx.doi.org/10.6084/m9.figshare.1515919`.

The website contains a compressed folder containing 31075 numerical vectors. Each one represents word frequencies of an EBook from Project Gutenberg written in English. Vectors are named containing the ID number of their corresponding text in Project Gutenberg (https://www.gutenberg.org).

2.5 Hypothesis Testing

The methods described in Section 2.2 allow us to fit a Zipf's distribution to a given data set, and provide an estimate of the parameters β. However, it gives no indications whether Zipf's law is indeed a good model for the data. One could hardly determine the goodness-of-fit of any forms of Zipf's distribution to a particular data set through qualitative methods. Even if data are drawn from a Zipf's law, the observed distribution is unlikely to follow the exact form of Zipf's law due to small deviations as a random nature caused by the sampling process; vice versa, it is always possible that a non-Zipf's law process will happen to generate a data set with a distribution very close to a Zipf's law. Thus hypothesis testing is required to obtain statistical evidence of the true plausibility of the distribution. Such test is essential before any parameter estimation is performed. With this hypothesis, one may define

$$H_0 : D(y_e; \beta) - D(y_s; \beta) = 0 \tag{2.18}$$

$$H_1 : D(y_e; \beta) - D(y_s; \beta) \neq 0 \tag{2.19}$$

for y_e being the empirical distributed variable, and y_s the synthetic. The null hypothesis to be tested, is the deviation of the empirical data makes no difference to the synthetic data. If the zero difference is verified to the significance level, then the null hypothesis is rejected, indicating that the Zipf's law is not plausible for the empirical data. The effectiveness of this approach depends on how we measure the deviance between the data set and the distribution. Here, we use the *Kolmogorov-Smirnov* (KS) statistic, which is a good method of choice (Press et al. 1992). The procedure of the hypothesis test is detailed in Section 2.3.1. A large sample size of 31,075 text frequency data set described in Section 2.5.2 is used to overcome false acceptance or false rejection of hypothesis that may fail the test when single data set is used.

2.5.1 Goodness-of-fit procedure

The goal of this goodness-of-fit process is to determine significant difference between the empirical data from those drawn from the true Zipf's distribution. The testing procedure is described as follows.

- Estimate the scale parameter β of the given empirical data set.
- With the estimate $\widehat{\beta}$, generate random deviates from the true distribution.
- Calculate deviations of both data sets from the selected Zipf's distribution; $D(y, \beta) = 2\{\ell(y; y) - \ell(\mu; y)\}$.
- Calculate KS statistics of the hypothesis (2.19) using deviates from step 3.
- With significance level set at $\alpha = 0.1$, if the resulting p-value is less than 0.1, then the hypothesis is rejected concluding that the empirical data is not generated from Zipf's law.

One should note that the test can only be performed on nested models. Step 1–2 will make use of the `dpqr`-type functions of each variant for random deviates, and Fisher scoring maximization incorporated by VGAM framework for MLE accordingly. The same collection of text data sets described in Section 2.5.2 will be applied to each variant, which the results will be compared as an indication of the better fit of variant to general text data set. Pre-existing models of the first two variants are used along with the third variant developed in Chapter 4 for analysis mentioned above, with all results are discussed in Chapter 5.

2.5.2 Likelihood Ratio

In Section 2.3, the method provides a reliable way to test whether a given data set is plausibly drawn from a Zipf's law distribution. However, in many practical situations, the only interest may be to determine which available distribution is the best fit of the empirical data. Also, even if our data are well fit by a variant, it is still possible that another distribution would give a fit as good or better. In such cases, methods exist which allow direct comparisons between two distributions given. In this section, we chose to use the *likelihood ratio test* for determining the better of the two models.

The likelihood-ratio test assesses the goodness of fit of two competing statistical models based on the ratio of their likelihoods. The null hypothesis of the test suggests that the two likelihoods not differ by more than sampling error. The test tests whether the likelihood-ratio is significantly different from one. Equivalently, when log-likelihood is taken, it is to test whether the ratio is significantly different from zero. Two pairs of LR will be generated for each empirical data, to compare (1) distributions f_1 with f_3, and (2) distribution f_2 with f_3. The logarithmic ratio between two likelihoods is also computed, which may be positive or negative indicating the better distribution, or a zero indicating an event of a tie. The likelihood ratio between two distribution may simply be defined

$$R_{1,2} = \sum_{i=1}^{n} (\log f_1(x_i) - \log f_2(x_i)) \tag{2.20}$$

and under the null hypothesis that both models are equally good to describe the data $R_{1,2}$ should be normally distributed with $\mu = 0$ and a variance σ^2. Large absolute values of $R_{1,2}$ will lead to the rejection of the null hypothesis.

2.5.3 Likelihood Ratios $L_{1,3}$ and $L_{2,3}$

Recalling three distributions—f_1, f_2 and f_3—to be considered, with probability density p_1, p_2 and p_3. With a sample size of n, the likelihoods of the three distributions are

$$L_j = \prod_{i=1}^{n} p_j(x_i), j = 1, 2, 3. \tag{2.21}$$

$$\tag{2.22}$$

The ratios of the likelihoods of our interest $L_{1,3}$ and $L_{1,4}$ are

$$\begin{aligned} R_{1,3} &= \frac{L_1}{L_3} \prod_{i=1}^{n} \frac{p_1(x_i)}{p_3(x_i)}, \\ R_{2,3} &= \frac{L_2}{L_3} \prod_{i=1}^{n} \frac{p_2(x_i)}{p_3(x_i)}. \end{aligned} \tag{2.23}$$

The logarithmic ratios are then

$$\begin{aligned} R_{1,3} &= \sum_{i=1}^{n} [\log p_1(x_i) - \log p_3(x_i)] = \sum_{i=1}^{n} \log[\ell_i^{(1)} - \ell_i^{(3)}], \\ R_{2,3} &= \sum_{i=1}^{n} [\log p_1(x_i) - \log p_3(x_i)] = \sum_{i=1}^{n} \log[\ell_i^{(1)} - \ell_i^{(3)}]. \end{aligned} \tag{2.24}$$

The performance and effectiveness of the likelihood ratio test is validated with synthetic data in (Clauset et al. 2009). Future direction of the study of Zipf's law may be found in Chapter 6. The results are computed in conjunction with the new implementation of this work and discussed in Chapter 5.

Chapter 3

VGLMs

Being the centre piece of the VGAM package, VGLMs portray the foundational parametric class of models. Both VGAM and VGLM classes are implemented in the VGAM package for the R statistical computing environment (R Core Team 2018). This package provides a number of benefits surrounding parametric regression models. The nature of VGAM modular also provides an environment for programmers to add new methodology through the S4 object-oriented programming framework. This chapter gives a sketch picture of the ideas underpinning the VGLMs framework, with a particular emphasis toward the Zipf models described in Chapter 2. Section 3.1 describes the basic mechanism of VGLMs as a method of MLE computation based on IRLS. Section 3.2 provides a brief introduction to the software of VGAM. Section 3.3 illustrates the use of VGAM through two existing Zipf distributions in the VGAM R package. This chapter only introduces a small piece of the VGAM/VGLM framework. For more general details, refer to Yee (2015) and Yee (2013).

3.1 Basics

Suppose that the observed response $\boldsymbol{y}$ is a q-dimensional vector ($q \geq 1$), and the observed covariate $\boldsymbol{x} = (x_1, \ldots, x_p)^\top$ is a p-dimensional vector, where $x_1 = 1$ denotes the optional intercept. VGLMs are models comprising a conditional distribution of $\boldsymbol{y}$ given $\boldsymbol{x}$ in the form

$$f(\boldsymbol{y}|\boldsymbol{x}; \mathbf{B}) = g(\boldsymbol{y}, \eta_1, \ldots, \eta_M) \tag{3.1}$$

where $g(\cdot)$ is some known function, $\mathbf{B} = (\boldsymbol{\beta}_1 \boldsymbol{\beta}_2 \ldots \boldsymbol{\beta}_M)$ is a $p \times M$ matrix of unknown regression coefficients, and η_j denoting the jth linear predictor given by

$$\eta_j = \eta_j(\boldsymbol{x}) = \boldsymbol{\beta}_j^\top \boldsymbol{x} = \sum_{k=1}^{p} \beta_{(j)k}\, x_k, \quad j = 1, \ldots, M. \tag{3.2}$$

With (3.2) it is assumed that *all* parameters of the distribution may be potentially modelled as functions of $\boldsymbol{x}$. VGLMs may be seen as an extended version of GLMs that allows for multiple linear predictors, and encompasses models outside the small confines of the exponential family. In general there is no relationship between q and M: it depends specifically on the model or the distribution to be fitted. For Zipf's models, only the scaling parameter is to be estimated thus $M = 1$, and q may be any positive integer as the observed sample size.

Let $\boldsymbol{x}_i$ denote the explanatory vector for the ith observation, for $i = 1, \ldots, n$. Then the set of linear predictors for the ith individual, η_i, can be written as

$$\begin{aligned}
\boldsymbol{\eta} = \boldsymbol{\eta}(\boldsymbol{x}_i) &= \begin{pmatrix} \eta_1(\boldsymbol{x}_i) \\ \vdots \\ \eta_M(\boldsymbol{x}_i) \end{pmatrix} \\
= \mathbf{B}^\top \boldsymbol{x}_i &= \begin{pmatrix} \boldsymbol{\beta}_1^\top \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{\beta}_M^\top \boldsymbol{x}_i \end{pmatrix} = \begin{pmatrix} \beta_{(1)1} & \cdots & \beta_{(1)p} \\ \vdots & \ddots & \vdots \\ \beta_{(M)1} & \cdots & \beta_{(M)p} \end{pmatrix} \boldsymbol{x}_i \\
&= \begin{pmatrix} \boldsymbol{\beta}_{(1)} & \ldots & \boldsymbol{\beta}_{(p)} \end{pmatrix} \boldsymbol{x}_i.
\end{aligned} \tag{3.3}$$

The η_j of VGLMs may be applied directly to any parameters of a distribution rather than to a mean for GLMs. A simple example is a univariate distribution with a location parameter μ and a scale parameter $\sigma > 0$, where we may take $\eta_1 = \mu$ and $\eta_2 = \sigma$. In VGAM , there are currently over a dozen link functions available to be flexibly assigned to each η_j, according to the nature of each model. Details of link functions may be found in Section 3.2.1.

