Some Problems Concerning the Generalized Hyperbolic Distribution

By

XINXING LI

Department of Statistics
UNIVERSITY OF AUCKLAND

A thesis submitted in fulfilment of the requirements for the degree of DOCTOR OF PHILOSOPHY in Statistics, the University of Auckland, 2015.

Abstract

Generalized Hyperbolic distribution and its family were recoginized by researchers for the valuable non-Gaussian properties that are applicable in almost all areas of finance and risk management. However due to the complexity of the distribution, there are several problems revealed during implement the distribution to statistical application, in particular the `R` language. The primary aim of my research is to identify those problems, investigate both theoretical and computational solutions. The basic function that base `R` provides for most of the classical distribution are probability distribution function (p), density function (d), quantile function (q) and random number generation (r). For generalized hyperbolic distribution and its family, we also followed the same rule. The current estimation approaches for probability distribution function (CDF) and quantile functions of GHyp distribution and skew hyperbolic student's t distribution which is the limiting case are found to be unstable, inaccurate and lack of efficiency. The problems of GHyp distribution can be solved computationally whilst the problems of the skew hyperbolic student's t distribution requires proposing a new theoretical method, i.e. the split t transformation method.

Besides the problems with CDF and quantile function estimation approach, there are concerns with the current generalized inverse Gaussian distribution random number generation approach which is closely related to GHyp distribution random variates generator. There are few remedies of the concerns has been proposed but not yet implemented include rejection method using either gamma distributed hat function or a two/three parts hat function. These approaches are implemented and compared with the current implemented approaches. Besides the generalized inverse Gaussian distribution, we also investigate the random number generation approach of the hyperbolic distribution which is the sub-case of the GHyp distribution.

Apart from the basic functionality of the GHyp distribution, the other prospective of my research was to overhaul the current algorithm and functions for linear modeling using hyperbolic distribution. We also developed a set of functions include but not limited to `plot` function, `summary` function in order to have a degree of consistency with the existing model fitting procedures in `R`. These functions are demonstrated with some real data examples. The algorithm is then compared with several robust modeling techniques using those examples.

Dedication and acknowledgements

I take this opportunity to thank my supervisor Associate Prof. Dr. David Scott for his encouragement, confidence and always reliable support. I appreciate all his contributions of times, ideas to make my Ph.D. experience productive and stimulating. I would also like to thank the IT team for all the IT assistance they have provided and all the staff from Department of Statistics for their help throughout my Ph.D. study. Lastly, I would like to thank my family for all their encouragement and faithful support during the this Ph.D.

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

RESEARCH BACKGROUND

1.1 Overview

Theoretical distributions play a critical role in statistics as they model the pattern of variation. There are numerous theoretical distributions developed and these distribution can be categorized as discrete distributions or continuous distributions with respect to the random variate type. They can also be categorized as univariate distributions and multivariate distributions with respect to the dimension of the random variates.

My research is focused on continuous univariate distributions as continuous distributions are generally amenable to more elegant mathematical treatment than are discrete distributions ([72]). The most common univariate continuous distributions in practice have been systematically summarized in [72] and [73]. These distributions have been classified to 19 families, normal distributions, gamma distributions, Weibull distributions to name a few, in accordance to the characteristics which are considered as one of the important model selection criteria to explain the pattern of variation in the data.

Because of the non-Gaussian characteristics of the data in regard to financial market volatility, skewed and semi-heavy or heavy tailed distributions have often been proposed for applications in finance to capture the non-Gaussian characteristics. These distributions include but are not limited to: the Student's t distribution which can exhibit heavy and semi-heavy tail behaviour; and the normal-Laplace distribution and generalized normal-Laplace distribution discussed in [87] which exhibit semi-heavy tail behaviour.

Among these distributions, implementations of Student's t distribution, and in particular its skewed extensions, are commonly discussed in the literature. The asymmetric Student's t distribution was first proposed for finance applications in [55] when extending the ARCH model. Skewed extensions were then considered as a preferred remedy when the normality assumption

is violated and have been discussed in several papers, [66], [37], [109], [81], [58], [112], to name a few. However the skew t type of most of the discussions feature two identical polynomial tail rate decays, and in consequence the skewness of those proposed distributions is limited. There are two exceptions, the first one is discussed in [66] where the skew t distribution has two parameters to control the left and right tails respectively. The second one is discussed in [112] where the skew t distribution has an additional third parameter to control the skewness.

One alternative to the skew Student's t distributions are the distributions of the generalized hyperbolic (GHyp) family which also have valuable non-Gaussian properties, in particular skewness and semi-heavy tails, but have been less discussed. Therefore the primary aim of this thesis is to address and provide solutions to computational problems arising in the use of the family of univariate GHyp distributions in statistical applications in the environment of `R` in particular.

There are approximately 120 packages dealing with distributions implemented in *The Comprehensive R Archive Network* (CRAN) apart from the *GeneralizedHyperbolic* and the *SkewHyperbolic* packages. These packages deal with various aspects of distributions which include but are not limited to discrete distributions, continuous distributions, mixtures of probability laws, random matrices, copulas, random number generation, moments, skewness, and kurtosis.

Of these packages, there are two packages worth mentioning here. The first one is the *distr* package which provides a conceptual treatment of distributions by means of S4 classes ([31]). In particular, there are general functions `RtoDPQ`, `RtoDPQ.d` and `RtoDPQ.LC` in the *distr* package which approximate the density, cumulative distribution (CDF) and quantile function from random numbers for absolute continuous, discrete and Lebesgue decomposed distributions respectively. The results however are empirical as the functions use simulation to do the approximation. In Chapter 2, we describe some utility functions contained in the package *DistributionUtils*. These utility functions consist of general routine functions that calculate the CDF and quantile function for any unimodal univariate continuous distribution by numerical integration and root finding. The results obtained are not based on simulation and have specified levels of accuracy. Although the main intention has been to develop functions for distributions which conform to the standard approach suggested in [95], the package itself was developed in part to support the packages *GeneralizedHyperbolic* [91], *VarianceGamma* [92], *SkewHyperbolic* [94] and *NormalLaplace* [93] which implement functions to work with respectively the generalized hyperbolic, variance gamma, skew hyperbolic Student's t and normal Laplace distributions.

The second package to consider is the *Runuran* package. This package provides an interface to the UNU.RAN library for universal non-uniform random number generators ([80]). It provides a collection of algorithms for generating non-uniform pseudorandom variates as a library of C functions ([77]). The package provides random number generators for both continuous distributions, including the GHyp distribution, and discrete distributions as well as multivariate continuous distributions. For each type of distribution, there are several random number generation methods implemented. For instance: adaptive rejection sampling implemented in the `ars.new` function; in-

verse transformed density rejection implemented in `itdr.new` function; polynomial interpolation of the inverse CDF implemented in the `pinv.new` function; simple ratio-of-uniforms implemented in the `srou.new` function; and transformed density rejection implemented in the `tdr.new` function; all for univariate continuous distributions. Of these random number generators, of most interest is the polynomial interpolation of inverse CDF as it is not only an algorithm for random number generation but also for quantile function approximation. This algorithm is thoroughly reviewed in Chapter 4.