3.1.1 IRLS

The maximum likelihood of parameters are estimated by IRLS using weighted least squares iteratively. Having (3.3) to be estimated, the log-likelihood may be expressed as

$$\ell(\boldsymbol{\beta}) \;=\; \sum_{i=1}^{n} w_i \;\ell_i\{\eta_1(\boldsymbol{x}_i), \ldots, \eta_M(\boldsymbol{x}_i)\} \tag{3.4}$$

where η_j is defined in (3.2) and $\boldsymbol{\beta}$ is the iterated parameter to maximize the log-likelihood. Written in IRLS form,

$$\boldsymbol{\beta}^{(a)} = \left(\sum_{i=1}^{n} \mathbf{X}_i^{\top} \mathbf{W}_i^{(a-1)} \mathbf{X}_i\right)^{-1} \left(\sum_{i=1}^{n} \mathbf{X}_i^{\top} \mathbf{W}_i^{(a-1)} z_i^{(a-1)}\right) \tag{3.5}$$

at iteration a, where $\mathbf{X}_i$ is the ith term of the model matrix $\mathbf{X} \equiv (\mathbf{x}_i, \ldots, \mathbf{x}_n)^{\top}$. This corresponds to the covariates passed into the algorithm via the argument `formula` of `vglm()` in Section 3.2. And $\mathbf{W}^{(a)}$ is a known positive working weight matrix $(\mathbf{W}_1^{(a)}, \ldots, \mathbf{W}_n^{(a)})$, comprising

$$\mathbf{W}_i^{(a)} = -E\left(\frac{\partial^2 \ell_i}{\partial \eta_i^2}\right) \tag{3.6}$$

for each log-likelihood component ℓ_i. The individual EIMs are closely related to the working weight matrices $\mathbf{W}_i$ by the chain rule. The $\mathbf{z}_i$ are the *working responses* as an M-vector given by

$$\mathbf{z}_i^{(a-1)} = \mathbf{X}_i \boldsymbol{\beta}^{(a-1)} + \mathbf{W}_i^{-1(a-1)} \mathbf{u}_i^{(a-1)} \tag{3.7}$$

where the *score vector* $\mathbf{u}_i$ is

$$\mathbf{u}_i = \frac{\partial \ell_i}{\partial \eta}. \tag{3.8}$$

An initial approximation of $\boldsymbol{\beta}^{(0)}$ is chosen to give an initial estimate of $\mathbf{z}^{(0)}$ and $\mathbf{W}^{(0)}$. Then (3.5) is solved in order to obtain $\boldsymbol{\beta}^{(1)}$, which is used to get improved new values for $\mathbf{z}^{(t)}$ and $\mathbf{W}^{(t)}$, and so on until adequate convergence is achieved. The maximum likelihood estimate $\boldsymbol{\beta}^{(t+1)}$ is taken when the difference between successive approximations $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\beta}^{(t+1)}$ are sufficiently small. Practically, the EIMs are used—as opposed to *observed information matrices* (OIMs) derived by Newton-Raphson algorithm—because all the working weight matrices must be positive-definite. An important step when implementing VGLMs, therefore, is to derive the EIM where possible. Each of the log-likelihood, score vector and the EIM of the Zipf-Mandelbrot model may be found in Chapter 4.

3.2 The VGAM Package

Written in `S4` (Chambers 1998), a `vglm()` object is used in a similar manner as `glm()`, i.e., it inherits the ideas of data frames, families, IRLS and the formula language etc. The basic vglm() object comprises three arguments `formula`, `family` and `dataset`. For example,

```
vglm(formula = rank ~ 1,
     family = zetaff,
     data = book1)
```

solves for the MLE of the parameter β of the random sample *rank*, assumed from a *zeta* distribution, specified by the `zetaff()` family function. As $\beta > 0$, $\eta = \log \beta$ is the default. The `1` in the formula implies that there is an intercept only. The scope of the family function is fairly broad, which covers a wide range of multivariate response types and models, including univariate and multivariate distributions, categorical data analysis, quantile and expectile regression and much more.

Many options available with `glm()/gam()` are also available with `vglm()/vgam()`, e.g., `subset`, `na.action` and `trace` etc. Methods of the `vglm()/vgam()` object are also written for standard generic functions such as `coef()`, `fitted()`, `predict()`, `summary()` and `vcov()`. Some examples are included in Section 3.3

3.2.1 Link Functions

Almost all VGAM family functions allow for any link function to be assigned to each η to provide maximum capability. Recalling from Section 3.1 we have $\eta = g(\theta)$ where $g(\cdot)$ is the link function, θ is the parameter and η is the linear/additive predictor. The link $g(\cdot)$ must be strictly monotonic and twice-differentiable in its range. To assign the desired link, an extra argument needs to be passed in any known parameter associated with the link function. Table 3.1 lists the links, ranged within $0 \leq \theta \leq 1$ that may be used by the Zipf's models mentioned in Section 2.1.

While the default of the family function gives a reasonable first choice, users are able to assign different links. In order to encompass a wider selection of links, the model of (2.12) will be reparameterised before implementations in Section 4.1. Fuller choice of link functions may be found in (Yee 2015, pg. 10)

Link	$g(\theta)$	**Range of** θ
`cauchit`	$\tan(\pi(\theta - \frac{1}{2}))$	$(0, 1)$
`cloglog`	$\log_e\{-\log_e(1-\theta)\}$	$(0, 1)$
`fsqrt`	$\sqrt{2\theta} - \sqrt{2(1-\theta)}$	$(0, 1)$
`logit`	$\log_e(\theta/(1-\theta))$	$(0, 1)$
probit	$\Phi^{-1}(\theta)$	$(0, 1)$

TABLE 3.1: Some VGAM link functions available for the implemented model in Section 4.2. After reparameterisation, the shape parameter of (2.12) falls within $0 < \theta < 1$. After careful modifications to handle endpoints, commonly used link such as `cloglog`, `logit` and `probit` may be used.

3.2.2 The `summary` Generic Function

Having the generic function `summary()` being applied to a `vglm()/vgam()` object, an object of class `summary.vglm` is returned. The output is similar in its format with that of a `glm()` object, giving a matrix with 4-columns:

1. the estimates $\hat{\boldsymbol{\beta}}^*_{(j)k}$
2. the standard errors SE$\hat{\boldsymbol{\beta}}^*_{(j)k}$
3. the Wald statistics $(\hat{\boldsymbol{\beta}}^*_{(j)k}$ - $0)/SE(\hat{\boldsymbol{\beta}}^*_{(j)k})$, and
4. the 2-sided p-values.

The assumption is made that all the Wald statistics follow the standard normal distribution. Particularly, the null hypothesis $H_0 : \boldsymbol{\beta}^*_{(j)k} = 0$ and $H_0 : \boldsymbol{\beta}^*_{(j)k} \neq 0$ are tested. The matrix is suggested to be extracted by using the extractor `coef()/Coef()` to `summary()`. At the very right hand side of the output, one can find *significant starts* characters: '.', '*', '**', '***', indicating the significant level of the p-value. The use of stars has been controversial, and is sometimes suppressed by

```
summary(vglmObject, signif.stars = FALSE)
```

The overall residual *degrees of freedom* (df) is given at the bottom of the output, which can be called by `df.residual(vglmObject, type = ‘‘vglm’’)`. Example of the use can be found in Chapter 6.

3.3 Examples

Two variants mentioned in Section 2.1 are already implemented as a family function in VGAM. Two examples are below to illustrate the simple use of `vglm()` function.

3.3.1 Example 1: `zetaff()`

The family function `zetaff()` is written for the first variant (2.4). Recalling the Hurwitz zeta function $\zeta(z,\ q) = \sum_{n=0}^{\infty} (n+q)^{-z}$, where $q > 0$ is known here as the starting value. Since $q = 1$ by default, this function will therefore return *Riemann's Zeta Function* $\zeta_R(s) = \sum_{n=1}^{\infty} n^{-s}$. Thus the first variant may also be expressed as

$$P(Y = y) \ = \ \frac{1}{y^{\beta+1} \cdot \zeta_R(\beta + 1)}, \tag{3.9}$$

where $y = 1, 2...$ and the scaling parameter $\beta > 0$. Since the parameter is positive, a log link is the default. With this expression, the built-in `zeta()` function for Riemann's Zeta Function in the VGAM package may be used for computation. For further backgrounds of zeta formulations refer to Appendix B. An example with a data set `df` consisting the rank and frequency of words from *the book*, comparing to the true form of first zipf variant. A quick peak of the data set,

```
> head(df)


  freq rank
1 1525    1
2 1313    2
3 1001    3
4  880    4
```

```
5  825    5
6  488    6
```

Fitting the dataset to the distribution gives an coefficient of 0.2636.

```
> fit <- vglm(rank ~ 1, family = zetaff, data = df,
              trace = TRUE, weight = freq, crit = "coefficient")


VGLM    linear loop  1 :  coefficients = -1.3345821
VGLM    linear loop  2 :  coefficients = -1.3331693
VGLM    linear loop  3 :  coefficients = -1.3331682
VGLM    linear loop  4 :  coefficients = -1.3331682


> (p_hat <- Coef(fit))   # 0.2636407


  shape
0.26364
```