1.1.1 The Generalized Hyperbolic Distribution

The GHyp distribution family and its subclasses were introduced by O. Barndorff-Nielsen in his paper for modeling the particle-size distribution of windblown sand ([9]). It was then recognized that this distribution's properties would be useful for financial asset distributions which have non-Gaussian characteristics. For instance, it is well known that the returns of most financial assets have semi-heavy tails and feature skewness. Since the GHyp distribution is obtained as a variance-mean mixture of the normal distribution, it possesses semi-heavy tails and exhibits skewness for certain parameter values ([86]).

The GHyp distribution is also very flexible and has a number of important statistical distributions as members or limiting cases: the normal distribution; the Student's t distribution; the hyperbolic distribution; the normal inverse Gaussian (NIG) distribution and the Laplace distribution; to name a few. A comprehensive discussion regarding these distributions and their subclasses is given in [85]. In [22], the application of these distributions in finance has also been reviewed in detail. Of those, the hyperbolic, NIG and skew hyperbolic Student's t distributions are in particular of interest in this research.

The hyperbolic distribution attracted a number of researchers' interest after it was introduced in [9]. Some of its properties, such as infinite divisibility, were then presented in [15], [8] and [53]. The maximum likelihood estimation of the distribution was considered in some detail in [12]. [40] extensively discussed fitting empirical financial returns with high accuracy using the fact that the log density of hyperbolic distribution is a hyperbola whilst the log density of the normal distribution is a parabola. In that paper, a hyperbolic Lévy motion is applied as a model of stock returns. [41] investigated further empirical evidence that the hyperbolic model is superior to the well-known α-stable model for log return data. Both papers argue that the hyperbolic distribution provides an alternative approach to model heavy tailed return data by means of a Lévy process. [21] then proposed a hyperbolic diffusion model for stock prices. [74] in addition illustrated the fitting of the hyperbolic distribution to the daily returns of the German stocks that have been included in DAX during the period 1974 to 2002. Just as statistical modeling of financial returns by means of the hyperbolic distribution has been shown in the literature to be effective, the NIG distribution appears be an attractive alternative distribution.

Similar to the hyperbolic distribution, a mathematically simpler NIG Lévy process is proposed

in [10] and it is demonstrated that the NIG distribution can approximate most hyperbolic distributions closely. The NIG distribution has then been discussed extensively in [11], [19], [88] and [69]. Recently, there are a number of applications derived using the distribution, including [24], [39], [56], [20], [68], [44], [105].

Unlike these first two subclass distributions, the skew hyperbolic Student's t distribution has not been investigated as much. It was first briefly mentioned by [86]. Then it was briefly discussed in several papers include [16], [66], [22], [36] and [82]. Subsequently K. Aas and I. H. Haff provided a comprehensive discussion regarding the distribution in [1]. They also argued that the special behavior of its tails which could be an alternative for modeling skewed and heavy tailed data.

Apart from the subclass distributions, the proper generalized hyperbolic distribution is also considered in some research. The proper generalized hyperbolic distribution is often briefly discussed in the literature of the subclass distributions, [8] and [86] in particular. In the area of applications, [86] and [39] extended the hyperbolic Lévy process to the generalized hyperbolic Lévy process while [89] proposed the generalized hyperbolic diffusion model for finance.

Before giving a definition of the Ghyp distribution, it is necessary to introduce the generalized inverse Gaussian distribution (GIG). This distribution was introduced by [51]. However, there was not much discussion of the distribution until it was used by [100] to construct a compound Poisson distribution. The moments and density function are given by [23]. Some of the probabilistic properties including infinite divisibility and self-decomposability were shown by [15] and [53] respectively. The distribution was then reviewed and discussed extensively in [67]. [43] has shown an asymptotic convolution property of the distribution then applied it to calculate the probability of ruin in the general risk model and discussed some related applications. In recent years, the distribution has been applied to model neural activity in [64] using the results from [17].

Definition 1. *The GIG distribution has three parameters λ, χ, ψ. The probability density function is*

$$f(x) = \frac{1}{K_\lambda(\chi,\psi)} x^{\lambda-1} \exp\left[-\frac{1}{2}(\chi x^{-1} + \psi x)\right] I_{(0,-\infty)}(x) \tag{1.1}$$

where K_λ is the modified Bessel function of the third kind with order λ, $\lambda \in \mathbb{R}$, $\chi > 0$ and $\psi > 0$. When $\lambda > 0$, $\chi = 0$ and $\psi > 0$, the GIG distribution reduces to the gamma distribution. When $\lambda < 0$, $\chi > 0$ and $\psi = 0$, the distribution becomes the inverse gamma distribution.

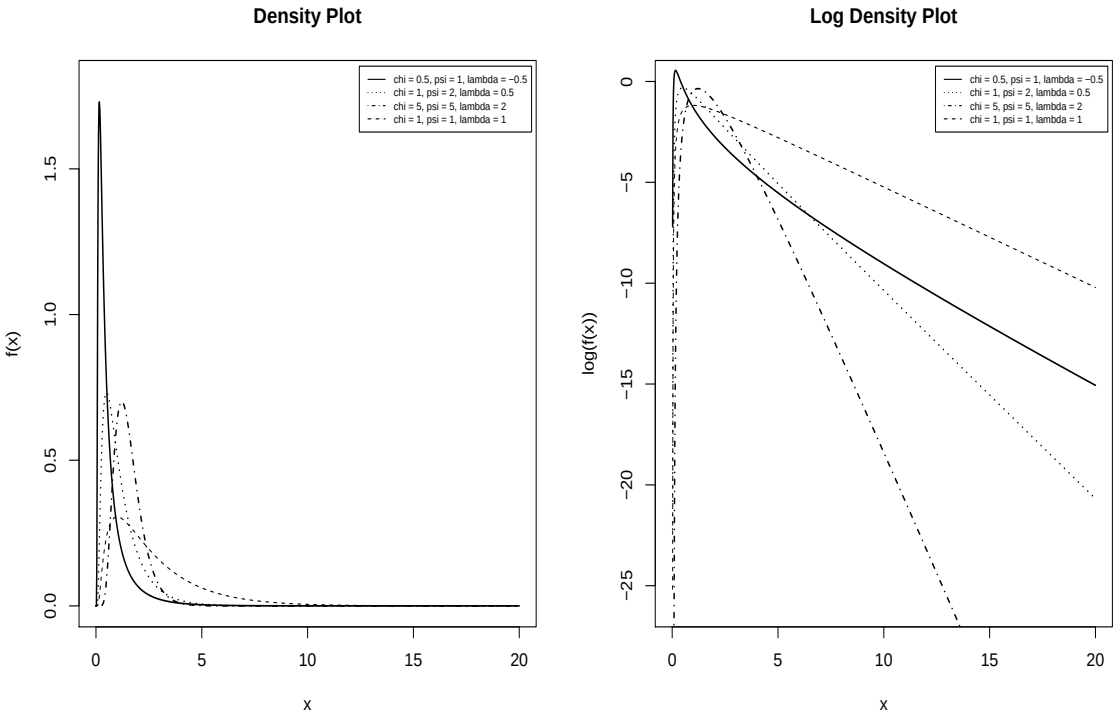

Figure 1.1: A selection of GIG distributions with various χ, ψ and λ

There are at least 4 parameterizations of the GIG distribution in the literature ([67] and [33]) which are

1. λ, χ, ψ

2. λ, δ, γ

3. λ, α, β

4. λ, ω, η

Table 1.1: Relationships between four parameterizations of the GIG distribution at fixed values of μ and δ.

	To: Parameterization			
From: Parameterization	1	2	3	4
	λ, χ, ψ	λ, δ, γ	λ, α, β	λ, ω, η
1. λ, χ, ψ ($\chi > 0, \psi > 0$)		$\delta = \sqrt{\chi}$ $\gamma = \sqrt{\psi}$	$\alpha = \sqrt{\psi/\chi}$ $\beta = \sqrt{\chi\psi}$	$\omega = \sqrt{\chi\psi}$ $\eta = \sqrt{\chi/\psi}$
2. λ, δ, γ ($\gamma > 0, \delta > 0$)	$\chi = \delta^2$ $\psi = \gamma^2$		$\alpha = \gamma/\delta$ $\beta = \gamma\delta$	$\omega = \delta\gamma$ $\eta = \delta/\gamma$
3. λ, α, β ($\alpha > 0, \beta > 0$)	$\chi = \beta/\alpha$ $\psi = \alpha\beta$	$\delta = \sqrt{\beta/\alpha}$ $\gamma = \sqrt{\alpha\beta}$		$\omega = \beta$ $\eta = 1/\alpha$
4. λ, ω, η ($\omega > 0, \eta > 0$)	$\chi = \omega\eta$ $\psi = \omega/\eta$	$\delta = \sqrt{\omega\eta}$ $\gamma = \sqrt{\omega/\eta}$	$\alpha = 1/\eta$ $\beta = \omega$	

The moment generating function can be shown as

$$\mathrm{M}(t) = \frac{\mathrm{K}_\lambda(\chi, \psi - 2t)}{\mathrm{K}_\lambda(\chi, \psi)} \tag{1.2}$$

The mean and variance are therefore

$$\mathrm{E}(X) = \frac{\mathrm{K}_{\lambda+1}(\chi, \psi)}{\mathrm{K}_\lambda(\chi, \psi)} \tag{1.3}$$

$$\mathrm{Var}(X) = \frac{\mathrm{K}_\lambda(\chi, \psi)\mathrm{K}_{\lambda+2}(\chi, \psi) - (\mathrm{K}_{\lambda+1}(\chi, \psi))^2}{(\mathrm{K}_\lambda(\chi, \psi))^2} \tag{1.4}$$

The description of the GHyp distribution starts with the definition of the distribution class followed by the relevant subclasses and limiting distributions.

Definition 2. *The generalized hyperbolic distributions GHyp$(\lambda, \alpha, \beta, \delta, \mu)$ are defined as a generalized inverse Gaussian (GIG) distribution-variance-mean mixture of the normal distribution, i.e.*

$$\mathrm{GHyp}(\lambda, \alpha, \beta, \delta, \mu) := \mathrm{Mix}_{\mathrm{GIG}(\lambda, \delta^2, \alpha^2 - \beta^2)}(\mu, \beta) \tag{1.5}$$

The parameters $\lambda, \mu \in \mathbb{R}$, $\alpha, \delta \geq 0$, $|\beta| \leq \alpha$ must satisfy the following conditions:

- $|\beta| < \alpha$ and $\delta > 0$ if $\lambda = 0$
- $|\beta| < \alpha$ and $\delta \geq 0$ if $\lambda > 0$
- $|\beta| = \alpha$ and $\delta > 0$ if $\lambda < 0$

The GHyp distribution investigated in the research, which appears from next chapter onwards, refers to the proper GHyp distribution defined as follows.

Definition 3. *The univariate proper GHyp distribution has five parameters λ, α, β, μ and δ. The probability density function is*

$$f(x) = \frac{(\sqrt{\alpha^2-\beta^2}/\delta)^\lambda}{\sqrt{2\pi}K_\lambda(\delta\sqrt{\alpha^2-\beta^2})} e^{\beta(x-\mu)} K_{\lambda-1/2}(\alpha\sqrt{\delta^2+(x-\mu)^2}) \tag{1.6}$$

where K_λ is the modified Bessel function of the third kind with order λ and $\lambda, \mu \in \mathbb{R}$, $\alpha, \delta > 0$ and $|\beta| < \alpha$

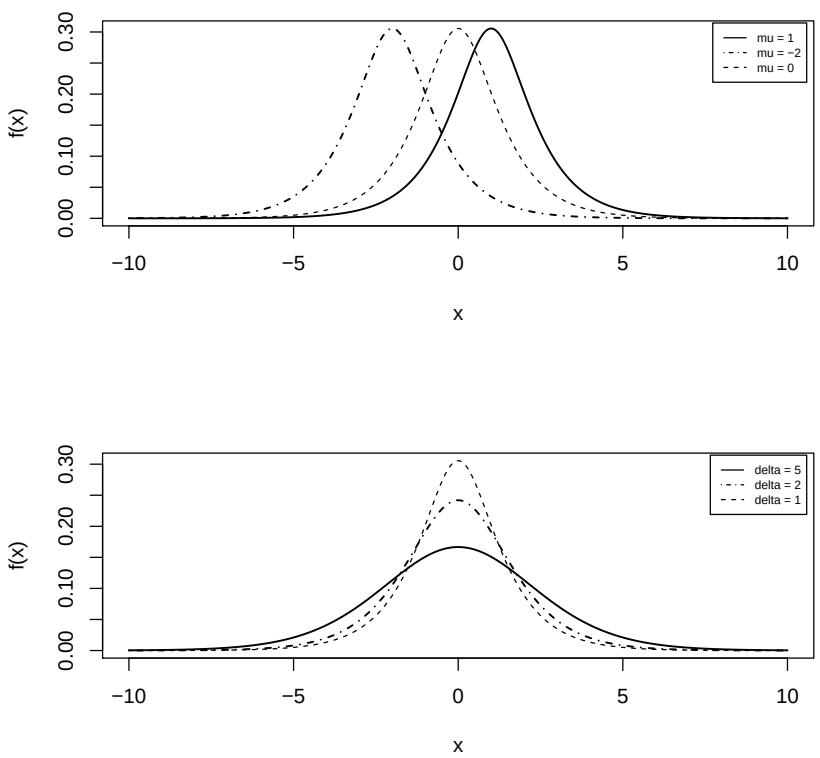

Figure 1.2: A selection of GHyp distributions with various μ and δ when $\alpha = 1$, $\beta = 0$ and $\lambda = 1$