Validating the data set to the true distribution generated by the estimated parameter.

```
> true <- round(dzeta(df$rank, p_hat) * sum(df$freq), 1)
> head(cbind(true, df), n = 10)


     true freq rank
1  5127.7 1525    1
2  2135.6 1313    2
3  1279.4 1001    3
4   889.5  880    4
5   670.9  825    5
6   532.9  488    6
7   438.6  434    7
8   370.5  351    8
9   319.2  312    9
10  279.4  311   10
```

```
> par(mfrow = c(1, 2))
> plotvglm(fit)
```

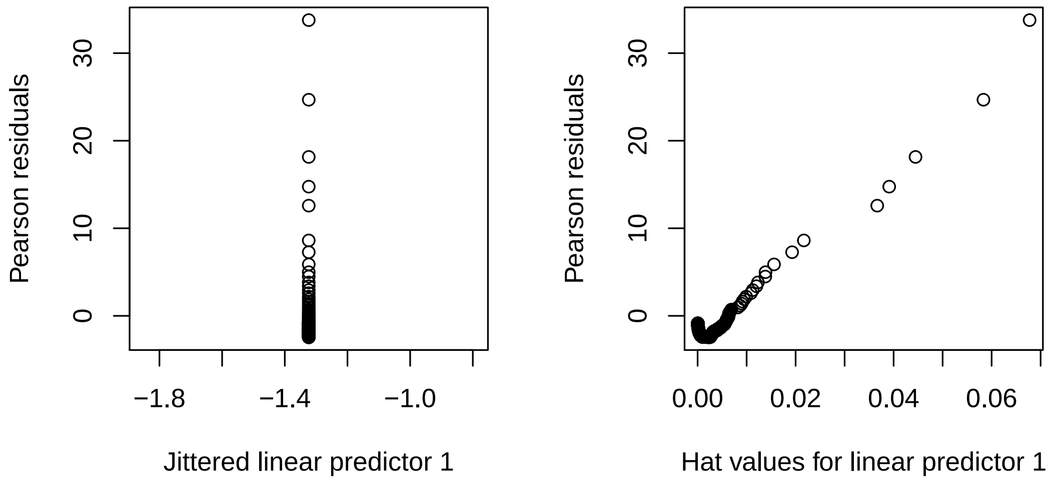

FIGURE 3.1: Visualization of Pearson residuals by fitting the book using `zetaff()`.

We may see that the data set begins to more closely follow the true distribution after a few values. In this case, the true starting value may be estimated in future work to provide more accurate inclusion of the distribution. Using vglm generic plotting function `plotvglm()`, one may see the Pearson residuals of the linear predictors.

3.3.2 Example 2: `diffzeta()`

The family function `diffzeta()` is written to accommodate the second variant (2.4). In this example,1000 random deviates from the distribution are generated using `rdiffzeta()`, with simulated shape parameters generated using log link functions `loge(-0.25 + runf(n), inverse = TRUE)` where inverse = TRUE indicates the inverse link value of θ, which returns η.

```
> n <- 10000
> x <- runif(n)
> shape <- loge(1.5 - x, inverse = TRUE)
> y <- rdiffzeta(n, shape)
> df2 <- data.frame(x, y, shape) ; head(df2)

         x y  shape
1 0.803030 3 2.0077
2 0.295252 1 3.3359
3 0.566409 1 2.5436
4 0.113888 1 3.9993
5 0.017597 1 4.4035
6 0.790463 1 2.0331
```

With the synthetic data set, MLE is performed using the family function `diffzeta()`,

```
> fit <- vglm(y ~ x, family = diffzeta, data = df2,
              trace = TRUE, crit = "coef")

VGLM    linear loop  1 :  coefficients =  1.7339434, -1.1610223
VGLM    linear loop  2 :  coefficients =  1.6049662, -1.1087857
VGLM    linear loop  3 :  coefficients =  1.5349428, -1.0393806
VGLM    linear loop  4 :  coefficients =  1.5216139, -1.0228557
VGLM    linear loop  5 :  coefficients =  1.5209610, -1.0221166
VGLM    linear loop  6 :  coefficients =  1.5209399, -1.0220918
VGLM    linear loop  7 :  coefficients =  1.5209393, -1.0220911
VGLM    linear loop  8 :  coefficients =  1.5209393, -1.0220911

> coef(summary(fit, matrix = TRUE))

            Estimate Std. Error z value    Pr(>|z|)
(Intercept)   1.5209   0.025541  59.548  0.0000e+00
x            -1.0221   0.041988 -24.342 6.9693e-131
```

We may see that the estimated intercept and the estimated x, counting into their standard errors, are fairly close to the preset value 1.5 and -1.

Chapter 4

The New zipfmbrot() Family Function

Recalling Chapter 3, the VGAM R package allows users to solve for MLEs of given empirical data. To do so, one is required to specify the distribution by passing in the specific family function to the `family` argument. Should one wish to self-implement a family function under the VGAM framework, sufficient knowledge of both the distribution and the functionality of VGAM are needed. This chapter aims to develop the third variant mentioned in Section 2.1.3. Section 4.1 gives an overview of the implementation, including the reparameterised version of the distribution that is being implemented. Section 4.2 discusses the newly developed `zipfmbrot()` family function. Section 4.3 discusses the associate `dpqr`-type functions. Section 4.4 wraps up the chapter with notes and package information. Programming details of all functions may be found in Appendix A.

4.1 Overview

To implement a distribution on top of VGAM, the work may be split into two parts: (1) the mathematical theory, and (2) the realization of the software. Speaking of the theory, a distribution model to be implemented is required to be one that can be estimated by IRLS. At its simplest, this points to a distribution possessing a log-likelihood (3.4), score vector (3.8) and EIM (3.6) that are mathematically tractable. Having the above satisfied and computed, each part may be placed into a `slot` of the family function written as an `S4` object. A number of essential and optional slots are present in each family function, to provide different functionalities when called by `vglm()/vgam()`. The slots used in the new `zipfmbrot()` family function and the respective mathematical computations, if applicable, may be found in Table 4.1.

Slot	Descriptions
`@blurb`	Prints out the name of the distribution and information of the parameter link functions.
`@infos`	Returns a list defining several entries, e.g., $M_1 = 1$, $Q_1 = 1$ and logical `multipleResponses = TRUE`
`@initialize`	Returns initial value that is used for the initial state of iteration, via `mustart` and/or `etastart`.
`@linkinv`	Returns fitted value ruturned by method `fitted()`. e.g., a matrix with rows of upper 0.75 quantile.
`@last`	Assings information such as `misc$link`, `misc$earg` (extra arguments) and `misc$start` to be evaluated at after final IRLS iteration.
`@loglikelihood`	Returns ℓ_i, computed in Section 4.2.1.
`@vfamily`	Family function name "zipfmbrot" as an identification used for S4 classes.
`@validparams`	Tests that the estimates are within the parameter domain. If the test fails, half-stepping is invoked, detailed in Section 4.2.4.
`@deriv`	Returns the score vectors in rows $w_i \partial \ell_i / \partial \eta$ with respect to the linear predictors, computed in Section 4.2.2.
`@weight`	Returns working weights `wz`, $w_i \mathbf{W}_i$, with respect to the linear predictors, computed in Section 4.2.3.

Table 4.1: The slots of family function `zipfmbrot()` (of S4 class `"vglmff"`, and the returned value.)

Before starting the implementation, a slight modification of the expression (2.3) is taken to accommodate greater flexibility provided by the link functions. Recalling Table 3.1, each link function associates with a domain for appropriate parameters to pass in to. As the parameter β from Section 2.1.2 falls within the range $1 < \beta < 2$, an adjustment is made with reparameterised $s = \beta - 1$ so that the new parameter switches to the range $0 < s < 1$. This enables a wider option of built-in link functions such as `logitlink`, `probitlink`, `clogloglink`, etc. to be used. Appropriate choice of link functions may increase the chances of successful convergence by avoiding numerical problems. After parameterization, the PMF becomes

$$f_3(y) \;\equiv\; \Pr(Y = y; s) \;=\; \frac{s\,\Gamma(y_{min})}{\Gamma(y_{min} - s)} \cdot \frac{\Gamma(y - s)}{\Gamma(y + 1)} \tag{4.1}$$

and CDF

$$F_3(y) \equiv Pr(Y = y; s) = \frac{\Gamma(y_{min})\,\Gamma(y - s)}{\Gamma(y)\,\Gamma(y_{min} + s)} \tag{4.2}$$

with the new parameter s. The PMF may be linked back to (3.1) as the expression of the Zipf-MandelbrotII in VGLM. These reparameterised expressions will be used throughout the implementation of both `zipfmbrot()` and `dpqr-zipfmb()`. More details on the implementation of family functions may be found in (Yee 2015, ch. 18).

4.2 Family Function `zipfmbrot()`

In this chapter, some important slots in the family function `zipfmbrot()` are unfolded and discussed. Among all slots, the greatest obstacle in the implementation is to compute adequate working weight matrices in Section 4.2.3, which is usually the case in implementing any family function. Some slots that are more obviously understandable are not explicitly detailed for the sake of simplicity.

4.2.1 Log-likelihood

As named, the slot `@loglikelihood` consists of the log-likelihood of the Zipf-Mandelbrot distribution, corresponding to (3.4). The `d`-type function `dzipfmb()` implemented in Section 4.3.1 is used with argument `log = TRUE` for this evaluation. The log-likelihood may be computed as

$$\begin{aligned}\ell_i(y_i; s_i) &= \log f(y_i; s_i)\\ &= \log s_i + \log\Gamma(y_i - s_i) + \log\Gamma(a) - \log\Gamma(y_i + 1) - \log\Gamma(a - s_i).\end{aligned} \tag{4.3}$$

As a note, the slot consists of the form

```
> args(zipfmbrot()@loglikelihood)


function (mu, y, w, residuals = FALSE, eta, extra = NULL, summation = TRUE)
NULL
```

The argument `summation = TRUE` indicates that all the ℓ_i is returned. If set `FALSE`, it returns a vector or matrix with elements $w_i \ell_i$. The full code of the slot may be found in Appendix A.