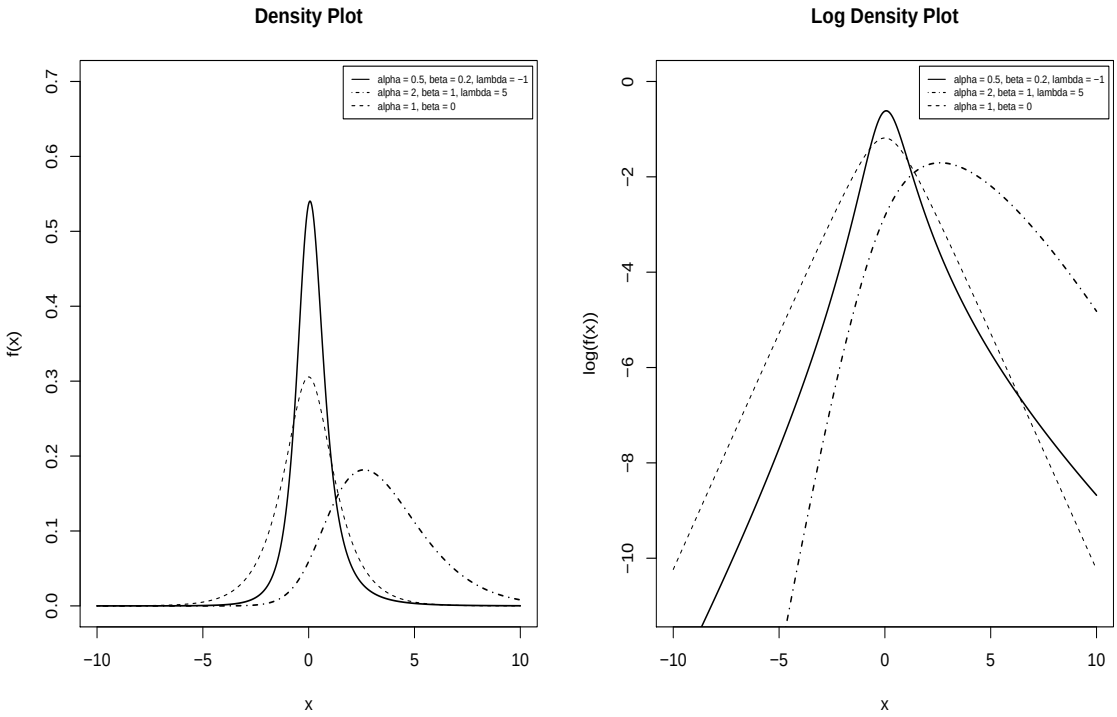

Figure 1.3: A selection of GHyp distributions with various α, β and λ when $\mu = 0$ and $\delta = 1$

The moment generating function can be shown to be

$$\mathrm{M}(t) = e^{t\mu}\left(\frac{\alpha^2-\beta^2}{\alpha^2-(\beta+t)^2}\right)^{\lambda/2}\frac{\mathrm{K}_\lambda(\delta\sqrt{\alpha^2-(\beta+t)^2})}{\mathrm{K}_\lambda(\delta\sqrt{\alpha^2-\beta^2})} \tag{1.7}$$

Therefore the mean and variance of the proper GHyp distribution are derived as

$$\mathrm{E}(X) = \mu + \frac{\beta\delta}{\sqrt{\alpha^2-\beta^2}}\frac{\mathrm{K}_{\lambda+1}(\zeta)}{\mathrm{K}_\lambda(\zeta)} \tag{1.8}$$

$$\mathrm{Var}(X) = \delta^2\left(\frac{\mathrm{K}_{\lambda+1}(\zeta)}{\zeta\mathrm{K}_\lambda(\zeta)} + \frac{\beta^2}{\alpha^2-\beta^2}\left(\frac{\mathrm{K}_{\lambda+2}(\zeta)}{\mathrm{K}_\lambda(\zeta)} - \left(\frac{\mathrm{K}_{\lambda+1}(\zeta)}{\mathrm{K}_\lambda(\zeta)}\right)^2\right)\right) \tag{1.9}$$

where $\zeta = \delta\sqrt{\alpha^2-\beta^2}$. With the Bessel function involved, the density function can only be integrated numerically which means neither the CDF nor the quantile function possesses a closed form.

The tails of the proper GHyp distribution can be shown to behave as

$$f(x) \sim \mathrm{const}|x|^{\lambda-1}e^{(-\alpha|x|+\beta x)} \quad x \to \pm\infty \tag{1.10}$$

The proper GHyp distribution has at least four parameterizations in the literature ([86]) which are

1. $\mu, \delta, \alpha, \beta, \lambda$
2. $\mu, \delta, \zeta, \rho, \lambda$
3. $\mu, \delta, \xi, \chi, \lambda$
4. $\mu, \delta, \bar{\alpha}, \bar{\beta}, \lambda$

Table 1.2: Relationship between four parameterizations of the GHyp distribution at fixed values of μ and δ.

	To: Parameterization			
From: Parameterization	1	2	3	4
	$\mu, \delta, \alpha, \beta, \lambda$	$\mu, \delta, \zeta, \rho, \lambda$	$\mu, \delta, \xi, \chi, \lambda$	$\mu, \delta, \bar{\alpha}, \bar{\beta}, \lambda$
1. $\mu, \delta, \alpha, \beta, \lambda$ $(\alpha > 0, \delta > 0)$		$\zeta = \delta\sqrt{\alpha^2 - \beta^2}$ $\rho = \beta/\alpha$	$\xi = \frac{1}{\sqrt{1+\delta\sqrt{\alpha^2-\beta^2}}}$ $\chi = \frac{1}{\alpha\sqrt{1+\delta\sqrt{\alpha^2-\beta^2}}}$	$\bar{\alpha} = \delta\alpha$ $\bar{\beta} = \delta\beta$
2. $\mu, \delta, \zeta, \rho, \lambda$ $(\zeta > 0, \delta > 0)$	$\alpha = \frac{\zeta}{\delta\sqrt{1-\rho^2}}$ $\beta = \rho\alpha$		$\xi = 1/\sqrt{1+\zeta}$ $\chi = \zeta\rho$	$\bar{\alpha} = \zeta/\sqrt{1-\rho^2}$ $\bar{\beta} = \rho\bar{\alpha}$
3. $\mu, \delta, \xi, \chi, \lambda$ $(\xi > 0, \delta > 0)$	$\alpha = \frac{1-\xi^2}{\delta\xi\sqrt{\xi^2-\chi^2}}$ $\beta = \alpha\chi/\xi$	$\zeta = 1/\xi^2 - 1$ $\rho = \chi/\xi$		$\bar{\alpha} = \frac{1-\xi^2}{\xi\sqrt{\xi^2-\chi^2}}$ $\bar{\beta} = \bar{\alpha}\chi/\xi$
4. $\mu, \delta, \bar{\alpha}, \bar{\beta}, \lambda$ $(\bar{\alpha} > 0, \delta > 0)$	$\alpha = \bar{\alpha}/\delta$ $\beta = \bar{\beta}/\delta$	$\zeta = \sqrt{\bar{\alpha}^2 - \bar{\beta}^2}$ $\rho = \bar{\beta}/\bar{\alpha}$	$\xi = \frac{1}{\sqrt{1+\sqrt{\bar{\alpha}^2-\bar{\beta}^2}}}$ $\chi = \frac{\bar{\beta}}{\bar{\alpha}\sqrt{1+\sqrt{\bar{\alpha}^2-\bar{\beta}^2}}}$	

The hyperbolic distribution is a subclass of the GHyp distribution when $\lambda = 1$.

Definition 4. *The hyperbolic distribution has four parameters α, β, μ and δ. The probability density function is*

$$f(x) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\delta\alpha K_1(\delta\sqrt{\alpha^2 - \beta^2})} e^{-\alpha\sqrt{\delta^2+(x-\mu)^2}+\beta(x-\mu)} \tag{1.11}$$

where $\mu \in \mathbb{R}$, $|\beta| < \alpha$ and $\delta \geq 0$.

As for the proper GHyp distribution, there are multiple parameterizations for the hyperbolic distribution ([14] and [86]), four of which are

1. $\mu, \delta, \alpha, \beta$
2. μ, δ, ζ, π
3. $\mu, \delta, \psi, \gamma$
4. μ, δ, ξ, χ

Table 1.3: Relationships between four parameterizations of the Hyperbolic distribution at fixed values of μ and δ.

	To: Parameterization			
From: Parameterization	1	2	3	4
	$\mu, \delta, \alpha, \beta$	μ, δ, ζ, π	$\mu, \delta, \psi, \gamma$	μ, δ, ξ, χ
1. $\mu, \delta, \alpha, \beta$ $(\alpha > 0, \delta > 0)$		$\zeta = \delta\sqrt{\alpha^2 - \beta^2}$ $\pi = \beta/\sqrt{\alpha^2 - \beta^2}$	$\psi = \alpha + \beta$ $\gamma = \alpha - \beta$	$\xi = \frac{1}{\sqrt{1+\delta * \sqrt{\alpha^2-\beta^2}}}$ $\chi = \frac{\beta}{\alpha\sqrt{1+\delta\sqrt{\alpha^2-\beta^2}}}$
2. μ, δ, ζ, π $(\zeta > 0, \delta > 0)$	$\alpha = \zeta\sqrt{1+\pi^2}/\delta$ $\beta = \zeta\pi/\delta$		$\psi = \frac{\zeta}{\delta(\sqrt{1+\pi^2}+\pi)}$ $\gamma = \frac{\zeta}{\delta(\sqrt{1+\pi^2}-\pi)}$	$\xi = 1/\sqrt{1+\zeta}$ $\chi = \frac{\pi}{\sqrt{(1+\zeta)(1+\pi^2)}}$
3. $\mu, \delta, \psi, \gamma$ $(\psi > 0, \gamma > 0, \delta > 0)$	$\alpha = (\phi + \gamma)/2$ $\beta = \alpha\chi/\xi$	$\zeta = \delta\sqrt{\psi\gamma}$ $\rho = \chi/\xi$		$\xi = \frac{1}{\sqrt{1+\delta\sqrt{\phi\gamma}}}$ $\bar{\beta} = \bar{\alpha}\chi/\xi$
4. μ, δ, ξ, χ $(\xi > 0, \delta > 0)$	$\alpha = \frac{1-\xi^2}{\delta\xi\sqrt{xi^2 - chi^2}}$ $\beta = \frac{\chi(1-\xi^2)}{\delta\xi^2\sqrt{\xi^2-\chi^2}}$	$\zeta = (1 - xi^2)/xi^2$ $\pi = \frac{\chi}{\sqrt{\xi^2-\chi^2}}$	$\psi = (1-\xi^2)\frac{\xi+\chi}{\delta\xi^2\sqrt{\xi^2-\chi^2}}$ $\gamma = \frac{(1-\xi^2)(\xi-\chi)}{\delta\xi^2\sqrt{xi^2-chi^2}}$	

The mean and variance of the hyperbolic distribution are given by

$$\mathrm{E}(X) = \mu + \delta\pi R_1(\zeta) \tag{1.12}$$

$$\mathrm{Var}(X) = \delta^2\left(\zeta^{-1}R_1(\zeta) + \pi^2 S_1(\zeta)\right) \tag{1.13}$$

where R_λ and S_λ are defined as

$$R_\lambda(\zeta) = \frac{\mathrm{K}_{\lambda+1}(\zeta)}{\mathrm{K}_\lambda(\zeta)} \tag{1.14}$$

$$S_\lambda(\zeta) = \frac{\mathrm{K}_{\lambda+2}(\zeta)\mathrm{K}_\lambda(\zeta) - K^2_{\lambda+1}(\zeta)}{\mathrm{K}^2_\lambda(\zeta)}. \tag{1.15}$$

The hyperbolic distribution appears not as complex as the proper GHyp distribution. As a consequence, there were functions to implement linear modeling with hyperbolic errors ([107]). Those functions are reviewed in Section 1.1.2.

The second subclass distribution is the NIG distribution which is obtained when $\lambda = -1/2$.

Definition 5. *The NIG distribution has four parameters α, β, μ and δ. The probability density function is*

$$f(x) = \frac{\alpha\delta}{\pi} e^{\delta\sqrt{\alpha^2-\beta^2}+\beta(x-\mu)} \frac{\mathrm{K}_1(\alpha\sqrt{\delta^2+(x-\mu)^2})}{\sqrt{\delta^2+(x-\mu)^2}} \tag{1.16}$$

where $\mu \in \mathbb{R}$, $0 \le |\beta| \le \alpha$ and $\delta \ge 0$.

There are at least three parameterizations of the NIG distribution in the literature ([11] and [88]) which are

1. $\alpha, \beta, \mu, \delta$
2. $\bar{\alpha}, \bar{\beta}, \mu, \delta$
3. χ, ξ, μ, δ

Table 1.4: Relationships between three parameterizations of the NIG distribution at fixed values of μ and δ.