4.2.2 Score Vector

The slot `@deriv` typically returns $w_i \mathbf{u}_i$, where the elements of $\mathbf{u}_i$ corresponds to the score vector (3.8). The score vector is simply the first derivative of the log-likelihood to η, which may be evaluated using the chain rule in the form of

$$\mathbf{u}_i = \frac{\partial \ell_i}{\partial \eta} = \frac{\partial \ell_i}{\partial s} \cdot \frac{\partial s}{\partial \eta} \tag{4.4}$$

where we have

$$\begin{aligned} \frac{\partial \ell_i}{\partial s} &= \frac{1}{s} + \frac{\Gamma'(y-s)}{\Gamma(y-s)} - \frac{\Gamma'(a-s)}{\Gamma(a-s)} \\ &= \frac{1}{s} + \psi(y-s) - \psi(a-s) \end{aligned} \tag{4.5}$$

where $\psi(x) = \Gamma'(x)/\Gamma(x)$ is the digamma function. For expressions involving a link function η, internal functions such as `dtheta.deta(theta, link, earg)`, corresponding to $\partial\theta/\partial\eta$, may be used for computation of derivatives. Several of which are used throughout the implementation. Refer to Table 4.2 for functions of such. For full implementation of the slot refer to Appendix A.

4.2.3 Working Weight Matrix

The slot `@weight` returns the working weight matrix $w_i \mathbf{W}_i$ where the elements of $\mathbf{W}_i$ corresponds to the EIM of the distribution defined by (3.6). However, a challenging problem emerges for the computation of EIM, where an infinite series is involved due to the recurrence relation between two trigamma functions. An

Function	Value
`dtheta.deta(theta, link, earg)`	$\partial\theta/\partial\eta$
`d2theta.deta2(theta, link, earg)`	$\partial^2\theta/\partial\eta^2$
`eta2theta(theta, link, earg)`	$\theta = g^{-1}(\eta)$
`theta2eta(theta, link, earg)`	$\eta = g(\theta)$

TABLE 4.2: Internal functions of VGAM that invoke the link functions in the slot. These are used because the link chosen by the user is passed into the appropriate slot using `substitute()`. The argument `earg` contains a list which often defaults to `list(inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)`.

approximation is thus needed for replacement. Two algorithms are required for good computation of the working weight matrix:

4.2.3.1 Algorithm 1: `EIM.zipfmb.specilp()`

The first method approximates the EIM with the CDF of the distribution. First of all, the EIM may be obtained by using the chain rule

$$\frac{\partial^2 \ell_i}{\partial \eta^2} = \frac{\partial^2 \ell_i}{\partial s^2}\left(\frac{\partial s}{\partial \eta}\right)^2 + \left(\frac{\partial \ell_i}{\partial s}\frac{\partial^2 s}{\partial \eta^2}\right) \tag{4.6}$$

where $\partial\ell_i/\partial s$ is the score vector computed in slot `@deriv`; $\partial s/\partial\eta$ and $\partial^2 s/\partial\eta^2$ may be evaluated using `dtheta.deta(theta, link, earg)` and `d2theta.deta2(theta, link, earg)` respectively. Since $\partial^2\ell_i/\partial s^2$ is the second derivative of the log-likelihood to the parameter s

$$\frac{\partial^2 \ell_i}{\partial s^2} = -\frac{1}{s^2} + \psi'(y-s) - \psi'(a-s) \tag{4.7}$$

then the EIM is

$$-E\left(\frac{\partial^2 \ell_i}{\partial s^2}\right) = \frac{1}{s^2} + \psi'(a-s) - E[\psi'(y-s)] \tag{4.8}$$

where an the infinite series is involved with trigamma function evaluations

$$\psi'(a-s) - E[\psi'(y-s)] \quad (\equiv A, say). \tag{4.9}$$

Exploiting the recurrence relation $\psi'(x+1) = \psi'(x) - x^{-2}$ gives

$$\begin{aligned} A &= \psi'(a-s) - \sum_{y=a}^{\infty} f(y-s) \\ &= \psi'(a-s) - f(a)\psi'(a-s) - \sum_{y=a+1}^{\infty} f(y)\left\{\psi'(a-s) - \sum_{i=a}^{y-1}\frac{1}{(i-s)^2}\right\} \\ &= \sum_{y=a}^{\infty}\frac{\Pr(Y \geq y+1)}{(y-s)^2} \end{aligned} \tag{4.10}$$

As a result we could approximate the infinite series A with a finite sum of inverse CDF values:

$$A \approx \sum_{y=a}^{U}\frac{Pr(Y \geq y+1)}{(y-s)^2} \tag{4.11}$$

for some finite upper bound U. The approximation is written as an extra function called `EIM.zipfmb.specialp()` for computation of the final expression. This expression is used in several distributions such as the negative binomial (Lawless 1987). With the final evaluation involving the CDF of the distribution, the numerator may be computed using `pzipfmb()` implemented in Section 4.3.2 with the argument `lower.tail = FALSE` in the slot. The full implementation of the slot may be found in Appendix A.

4.2.3.2 Algorithm 2: Simulated Fisher Scoring (SFS)

However, sometimes, the first algorithm fails when the upper bound reaches the limit of computation. In this case, the second choice is to approximate the EIM by the *Simulated Fisher Scoring* (SFS) method. The method approximates

$$Var\left(\frac{\partial \ell_i(y)}{\partial \boldsymbol{\beta}}\right) \tag{4.12}$$

by simulating y at the current iteration $\hat{\beta}$. Here, repeated realizations of the score vector are generated, by `rzipfmb()`, to compute the sample variance. The calculation is done for all i by a vectorised computation. With the model that is intercept-only, the variance is averaged over all i. The number of simulations is specified in the argument `nsimEIM`, which is defaulted by 300. A larger number of simulations would lead to a more accurate approximation.

4.2.4 Half Stepping

The slot `@validparam` tests that the estimates β fall within the parameter domain. If not, the slot returns `FALSE` and `vglm` issues a warning and invokes *half-stepping* to occur. Half-stepping is used to prevent an iteration stepping too far in IRLS—resulting in a reduction of the likelihood, $\ell^{(a)} < \ell^{(a-1)}$. The solution enables the algorithm to repeatedly half the step until the likelihood is increased. The iteration at (a) may be shown as

$$\boldsymbol{\beta}^{(a)} = \boldsymbol{\beta}^{(a-1)} + \mathbf{h}^{(a-1)} \tag{4.13}$$

where $\mathbf{h}^{(a-1)}$ is the *step* added to the estimation at iteration a. Half step-sizing allows $\mathbf{h}^{(a-1)}$ to be replaced by smaller steps such as $\frac{1}{2}\mathbf{h}^{(a-1)}$, $\frac{1}{4}\mathbf{h}^{(a-1)}$, $\frac{1}{8}\mathbf{h}^{(a-1)}, \ldots,$ if $\ell^{(a)} < \ell^{(a-1)}$. The iteration is not completed until $\ell^{(a)} > \ell^{(a-1)}$. The process involves computing ℓ at

$$\boldsymbol{\beta}^{(a-1)} + \alpha\big(\boldsymbol{\beta}^{(a)} - \boldsymbol{\beta}^{(a-1)}\big) = (1-\alpha)\boldsymbol{\beta}^{(a-1)} + \alpha\boldsymbol{\beta}^{(a)} \tag{4.14}$$

for $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ until an improvement occurs.

Half-stepping is used to ensures an improvement is made every iteration, and is particularly useful at early stages of the iteration if the initial values are poorly chosen.

4.3 dpqr-functions

As well as providing a VGAM family function in Section 4.2, the associate `dpqr`-type functions that compute probability density (mass) function, cumulative density function, quantile function and random generator respectively are also implemented. The four `dpqr-zipfmb()` functions follow standard `S` conventions as other `dpqr`-type functions in VGAM. Refer to Table 4.3 for the summary of the functions and default values. The fuller codes and details of implementations may be found in Appendix A.

Function	Returned Value
`dzipfmb(x, ..., log = FALSE)`	$f(x)$
`pzipfmb(q, ..., lower.tail = TRUE, log.p = FALSE)`	$F(q) = \Pr(X \leq q)$
`qzipfmb(p, ..., lower.tail = TRUE, log.p = FALSE)`	$\min_{q:p \leq F(q)} q = F^{-1}(p)$
`rzipfmb(n, ...)`	$y_i \sim F$ independently

TABLE 4.3: The four `dpqr-zipfmb()` functions following S conventions associated with distributions and random variates. Arguments specific to the distribution are denoted by .... Options of returning natural logarithmic values are available for `dpq-zipfmb()`, yet defaulted to `FALSE`. Lower tail is defaulted to be returned, `lower.tail = FALSE` returns the upper tail.

4.3.1 Density Function `dzipfmb()`

As a discrete distribution, `dzipfmb()` returns the PMF of (4.2), with default `start = 1` for inclusion of whole distribution, and `log = FALSE` for natural density. The Shape parameter falls between $0 < s < 1$, e.g.

```
> dzipfmb(x = 1:10, shape = 0.5, start = 1, log = FALSE)


 [1] 0.5000000 0.1250000 0.0625000 0.0390625 0.0273438 0.0205078
 [7] 0.0161133 0.0130920 0.0109100 0.0092735
```

To note that, in keeping consistent with `R`, the word *density* is loosely termed here to denote f in general. More strictly, in discrete distributions, $f(y) = \Pr(Y = y)$ is called the *probability mass function* (PMF), whereas the *probability density function* (PDF) is reserved for continuous Y. With further looseness, although Chapter 3 uses Y as the response, we deviate here and use X to keep it compatible with general `d`-type functions with the first argument `x`.