	To: Parameterization		
From: Parameterization	1	2	3
	$\mu, \delta, \alpha, \beta$	$\mu, \delta, \bar{\alpha}, \bar{\beta}$	μ, δ, χ, ξ
1. $\mu, \delta, \alpha, \beta$ $(0 \le \lvert\beta\rvert \le \alpha, \delta \ge 0)$		$\bar{\alpha} = \delta\alpha$ $\bar{\beta} = \delta\beta$	$\chi = \frac{\beta}{\alpha\sqrt{1+\delta\sqrt{(\alpha^2-\beta^2)}}}$ $\xi = \frac{1}{\sqrt{1+\delta\sqrt{(\alpha^2-\beta^2)}}}$
2. $\mu, \delta, \bar{\alpha}, \bar{\beta}$ $(0 \le \lvert\bar{\beta}\rvert \le \bar{\alpha}, \delta \ge 0)$	$\alpha = \bar{\alpha}/\delta$ $\beta = \bar{\beta}/\delta$		$\chi = \frac{\bar{\beta}}{\bar{\alpha}\sqrt{1+\sqrt{(\bar{\alpha}^2-\bar{\beta}^2)}}}$ $\xi = \frac{1}{\sqrt{1+\sqrt{(\bar{\alpha}^2-\bar{\beta}^2)}}}$
3. μ, δ, χ, ξ $(-1 < \chi < 1, 0 < \xi < 1, \delta > 0)$	$\alpha = \frac{1-\xi^2}{\xi^2\delta\sqrt{(1-\chi^2\xi^2)}}$ $\beta = \frac{\chi(1-\xi^2)}{\xi\delta\sqrt{(1-\chi^2\xi^2)}}$	$\bar{\alpha} = \frac{1-\xi^2}{\xi^2\sqrt{(1-\chi^2\xi^2)}}$ $\bar{\beta} = \frac{\chi(1-\xi^2)}{\xi\sqrt{(1-\chi^2\xi^2)}}$	

The moment generating function of the NIG distribution is in a much simpler form which can be shown to be

$$\mathrm{M}(t) = e^{\delta(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+t)^2})+\mu t} \tag{1.17}$$

Consequently, the mean and variance have simple explicit expressions which are

$$\mathrm{E}(X) = \mu + \frac{\delta\bar{\beta}/\bar{\alpha}}{\left(1-\left(\bar{\beta}/\bar{\alpha}\right)^2\right)^{1/2}} \tag{1.18}$$

and

$$\mathrm{Var}(X) = \frac{\delta^2}{\bar{\alpha}\left(1-\left(\bar{\beta}/\bar{\alpha}\right)^2\right)^{3/2}} \tag{1.19}$$

The tails of the NIG distribution can be shown to behave as

$$f(x) \sim \mathrm{const}|x|^{-3/2}e^{(-\alpha|x|+\beta x)} \quad x \to \pm\infty \tag{1.20}$$

The NIG distribution appears more tractable than the hyperbolic distribution without losing the attractive property of semi-heavy tails, therefore linear modeling using the NIG distribution can potentially be possible as it is for the hyperbolic distribution.

The skew hyperbolic Student's t distribution is the limiting distribution of the GHyp distributions when $\lambda = -\nu/2$ and $\alpha = |\beta|$.

Definition 6. *The skew hyperbolic Student's* t *distribution has four parameters β, ν, δ and μ. The probability density function is*

$$f(x) = \frac{2^{(1-\nu)/2}\delta^{\nu}|\beta|^{(\nu+1)/2}e^{\beta(x-\mu)}K_{\frac{\nu+1}{2}}(\sqrt{\beta^2(\delta^2+(x-\mu)^2)})}{\gamma(\nu/2)\sqrt{\pi}(\sqrt{\delta^2+(x-\mu)^2})^{(\nu+1)/2}}, \quad \beta \neq 0 \tag{1.21}$$

and

$$f(x) = \frac{\gamma(\nu+1)/2}{\sqrt{\pi}\delta\gamma\nu/2}\left(1+\frac{(x-\mu)^2}{\delta^2}\right)^{-(\nu+1)/2}, \quad \beta = 0 \tag{1.22}$$

where γ is the gamma function. $\delta, \mu, \beta \in R$ and $\nu > 0$. When $\beta = 0$ and $\delta = 1$, the distribution reduces to the ordinary Student's t *distribution.*

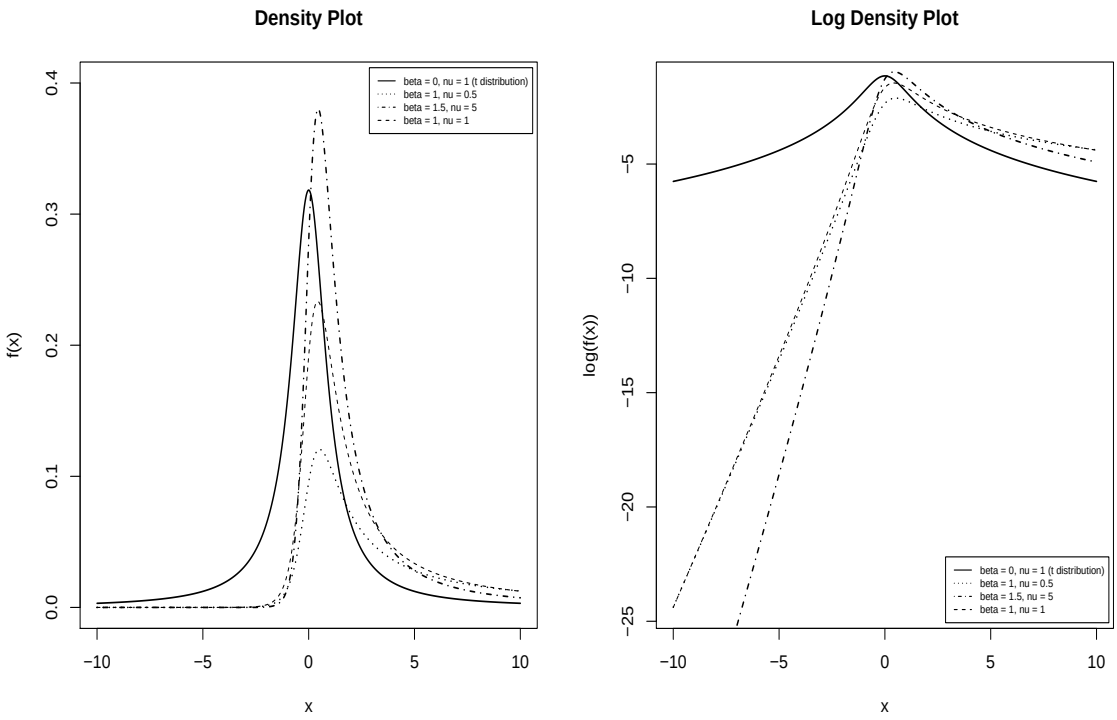

Figure 1.4: A selection of Skew Hyperbolic Student's t distributions with various β, ν when $\mu = 0$ and $\delta = 1$

The distribution does not appear to have alternative parameterizations in the literature.

The mean and variance are given by

$$\mathrm{E}(x) = \mu + \frac{\beta\delta^2}{\nu - 2} \tag{1.23}$$

and

$$\mathrm{Var}(X) = \frac{2\beta^2\delta^4}{(\nu-2)^2(\nu-4)} + \frac{\delta^2}{\nu-2} \tag{1.24}$$

Unlike the two subclasses described above, β is non-zero, the tails of skew hyperbolic Student's t distribution behave as one exponential and one polynomial. ([1])

Definition 7. *In the polynomial tail, the density function is given by*

$$f(x) \sim \mathrm{const}|x|^{-\nu/2-1} \text{ when } \begin{cases} \beta < 0, x \to -\infty \\ \beta > 0, x \to \infty \end{cases}$$

In the exponential tail, the density function is given by

$$f(x) \sim \mathrm{const}|x|^{-\nu/2-1}\exp(-2|\beta x|) \text{ when } \begin{cases} \beta < 0, x \to -\infty \\ \beta > 0, x \to \infty \end{cases}$$

As $\nu \to 0$, the polynomial tail tends to decay extremely slowly, therefore it becomes extreme heavy and is difficult to approximate numerically.