The function comes with 4 arguments: `x`, `shape`, `start` and `log`. The logical argument `log` indicates whether the log-likelihood, natural logarithm of the density, is returned. The implementation involves using `lgamma()` for computation of the natural logarithm of the gamma function for more accurate results, as it is less likely to suffer from numerical problems such as overflow. For example, taking $y = 1000$, $y_{min} = 1$ and shape $= 0{:}5$, one wishes to compute the density,

```
> y <- 1000; ymin <- 5; shape <- 0.2
> dens <- shape + gamma(ymin) + gamma(y - shape) -
          gamma(ymin - shape) - gamma(y + 1)
> dens


[1] NaN
```

yet the large input value y causes overflow and results in `NaN`; while if `lgamma()` is used instead,

```
> ldens <- log(shape) + lgamma(ymin) + lgamma(y - shape) -
           lgamma(ymin - shape) - lgamma(y + 1)
> exp(ldens)


[1] 6.7601e-05
```

we see that a more accurate answer is returned. Thus the calculation in the implementation always begins with the logged value, then take the exponential if `log = FALSE` is favoured over the other. The same technique is also used in the implementation of `pzipfmb()`.

4.3.2 Cumulative Function `pzipfmb()`

`pzipfmb()` returns (2.13) the CDF of the distribution, e.g.

```
> pzipfmb(q = 1:10, shape = 0.5, start = 1, lower.tail = TRUE, log = FALSE)


 [1] 0.50000 0.62500 0.68750 0.72656 0.75391 0.77441 0.79053 0.80362
 [9] 0.81453 0.82380
```

for the lower tail of the Mandelbrot-ZipfII distribution. Two functions `log1p()` and `expm1()` are used in conjunction with `lgamma()` mentioned in Section 4.3.1 to control overflow when computing the lower tail probability. The function log1p(x) computes `log(1 + x)` accurately even for $x << 1$; while `expm1(x)` computes `exp(x) - 1` accurately even for $x << 1$. For example, to compute the lower tail probability of quantile q, instead of computing `1 - exp(log.q)` we use `-expm1(log.q)`; with the probability of the upper tail, we use `log1p(-exp(log.q))` instead of `log(1-log.q)` for equivalent but more accurate results.

To validate the implementation, we can make use of `dzipfmb()` in conjunction with `cumsum()` which returns the cumulative sums, so

```
> cumsum(dzipfmb(1:10, shape = 0.5, start = 1))

 [1] 0.50000 0.62500 0.68750 0.72656 0.75391 0.77441 0.79053 0.80362
 [9] 0.81453 0.82380

> pzipfmb(1:10, shape = 0.5, start = 1)

 [1] 0.50000 0.62500 0.68750 0.72656 0.75391 0.77441 0.79053 0.80362
 [9] 0.81453 0.82380
```

The `pzipfmb()` with argument `log = T` is largely used in other implementations such as `qzipfmb()` and the family function `zipfmbrot()`.

4.3.3 Inversed CDF `qzipfmb()`

The `qzipfmb()` function returns the quantile, that is, the inversed CDF given by $q = F^{-1}(p)$. The *Bisection Method* is used for approximation since there is no close form expression for the solution. The method approximates the root α by repeatedly bisecting the closed interval $[a, b]$. According to the *intermediate value theorem*, the continuous function f must contain at least one root in the interval (a, b) when the inequality $f(a) \cdot f(b) < 0$ is satisfied. With an allowable error ϵ, we use the following step over iteration i:

1. Let $c_i = \frac{a_i+b_i}{2}$, that is, the midpoint of the interval $[a, b] = [a_i, b_i]$.
2. If $b_i - c_i = \frac{b_i-a_i}{2} \leq \epsilon$, accept c_1 as a sufficient approximation of the root α; otherwise, if $b_i - c_i = \frac{b_i-a_i}{2} > \epsilon$, move to step 3.
3. If $f(b_i) \cdot f(c_i) < 0$, set $a_{i+1} = c_i$ and $b_{i+1} = b_i$; otherwise $a_{i+1} = a_i$ and $b_{i+1} = c_i$ then return to step 1. for next iteration.

The method is called by `bisection.basic()`. The lower bound is set to be $low = Y_{min-0.5}$, and the higher bound $high = 2 \cdot low + 10$. To validate the approximated answer, we can make use of the CDF `pzipfmb()`,

```
> quantile <- 1:10
> (prob <- pzipfmb(quantile, shape = 0.7, start = 1)) #CDF

 [1] 0.70000 0.80500 0.85050 0.87666 0.89393 0.90630 0.91567 0.92305
 [9] 0.92904 0.93400

> qzipfmb(prob, shape = 0.7, start = 1) # quantile should be 1:10

 [1]  1  2  3  4  5  6  7  8  9 10
```

where one can see that the CDF is properly inversed.

4.3.4 Random Deviates `rzipfmb()`

To generate random deviates from the Zipf-MandelbrotII distribution, the *transformation method* is used for its simplicity. Having the PMF from (4.1), we wish to draw random deviates of integer $y \geq y_{min}$. Usually a source of random deviates r uniformly distributed in $[0, 1]$, will be generated by a standard pseudo-random number generator. Then we have the PMF and $\Pr(r)$ related by

$$\Pr(y) \;=\; \Pr(r)\,\frac{dr}{dy} \;=\; \frac{dr}{dy} \tag{4.15}$$

where the second equality follows because $\Pr(r) = 1$ over the interval $[0, 1]$. Integrating both sides with respect to y, one gains

$$\Pr(y) \;=\; \sum_{y'=y}^{\inf} \Pr(y') \;=\; 1 - r \tag{4.16}$$

or equivalently

$$y \;=\; P^{-1}(1 - r). \tag{4.17}$$

where P^{-1} indicates the inverse of the CDF (4.2), which can be computed by `qzipfmb()`. For example, to generate 30 deviates with shape parameter $= 0.2$,

```
> set.seed(100)
> sort(rzipfmb(n = 20, shape = 0.2), decreasing = T)


 [1] 20580  2012   619   164   118    63    26    24    13    11     6
[12]     5     4     4     3     3     2     2     1     1
```

4.4 Writing R package

All the functions written developed in this chapter is built into an R package VGAMzm. Some data sets mentioned in Chapter 5 are also included for analysis. The package includes

1. `zipfmbrot()`: the Zipf-MandelbrotII family function
2. `dzipfmb()`: the probability mass function
3. `pzipfmb()`: the cumulative probability function
4. `qzipfmb()`: the quantile function, that is, the inversed CDF
5. `rzipfmb()`: the random deviates generator
6. `EIM.zipfmb.specialp()`: internal function, returns special p-function for computation of EIM;
7. books: data sets, containing 20 books from the Gutenburg Database, with 2 variables `rank` and `freq`

An `.Rd` help file for each function and data set, with a R manual are written. Inside the package, the file `NAMESPACE` is included for declaration of which variables, functions and S4-style classes to export and import. The `DESCRIPTION` includes package dependencies. An `.Rd` help file for each function is also created. The package may be found as a supplementary file to this thesis.

Chapter 5

Uses and Applications

In this chapter, the basic usage of the new family function `zipfmbrot()` is covered, and further compared with other two models `zetaff()` and `diffzeta()`. Data sets collected from the Gutenberg database described in Section 2.4 are being considered, in which 20 of them are attached in the VGAMzm R package to be used here.

5.1 Basic Use of `zipfmbrot()`

Data set `book2` is loaded from the package VGAMzm, with the frequency and the respective rank of the top 10 words.

```
> data(books) # loading 20 bookss
> head(book2, n = 10)


   freq rank
1   949    1
2   781    2
3   605    3
4   407    4
5   326    5
```

```
6   288    6
7   284    7
8   270    8
9   255    9
10  212   10
```

The data set is now fitted with the rank of each word as the response y, weighted by the word frequency `weight = freq`. With the family function `zipfmbrot()`, we are fitting the response to the Zipf-Mandelbrot distribution.

```
> fit <- vglm(rank ~ 1, family = zipfmbrot, data = book2,
                   weight = freq, trace = TRUE)


VGLM    linear loop  1 :  loglikelihood = -93934.5034
VGLM    linear loop  2 :  loglikelihood = -126158.985
Taking a modified step...................
```

In vglm, the argument `trace = TRUE` should always be included to monitor the convergence of the estimate. Recalling Section 3.2.2, generic functions such as `summary()` and `Coef()/coef()` may be applied.

```
> summary(fit)



Call:
vglm(formula = rank ~ 1, family = zipfmbrot, data = book2, weights = freq,
    trace = TRUE)


Pearson residuals:
                  Min   1Q Median   3Q  Max
logitlink(shape) 2.54 3.71   4.51 6.18 25.3
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.16081    0.00973   -16.5   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Name of linear predictor: logitlink(shape)

Log-likelihood: -93935 on 1371 degrees of freedom

Number of Fisher scoring iterations: 2

No Hauck-Donner effect found in any of the estimates
```

The result gives an estimate of -0.16081, with a standard error 0.00973. To only extract the summary, we use the extractor `coef()` in conjunction with the result.

```
> coef(summary(fit))

            Estimate Std. Error z value   Pr(>|z|)
(Intercept) -0.16081  0.0097298 -16.528 2.3067e-61
```

`Coef()` is used to obtain the regression coefficients. Then we have the estimate

```
> Coef(fit)

  shape
0.45988
```