Despite the valuable properties of the GHyp distributions, the package developments in *The Comprehensive R Archive Network* remain incomplete due to the complexity of the GHyp distributions.

1.1.2 The GeneralizedHyperbolic Package

The **GeneralizedHyperbolic** package ([91]) (version 0.2-0) in CRAN, published March 1, 2010 by Associate Professor David Scott, provides density, distribution and quantile functions for the hyperbolic distribution, the GHyp distribution, the generalized inverse Gaussian distribution and the skew-Laplace distribution. In addition it provides functions for fitting the hyperbolic distribution and the normal inverse Gaussian distribution to data. Apart from these core functions, it also contains functions `gigChangePars`, `ghypChangePars` and `hyperbChangePars` to interchange respectively between the parameterizations of GIG, GHyp and hyperbolic distribution illustrated in Table 1.1, 1.2 and 1.3.

Before reviewing the functions for the GHyp distribution, it is worthwhile to point out possible failures of the algorithm of the CDF of the GIG distribution. The algorithm which is implemented in the `pgig` function can potentially fail to converge or to return incorrect results when the it computes cumulative probabilities in the tail area. Since the core part of CDF approximation of the GIG distribution is actually the incomplete Bessel function calculation (see Chapter 2 and 3

for more discussions), approaches to remedy these shortcoming are discussed in Chapter 2, while the improved results for the `pgig` function are discussed in Chapter 3.

The functions `pghyp` and `qghyp` are designed to obtain the CDF and quantile functions of the GHyp distribution. The function `pghyp` of package version 0.2-0 breaks the real line into 8 regions by function `ghypBreaks` in order to determine the integral of the density function. For the upper and lower extreme regions (density $< 10^{-5}$), the probability is set to 1 or 0 accordingly. The function `qghyp`, on the other hand, is more complex. The chief drawbacks of the published `qghyp` function are:

- Inefficiency. It tends to be slow when dealing with large number of quantiles. It again uses the breakup of the real line into the same 8 regions as `pghyp` function.

- Lack of flexibility. Only one method is available. It first sets the two extreme tails (density $< 10^{-5}$) as $-\infty$ or ∞. In the 6 inner regions the function `splinefun` is used to fit values of the distribution function generated by `pghyp`. The quantiles are then found using the `uniroot` function.

- Not particularly accurate, generally expected to be accurate to five decimal places.

The search for accurate and efficient integration routines for obtaining the CDF and quantile function is discussed in Chapter 4.

Besides the probability and quantile function, a good random number generator function is equally important for a distribution. Since the hyperbolic distribution is a GHyp distribution with $\lambda = 1$ and the GHyp distribution is a mixture of the GIG distribution and normal distribution, possible approaches for the function `rhyperb` and `rgig` are reviewed respectively in Chapter 5. The current `rghyp` function uses the mixing method which generates random observations using the mixing property of the generalized inverse Gaussian distribution. It first samples σ^2 from the generalized inverse Gaussian distribution $N^-(\lambda, \sigma^2, \alpha^2 - \beta^2)$. The sampling approach for this step is Dagpunar's approach ([33]) for the GIG distribution. Then it returns the random variable with conditional normal distribution $Y \sim N(\mu + \beta\sigma^2, \sigma^2)$. Y is GHyp distributed and in the case $\lambda = 1$, a random observation from the hyperbolic distribution is obtained. However there are concerns with the current GIG distribution random number generation approach. Some remedies for these concerns which have been proposed but not implemented, including [34], [79] and [78]. These approaches are implemented and compared with the currently implemented approaches in 5. Besides these, we also implement one approach for the hyperbolic distribution without using the mixing property and investigate the best approaches for random number generation of the GIG distribution as well as the hyperbolic distribution.

Another aspect of the **GeneralizedHyperbolic** package is the fitting of the subclass distributions to data. These applications, reviewed separately, are derived for the hyperbolic distribution and the NIG distribution.

Fitting the hyperbolic distribution is done through the `hyperbFit` function. The `hyperbFit` function implements this with the `hyperbFitStart` function providing the starting values for the optimization routine ([107]). There are three approaches in the `hyperbFit` function to optimize the transformed parameters π, $\log(\zeta)$, $\log(\delta)$ and μ to ensure $\zeta, \delta > 0$

- "BFGS" A quasi-Newton method which is implemented in the `optim` function.
- "Nelder-Mead" An implementation of the Nelder-Mead simplex method which is in the `optim` function as well.
- "nlm" The `nlm` function which is an implementation of the method proposed by Schnabel, Koontz and Weiss ([90]).

However, without good starting values, the optimization routine is likely to fail using any of these three approaches. Therefore, the `hyperbFitStart` provides four approaches to find a set of starting values.

- "BN" A method from [9] which based on ψ and γ (see section 3.2) can represent the absolute slopes of the left and right asymptotes to the log density function.
- "FN" Based on a fitted normal distribution as it is a limit of the hyperbolic distribution.
- "SL" Based on a fitted skew-Laplace distribution of which the log density are two straight line with absolute slopes $1/\alpha$, $1/\beta$.
- "MoM" A method based on [18].

The work in Richard Trendall's Master Thesis ([107]) shows the fitting functions still need to be improved to provide a reliable platform for estimation in regression with hyperbolic errors. Consideration of fitting functions are the basis of the material discussed in Chapter 6.

Likewise fitting the NIG distribution is done through the `nigFit` function. The `nigFit` function implements this with the `nigFitStart` function providing the starting values for the optimization routine ([91]). Apart from the three optimization approaches implemented for fitting the hyperbolic distribution, there are three additional approaches in the `nigFit` function to optimize the transformed parameters π, $\log(\zeta)$, $\log(\delta)$ and μ to ensure $\zeta, \delta > 0$

- "L-BFGS-B" A limited-memory algorithm, described in [30], based on the gradient projection method and uses a limited memory BFGS matrices to approximate the Hessian of the objective function. The algorithm is implemented in the `optim` function.
- "nlminb" An implementation of the PORT optimization routine, described in [47], which is in the `nlminb` function.
- "constrOptim" The `constrOptim` function minimizes the objective function subject to linear inequality constraints using the barrier method with logarithmic barrier function ([75]).

Although the NIG distribution appears more tractable than the hyperbolic distribution, the optimization routines are not stable with arbitrarily chosen start values using any of the six approaches. Therefore, the `nigFitStart` provides three approaches to find a set of starting values.

- "FN" Based on a fitted normal distribution as it is a limit of the NIG distribution.
- "Cauchy" Based on a fitted Cauchy distribution as it is also a limit of the NIG distribution.
- "MoM" A method based on [18].