For more extractor, refer to Yee (2015) [ch. 8]. Further, we may fit the model with the frequency being the covariate. This time, 100 data points are generated using `rzipfmb()`, with simulated shape value generated by the link function `logitlink()`.

```
> n <- 100
> x <- runif(n)
>
> shape <- logitlink(1.5, inverse = TRUE)
> rank <- 1:n
> freq <- sort(rzipfmb(n, shape), decreasing = T)
>
> df2 <- data.frame(rank, freq, shape)
> head(df2, n = 10) # quick check of data

   rank freq   shape
1     1   92 0.81757
2     2   58 0.81757
3     3    8 0.81757
4     4    7 0.81757
5     5    7 0.81757
6     6    7 0.81757
7     7    6 0.81757
8     8    4 0.81757
9     9    4 0.81757
10   10    3 0.81757

> fit <- vglm(rank ~ freq, family = zipfmbrot, data = df2, trace = TRUE)

VGLM    linear loop  1 :  loglikelihood = -641.3396
VGLM    linear loop  2 :  loglikelihood = -735.41279
Taking a modified step...................
```

```
> coef(summary(fit, matrix = TRUE))

              Estimate Std. Error  z value   Pr(>|z|)
(Intercept) -0.3699915  0.0849939 -4.35316 1.3419e-05
freq        -0.0063937  0.0083712 -0.76378 4.4500e-01
```

Also, we may compare the empirical data, with the true density generated by `dzipfmb()` using the estimated scaling parameter.

```
> fit <- vglm(rank ~ 1, family = zipfmbrot, data = book5,
                   trace = TRUE, weight = freq)

VGLM    linear loop  1 :  loglikelihood = -469565.265
VGLM    linear loop  2 :  loglikelihood = -657216.938
Taking a modified step..................

> (p_hat <- Coef(fit)) #0.4551465

  shape
0.48107

> true <- round(dzipfmb(book5$rank, p_hat) * sum(book5$freq), 1)
> head(cbind(true, book5), n = 10)

     true freq rank
1  30251.8 4055    1
2   7849.2 2135    2
3   3974.1 2069    3
4   2502.6 1471    4
5   1761.3 1445    5
```

```
6    1326.5  1389    6
7    1045.9  1203    7
8     852.2   831    8
9     712.0   694    9
10    606.5   647   10
```

Some similarities can already be seen, just with this simple comparison.

5.2 Comparisons over 3 Variants

The three variants `zetaff()`, `diffzeta()` and `zipfmbrot()` are fitted to the same data set.

```
> fit1 <- vglm(rank ~ 1, family = zetaff, data = book11,
    trace = FALSE, weight = freq)
> fit2 <- vglm(rank ~ 1, family = diffzeta, data = book11,
    trace = FALSE, weight = freq)
> fit3 <- vglm(rank ~ 1, family = zipfmbrot, data = book11,
    trace = FALSE, weight = freq)
```

The coefficients are given as

```
> Coef(summary(fit1)); Coef(summary(fit2)); Coef(summary(fit3))


  shape
0.21724
  shape
0.24279
  shape
0.45657
```

We can also perform the LR test easily with the vglm/vgam objects.

```
> lrtest(fit1, fit3)


Likelihood ratio test


Model 1: rank ~ 1
Model 2: rank ~ 1
   #Df LogLik Df Chisq Pr(>Chisq)
1 2456 -70793
2 2456 -76603  0 11622     <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


> lrtest(fit2, fit3)


Likelihood ratio test


Model 1: rank ~ 1
Model 2: rank ~ 1
   #Df LogLik Df Chisq Pr(>Chisq)
1 2456 -70166
2 2456 -76603  0 12876     <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From here, we expand the analysis into 20 data sets, with estimates over 20 books given in the same data frame for comparisons.

Table 5.1 shows estimates of Zipf's exponent β in the fits of three variants. Estimated mean and standard deviation of β are 0.212 and 0.017 respectively for f_1; 0.236 and 0.021 for f_2; 0.468 and 0.009 for f_3. Visualisation of the comparison over three different variants of the 20 books may be found at Figure 5.1.

Rank	zeta	diffzeta	zipfmbrot
1	0.23055	0.25897	0.47665
2	0.23960	0.27059	0.45988
3	0.21291	0.23815	0.45515
4	0.21969	0.24578	0.47230
5	0.20986	0.23397	0.48107
6	0.20973	0.23358	0.47382
7	0.20721	0.23059	0.48048
8	0.21237	0.23733	0.46267
9	0.21324	0.23839	0.47163
10	0.20775	0.23152	0.47580

TABLE 5.1: Three Zipf's variants are fitted to 20 books. This table shows each exponent β estimate. Estimated mean and standard deviation of β are 0.212 and 0.017 respectively for f_1; 0.236 and 0.021 for f_2; 0.468 and 0.009 for f_3. Estimated mean and standard deviation of β are 0.212 and 0.017 respectively for f_1; 0.236 and 0.021 for f_2; 0.468 and 0.009 for f_3.

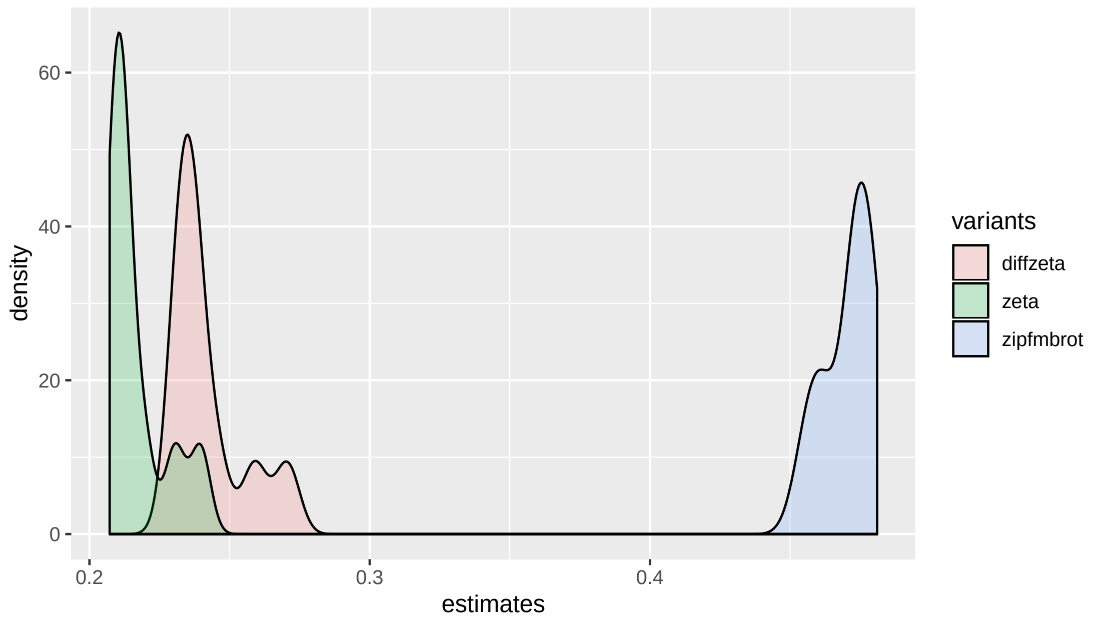

FIGURE 5.1: Comparisons of β estimates between three variants over 20 books

We conclude that the second variant `diffzeta()` does perform superior than others, as shown in Moreno-Sanchez et al. (2016). However, as mentioned, all variants do capture different aspects of the full distribution.

Chapter 6

Conclusions and Future Work

6.1 Theoretical Contribution

This work adds an additional model *Zipf-MandelbrotII* upon the VGAM framework on top of 150+ models. The extension is developed into a new VGAMzm R package depending on the VGAM R package. The package includes the family function `zipfmbrot()` that estimates the MLE of scaling parameter; and the associate `dpqr-zipfmb()` functions that correspond to the density mass function, cumulative distribution function, quantile function and the random deviates generator.

6.2 Practical Application

This additional model provides an additional accessibility to perform parametric estimation on empirical distribution of linguistics data sets. A suitable data set is in the form of two variables—the *frequency* of each occurring word, and the frequency *rank* of which. Such data set may be found in projects such as *A standardized Project Gutenberg corpus for statistical analysis of natural language and quantitative linguistics* (Gerlach & Font-Clos 2018), where data from books in the Gutenberg database is collected in such form.

6.3 Limitations and Future Work

While the new `zipfmbrot()` family function may perform maximum likelihood estimation on the scaling parameter, the lower bound y_{min} is assumed to be known. The estimation of the true y_{min} is yet to be implemented, where in most practical cases the exact value is unknown and to be estimated. As VGAM is designed to handle multiple parameter estimations, future work may focus on including the additional parameter y_{min} to be estimated as another interest of quantity.

Appendix A

Source Code

A.1 Density Function `dzipfmb()`

The `dzipfmb()` function contains 4 arguments: `x`, `shape`, `start` and `log`. As mentioned in Section 2.1.3 the starting value will be set to `start = 1` for inclusion of the whole distribution. The `log` argument follows the standard S conventions to be set `FALSE` as default. Refer back to Section 4.3.1 for more details.

```
dzipfmb <- function(x, shape, start = 1, log = FALSE) {

  if (!is.logical(log.arg <- log) || length(log.arg) != 1)
    stop("bad input for argument 'log': value must be of unit length")
  rm(log)

  LLL <- max(length(shape), length(x), length(start))
  if (length(x)     != LLL) x     <- rep_len(x,     LLL)
  if (length(shape) != LLL) shape <- rep_len(shape, LLL)

  bad <- !is.finite(x) | round(x) != x | x < 1 |
    shape <= 0 | shape >= 1 | start > x

  ans <- rep_len(if (log.arg) log(0) else 0, LLL)

  if (any(!bad)) {
    ans[!bad] <- log(shape[!bad]) +
      lgamma(start) + lgamma(x[!bad]-shape[!bad]) -
      lgamma(start-shape[!bad]) - lgamma(x[!bad]+1)
  }

  ans[round(x) != x | x < 1 | x == Inf | x == -Inf] <- 0
  ans[is.nan(x) | is.na(x)] <- 0
  ans[shape <= 0 | shape >= 1] <- 0
  ans[start > x] <- 0

  if (log.arg)
    ans
  else exp(ans)

} # dzipfmb
```

A.2 Cumulative Function pzipfmb()

The `pzipfmb()` function contains an additional argument: `log.p = FALSE` which controls if the final output is to be in the logarithmic form when set `TRUE`. Note that `lgamma()` is used followed by taking the exponential to avoid overflow. The equation of the cumulative function is referred to (4.2). Refer back to Section 4.3.2 for more details.

```
pzipfmb <-
  function(q, shape, start = 1, lower.tail = TRUE, log.p = FALSE) {
    if (!is.logical(lower.tail) || length(lower.tail) != 1)
      stop("bad input for argument 'lower.tail'")
    if (!is.logical(log.p) || length(log.p) != 1)
      stop("bad input for argument 'log'")

    LLL <- max(length(shape), length(q), length(start))
    if (length(q)     != LLL) q     <- rep_len(q,     LLL)
    if (length(shape) != LLL) shape <- rep_len(shape, LLL)
    if (length(start) != LLL) start <- rep_len(start, LLL)
    q.use <- pmax(start, floor(q + 1))

    bad0 <- shape <= 0 | 1 <= shape | !is.finite(start) |
      start != round(start) | start < 1
    bad <- bad0 | !is.finite(q.use)
    ans <- q
    ans[TRUE] <- NA_real_  # if (log.p) log(0) else 0

    # This is short because of [!bad] is selective. It is
    # the PLOSone formula for S.
    log.S.short <- lgamma(start[!bad]) +
                   lgamma(q.use[!bad] - shape[!bad]) -
                   lgamma(q.use[!bad]) -
                   lgamma(start[!bad] - shape[!bad])

    ans[!bad] <-
      if (lower.tail) {
        if (log.p) { # lower.tail = T, log.p = T
          log1p(-exp(log.S.short))
        } else {  # lower.tail = T, log.p = F
          -expm1(log.S.short)
        }
      } else {
        if (log.p) {  # lower.tail = F, log.p = T
          log.S.short
        } else {      # lower.tail = F, log.p = F
          exp(log.S.short)
        }
      }

    # This handles q==Inf:
    ans[!is.finite(q.use) & start < q.use & !bad0] <-
      if (lower.tail) { if (log.p) log(1) else 1} else
      { if (log.p) log(0) else 0}

    ans
  }  # pzipfmb()
```

A.3 Quantile Function qzipfmb()

Bisection method is applied to approximate the root. Refer back to Section 4.3.3 for more details.

```
qzipfmb <- function (p, shape, start = 1) {

  LLL <- max(length(p), length(shape), length(start))
  if (length(p)     != LLL) p     <- rep_len(p,     LLL)
  if (length(shape) != LLL) shape <- rep_len(shape, LLL)
  if (length(start) != LLL) start <- rep_len(start, LLL)

  ans <- rep_len(start, LLL)

  # First, bracket the solution between 'lo' and 'hi'.
  lo <- rep_len(start, LLL) - 0.5
  approx.ans <- lo  # True at lhs
  hi <- 2 * lo + 10
  dont.iterate <- p == 1 | shape <= 0 | 1 <= shape |
    start != round(start) | start < 1
  done <- p <= pzipfmb(hi, shape, start = start) |
    dont.iterate
  max.iter <- 50
  iter <- 0

  while (!all(done) && iter < max.iter) {
    hi.save <- hi[!done]
    hi[!done] <- 2 * lo[!done] + 10
    lo[!done] <- hi.save
    done[!done] <- (p[!done] <= pzipfmb(hi[!done],
                                        shape = shape[!done],
                                        start[!done]))
    iter <- iter + 1
  }

  # Second, solve for the root.
  foo <- function(q, shape, start, p)
```

```
    pzipfmb(q, shape = shape, start) - p

  lhs <- (p <= dzipfmb(start, shape = shape, start)) |
    dont.iterate

  approx.ans[!lhs] <-
    bisection.basic(foo, lo[!lhs], hi[!lhs], tol = 1/16,
                    shape = shape[!lhs],
                    start = start[!lhs],
                    p = p[!lhs])
  faa <- floor(approx.ans)
  ans <- ifelse(pzipfmb(faa, shape = shape, start) < p &
                  p <= pzipfmb(faa+1, shape = shape, start),
                faa+1, faa)

  # Some special cases:
  ans[p == 1] <- Inf
  ans[shape <= 0 | 1 <= shape] <- NaN

  ans
}  # qzipfmb
```

A.4 Random Deviates Generator `rzipfmb()`

The transformation method is used to generate random deviates from the Zipf-MandelbrotII distribution. Refer back to Section 4.3.4 for more details.

```
rzipfmb <- function(n, shape, start = 1) {
  # Using transformation method
  qzipfmb(runif(n), shape, start)
} # rzipfmb
```

A.5 Family function `zipfmbrot()`

This is the full implementation of the family function `zipfmbrot` depending on the VGAM R package. Refer back to Section 4.2 for more details.

A.5.1 `@blurb`

`@blurb` consists of a character string with descriptive information of the family function, including the name, the type of link functions, mean and variance if applicable.

```
blurb = c("Mandelbrot distribution (Type III) \n",
          "Link:    ",
          namesof("shape", lshape, earg = eshape),
          "\n\n",
          "Mean: \n",
          "Variance: "),
```

A.5.2 `@info`

`@infos` returns a list with, e.g. M1(M1), Q1(Q1) and logical `multipleResponses`, etc.

```
infos = eval(substitute(function(...) {
       list(M1 = 1,
               Q1 = 1,
               expected = TRUE,
               multipleResponses = TRUE,
               start = .start,
               parameters.names = "shape",
               zero  = .zero)
        }, list( .zero = zero,
                 .lshape = lshape,
                 .start = start
        ))),
```

A.5.3 `@linkinv`

`@linkinv` returns fitted values, e.g., a matrix with rows mu_i^T.

```
linkinv = function(eta, extra = NULL) exp(-eta)
```

A.5.4 `@constraints`

`@constraints` processes the constraint matrices. This comprises the (i) constraints argument, and (ii) arguments such as parallel and zero.

```
initialize = eval(substitute(expression({
          start <- .start
          temp5 <-
            w.y.check(w = w, y = y,
                      ncol.w.max = Inf,
                      ncol.y.max = Inf,
                      Is.integer.y = TRUE,
                      Is.positive.y = TRUE,
                      out.wy = TRUE,
                      colsyperw = 1,
                      maximize = TRUE)
          w <- temp5$w
          y <- temp5$y
          if (any(y < start))
            stop("some response values less than 'start'")

          predictors.names <-
            namesof("shape", .lshape, earg = .eshape, tag = FALSE)

          extra$start <- start
          if (!length(etastart)) {
            llfun <- function(shape, y, start, w) {
              sum(c(w) * dzipfmb(x = y, shape = .shape,
                                 start = extra$start, log = TRUE))
            }
            shape.init <- .ishape
            etastart <- theta2eta(shape.init, .lshape , earg = .eshape )
          }
        }), list( .lshape = lshape,
                  .eshape = eshape, .ishape = ishape, .start = start ))),

        # Log-likelihood of distribution

        loglikelihood = eval(substitute(
          function(mu, y, w, residuals = FALSE, eta,
                   extra = NULL,
                   summation = TRUE) {
            shape <- eta2theta(eta, .lshape , earg = .eshape )
            if (residuals) {
              stop("loglikelihood residuals not implemented yet")
            } else {
              ll.elts <- c(w) * dzipfmb(y = x, shape = .shape,
                                        start = .start, log = TRUE)
              if (summation) {
                sum(ll.elts)
              } else {
                ll.elts
              }
            }
          }, list( .lshape = lshape, .eshape = eshape ))),
```

A.5.5 `@last`

`@last` assigns `misclink`, `miscearg` and other information; evaluated after final IRLS iteration.

```
last = eval(substitute(expression({
            misc$expected <- FALSE
            misc$link <- c(shape = .lshape )
            misc$earg <- list(shape = .eshape )
            misc$start <- extra$start
        }), list( .lshape = lshape, .eshape = eshape ))),
```

A.5.6 `@loglikelihood`

`@loglikelihood` returns l, or $[(l)_l s]$ if summation = FALSE

```
loglikelihood = eval(substitute(
          function(mu, y, w, residuals = FALSE, eta,
                   extra = NULL,
                   summation = TRUE) {
            shape <- eta2theta(eta, .lshape , earg = .eshape )
            if (residuals) {
              stop("loglikelihood residuals not implemented yet")
            } else {
              ll.elts <- c(w) * dzipfmb(y = x, shape = .shape,
                                        start = .start, log = TRUE)
              if (summation) {
                sum(ll.elts)
              } else {
                ll.elts
              }
            }
          }, list( .lshape = lshape, .eshape = eshape ))),
```

A.5.7 `@vfamily`

`@vfamily` serves as an identification of the name of the family function.

```
vfamily = "mandelbrotff",
```

A.5.8 `@deriv`

`@deriv` returns an n x M matrix of score vectors; has rows wi...,

```
deriv = eval(substitute(expression({
          shape <- eta2theta(eta, .lshape, earg = .eshape )

          # break down to derive dl.dshape
          constant <- digamma(start) - digamma(y+1)
          term1 <- 1/(shape-1)
          term2 <- digamma(y+1-shape)
          term3 <- digamma(start+1-shape)

          dl.dshape <- constant + term1 + term2 + term3
          dshape.deta <- dtheta.deta(shape, .lshape, earg = .eshape )
          c(w) * dl.dshape * dshape.deta
        }), list( .lshape = lshape, .eshape = eshape ))),
```

A.5.9 `@weight`

```
        weight = expression({
          ned2l.dshape2 <- -(1/(shape-1)^2) +
            trigamma(y+1-shape) +
            trigamma(start+1-shape)
          wz <- c(w) * dshape.deta^2 * ned2l.dshape2
```

`@weight` computes working weights `wz`, `wW`, in matrix-band format. It should be evaluated immediately after `@deriv` so that assigned variables can be sued without interruption. Optional slots such as `@devian`, `@first`, `@linkfun` and `@simslot` may also be added into the implementation.

Appendix B

Background Material

Some further background material and details follow.

B.1 On Zipf and Zeta Distributions

Norman L. Johnson (1993) [pp.465-471] contains a good survey of distributions to be considered in this thesis. Other useful references include Baayen (2001) and Piantadosi (2014). Note that another zeta-variant, Haight's zeta distribution, also exists in VGAM (called `hzeta()`). Haight's zeta distribution arises as the limiting distribution to Zipf's conjecture concerning city sizes Simon (1955). It has PMF

$$\Pr(Y = y; \alpha) \;=\; (2y-1)^{-\alpha} \;-\; (2y+1)^{-\alpha}, \qquad y = 1, 2, \ldots, \qquad \alpha > 0.$$

It gets its name because $E(Y) = (1-2^{-\alpha})\,\zeta(\alpha)$ and $\mathrm{Var}(Y) = (1-2^{1-\alpha})\,\zeta(\alpha-1) - \mu^2$. Note that for $\alpha \leq 1$ the mean is infinite, and for $\alpha \leq 2$ the variance is infinite. More information can be found in, e.g., Norman L. Johnson (2005) [pp.533-4].

Haight (1966) used Haight's zeta distribution very successfully on four data sets on word associations. Haight (1966) investigation was concerned with models for word association data (the number of response words elicited by a stimulus word). He applied the Yule, Borel-Tanner and logarithmic distributions—these have been implemented in VGAM.

B.2 The Riemann and Hurwitz Zeta Functions

The Riemann formula for s is

$$\sum_{n=1}^{\infty} \frac{1}{n^s} \tag{B.1}$$

While the usual definition involves an infinite series that converges when the real part of the argument is > 1, more efficient methods have been devised to compute the value. In particular, this function uses Euler-Maclaurin summation.

B.3 The Hurwitz Zeta Function

The Hurwitz ζ function is defined for complex arguments and is

$$\zeta(s,\ q) \ = \ \sum_{n=0}^{\infty} (n+q)^{-s}, \qquad \Re(s) > 1, \tag{B.2}$$

with $\Re(q) > 0$. Hence $\zeta(s,\ 1)$ is the ordinary Riemann zeta function

$$\zeta(s) \ = \ \sum_{n=1}^{\infty} n^{-s}, \qquad \Re(s) > 1. \tag{B.3}$$

The Hurwitz ζ function can be used to define generalizations of the ordinary zeta and Zipf distributions. For example, Moreno-Sanchez et al. (2016) consider a random variable defined on $a(1)\infty$ [notation: $a(b)c = \{a, a+b, a+2b, \ldots, c\}$] based on $\zeta(s, a)$—although $a = 1$ usually, it is not always so with word-studies data. We wish to look at estimating a as a positive real-valued parameter Section 2.1.3, with the support of the distribution taken as $1(1)\infty$.

Bibliography

Ali, K. & Scarr, M. (2007), Robust methodologies for modeling web click distributions, ACM, New York, NY, USA, pp. 511–520.
URL: *http://doi.acm.org/10.1145/1242572.1242642*

Allan, J., Papka, R. & Lavrenko, V. (1998), On-line new event detection and tracking, ACM, New York, NY, USA, pp. 37–45.
URL: *http://doi.acm.org/10.1145/290941.290954*

Ananiadou, S. & Mcnaught, J. (2005), *Text Mining for Biology And Biomedicine*, Artech House, Inc., Norwood, MA, USA.

Axtell, R. (2001), 'Zipf distribution of u.s. firm sizes', *Science (New York, N.Y.)* **293**, 1818–20.

Baayen, H. (2001), *Word Frequency Distributions*, Springer Netherlands.

Camacho, J. & Sole, R. V. (2001), 'Scaling in ecological size spectra', *EPL (Europhysics Letters)* **55**(6), 774.
URL: *http://stacks.iop.org/0295-5075/55/i=6/a=774*

Chakrabarti, S. (2002), *Mining the Web: Discovering Knowledge from HyperText Data*, Science & Technology Books.

Chambers, J. M. (1998), *Programming with Data: A Guide to the S Language*, 1st edn, Springer-Verlag, Berlin, Heidelberg.

Clauset, A., Shalizi, C. & Newman, M. (2009), 'Power-law distributions in empirical data', *SIAM Review* **51**(4), 661–703.

Deluca, A. & Corral, A. (2013), 'Fitting and goodness-of-fit test of non-truncated and truncated power-law distributions', *Acta Geophysica* **61**(6), 1351–1394.
URL: *https://doi.org/10.2478/s11600-013-0154-9*

Ebel, H., Mielsch, L.-I. & Bornholdt, S. (2002), 'Scale-free topology of e-mail networks', *Phys. Rev. E* **66**, 035103.
URL: *https://link.aps.org/doi/10.1103/PhysRevE.66.035103*

Furusawa, C. & Kaneko, K. (2003), 'Zipf's law in gene expression', *Phys. Rev. Lett.* **90**, 088102.
URL: *https://link.aps.org/doi/10.1103/PhysRevLett.90.088102*

Gerlach, M. & Font-Clos, F. (2018), 'A standardized project gutenberg corpus for statistical analysis of natural language and quantitative linguistics', *CoRR* **abs/1812.08092**.

Gilleland, E., Ribatet, M. & Stephenson, A. G. (2013), 'A software review for extreme value analysis', *Extremes* **16**(1), 103–119.
URL: *https://doi.org/10.1007/s10687-012-0155-0*

Goldstein, M. L., Morris, S. A. & Yen, G. G. (2004), 'Problems with fitting to the power-law distribution', *The European Physical Journal B - Condensed Matter and Complex Systems* **4**(2), 255–258.

Haight, F. A. (1966), 'Some statistical problems in connection with word association data', *Journal of Mathematical Psychology* **3**(1), 217 – 233.

Haro, M., Serra, J., Herrera, P. & Corral, A. (2012), Zipf's law in short-time timbral codings of speech, music, and environmental sound signals.

Lawless, J. F. (1987), 'Negative binomial and mixed poisson regression', *Canadian Journal of Statistics* **15**(3), 209–225.

Li, W. (2002), 'Zipf's law everywhere', *Glottometrics* **5**.

Li, W., Miramontes, P. & Germinal, C. (2010), 'Fitting ranked linguistic data with two-parameter functions', *Entropy* **12**.

Mandelbrot, B. (1961), *On the Theory of Word Frequencies and on Related Markovian Models of Discourse.*

Mandelbrot, B. B. & Wallis, J. R. (1968), 'Noah, joseph, and operational hydrology', *Water Resources Research* **4**(5), 909–918.

Moreno-Sanchez, I., Font-Clos, F. & Corral, A. (2016), 'Large-scale analysis of zipf's law in english texts', *PLOS ONE* **11**.

Nelder, J. A. & Wedderburn, R. W. M. (1972), 'Generalized linear models', *Journal of the Royal Statistical Society. Series A (General)* **135**(3), 370–384.
URL: *http://www.jstor.org/stable/2344614*

Newman, D., Chemudugunta, C., Smyth, P. & Steyvers, M. (2006), Analyzing entities and topics in news articles using statistical topic models, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 93–104.

Newman, M. E. J. & Park, J. (2003), 'Why social networks are different from other types of networks', *Phys. Rev. E* **68**, 036122.
URL: *https://link.aps.org/doi/10.1103/PhysRevE.68.036122*

Norman L. Johnson, Samuel Kotz, A. W. K. (1993), *Univariate Discrete Distributions*, Wiley-Interscience, Hoboken, USA.

Norman L. Johnson, Samuel Kotz, A. W. K. (2005), *Univariate Discrete Distributions*, Wiley-Interscience, Hoboken, USA.

Piantadosi, S. (2014), 'Zipf's word frequency law in natural language: A critical review and future directions', *Psychonomic bulletin review* **21**.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992), *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA.

Project Gutenberg (2019).
URL: *https://www.gutenberg.org*

Pueyo, S. & Jovani, R. (2006), 'Comment on "a keystone mutualism drives pattern in a power function"', *Science* **313**(5794), 1739–1739.
URL: *http://science.sciencemag.org/content/313/5794/1739.3*

R Core Team (2018), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org/*

Sani, S. & Daman, O. (2014), 'Mathematical modeling in heavy traffic queuing systems', *American Journal of Operations Research* **4**, 340–350.

Seal, H. L. (1952), 'The maximum likelihood fitting of the discrete pareto law', *Journal of the Institute of Actuaries* **78**(1).

Serra, J., Corral, A., Boguna, M., Haro, M. & Arcos, J. L. (2012), 'Measuring the evolution of contemporary western popular music', *CoRR* **abs/1205.5651**.
URL: *http://arxiv.org/abs/1205.5651*

Simon, H. A. (1955), 'On a class of skew distribution functions', *Biometrika* **42**(3-4), 425–440.

White, E. P., Enquist, B. J. & Green, J. L. (2008), 'On estimating the exponent of power-law frequency distributions', *Ecology* **89**(4), 905–912.

Yee, T. W. (2010), 'The VGAM Package for Categorical Data Analysis', *Journal of Statistical Software* .

Yee, T. W. (2013), 'Two-parameter reduced-rank vector generalized linear models', *Computational Statistics and Data Analysis* .
URL: *http://ees.elsevier.com/csda*

Yee, T. W. (2015), *Vector Generalized Linear and Additive Models: With an Implementation in R*, Springer, New York, USA.

Yee, T. W. (2019), 'The VGAM package'.
URL: *http://cran.r-project.org/package=VGAM*

Yu, S., Xu, C. & Liu, H. (2018), 'Zipfs law in 50 languages: its structural pattern, linguistic interpretation, and cognitive motivation', *CoRR* **abs/1807.01855**.
URL: *http://arxiv.org/abs/1807.01855*

Zipf, G. K. (1932), *Selected Studies of the Principle of Relative Frequency in Language.*, Harvard University Press.