Due to the similarity of the NIG and the hyperbolic distributions, the work discussed in Chapter 6 can also be extended to improve linear modeling using the NIG distribution.

1.1.3 The Skew Hyperbolic Student's *t* Distribution Package

Functions for the skew hyperbolic Student's t distribution are implemented specifically in the **SkewHyperbolic** package ([94]) (version 0.3-2) in *The Comprehensive R Archive Network* (CRAN), published February 26, 2013 by Associate Professor David Scott and Fiona Grimson. The package provides density, CDF, quantile functions and random number generation for the skew hyperbolic Student's t distribution as well as implements the functions that fit the skew hyperbolic distribution to data.

The functions `pskewhyp` and `qskewhyp` are designed to obtain the distribution and quantile functions of the skew hyperbolic Student's t distribution. The published version of function `pskewhyp` uses the numerical integration function `integrate` to integrate the density function from $-\infty$ to x if x is to the left of the mode, and from x to ∞ if x is to the right of the mode ([94]). However this approach fails badly when the distribution is skewed. This also has an impact on quantile function calculations. The quantile function `qskewhyp` uses numerical root finding function `uniroot` to find the x where $F(x) = q$. $F(x)$ is approximated either as CDF approach or by spline approximation. The two options aim for accurate and efficient calculation respectively. But we have discovered that neither of these is stable when the distribution is skewed.

Apart from the **SkewHyperbolic** package, there is an alternative approach, the so-called the black-box approach, proposed in [38], implemented in the **Runuran** package. The algorithm, primarily designed for random number generation, is based on polynomial interpolation with five nodes of the inverse CDF utilizing Newton's formula together with Gauss-Lobatto integration. We are convinced it is accurate for a fairly wide parameter range, however it is known that the polynomial interpolation becomes numerically unstable in the tails of the distributions ([38]). In addition, the approach is rather complicated and difficult to adjust if necessary.

In my research, we propose a new CDF approximation approach which transforms the density function by the split t transformation, then integrates numerically by the Gaussian quadrature

with Richardson extrapolation or the `integrate` function. This approach, discussed in Chapter 4, is straightforward, relatively stable and addresses some of the previous concerns.

The other aspect of the package is fitting the skew hyperbolic Student's t distribution to data. This is achieved by the `skewFit` and `skewFitStart` functions. The `skewFitStart` function provides the starting values for the optimization routine ([94]).

The three optimization approaches in the `skewhypFit` function are adapted from the `hyperbFit` function to optimize the transformed parameters π, $\log(\nu)$, $\log(\delta)$ and μ to ensure $\nu, \delta > 0$. Like the hyperbolic distribution, the optimization routine will fail to obtain good results using any of these three approaches without carefully chosen start values. Therefore, the `skewhypFitStart` function uses a linear approximation approach to estimate the start values. The function first linearly approximates the log-density of the fitted skew hyperbolic student's t distribution to estimate the start value of ν and β. Then it solves the moment equations for mean and variance to obtain the start value of δ and μ.

Since the variance $\to \infty$ when $\nu \leq 4$, the estimate of ν must > 4. It is noted the approach will not work if the distribution is too skewed, this is due to the number of points in the lighter tail being insufficient to fit the required linear model. The improvement of fitting the skew hyperbolic Student's t distribution, in particular the optimization routine, is indeed required to obtain more stable outcomes from linear regression with the skew hyperbolic Student's t distribution. Therefore the updated CDF approximation approach together with the linear regression with the hyperbolic distribution approach provide a starting point for such improvements in the future.

CHAPTER 2

UTILITY FUNCTIONS

The utility functions, contained in the package **DistributionUtils** can be categorized according to purpose, as testing functions for distributions, general routine functions, Bessel function related functions and plotting functions.

The first group of functions are functions which allow the testing of distribution function routines. Most of these functions have been developed in the past by David J. Scott and Christine Yang Dong to provide support for other packages. In this group are the functions for inversion tests, the Massart inequality, standard errors of moments, and integration of a density. Since **DistributionUtils** lacks a goodness-of-fit test ([95]), I have in addition implemented the goodness-of-fit test using Moran's log spacing statistic.

The second group of functions comprises general routines for performing common tasks to do with distributions: calculating the distribution function and the quantile function given the density function; and calculating moments by integration.

Further functions concern Bessel functions and their inclusion in the package is due to their importance in the distributions related to the generalized hyperbolic distribution. One of these functions is independently important being the first available implementation of the incomplete Bessel K function in R. Indeed, the incomplete Bessel K function is not implemented in the usual collections of special functions such as Netlib. This implementation is due to Slevinsky and Safouhi, see [101] and [102]. Their basic algorithm has been improved in order to allow it handle some extreme cases as well as vectors.

2.1 Functions for Testing Distributions

The functions that have been developed in the past for testing distributions are the functions for inversion tests, the Massart inequality, standard errors of moments, and the integration of a

density.

The inversion tests compare the difference of a given value and the value after transformation then back transformation. There are two functions in the package to carry out such tests. The function `inversionTestpq` basically aims to check that `qDist(pDist(x)) = x` for randomly generated values `x` from a specific distribution. It first generates `n` random variates from the test distribution then calculates the probabilities of those random values under the test cumulative distribution. Lastly, it computes the quantiles for those probabilities and compares them with original random values. Similarly, the function `inversionTestqp` aims to check that `pDist(qDist(qs)) = qs` for a set of probabilities `qs`. It applies the quantile function, followed by the distribution function to the set of probabilities specified in `qs`.

Both of the functions return a list but with slightly different components. The outcome of function `inversionTestpq` contains `qDist(pDist(x))`, `x`, the differences `qDist(pDist(x)) - x` and number of random numbers generated from the distribution. The outcome of function `inversionTestqp` includes `pDist(qDist(p))`, `p`, and the differences `pDist(qDist(qs)) - qs`.

The Massart Inequality test, implemented in the function `distIneqMassart`, is based on the Massart version of the Dvoretzky-Kiefer-Wolfowitz inequality :

$$\Pr\left(\sup_x |\hat{F}_n(x) - F(x)| > t|\right) \leq 2\exp(-2nt^2) \tag{2.1}$$

where $F(x)$ is the distribution function of the test distribution and $\hat{F}_n(x)$ is the empirical distribution function for a sample of n independent and identically distributed random variates with distribution function $F(x)$. This inequality is true for all distribution functions and all n and t.

The idea of the test using the Massart inequality is that if the value of t is such that the bound on the right side of the inequality is small, then if a sample is from the desired distribution, the supremum should be less than t with high probability. The `distIneqMassart` function generates n random variates from the test distribution, then computes the supremum `sup` of the absolute difference between the empirical distribution $\hat{F}_n(x)$ from those random variates and the true distribution function $F(x)$. Then it computes t which is the t as in (2.1) calculated from the supplied probability bound value (default 0.001). Finally, the function compares `sup` and t. If `sup` $< t$, then the test has not detected any evidence that the sample is not from the desired distribution.

2.1.1 Moran Test

The Moran test, proposed in [83], tests the goodness-of-fit of a random sample of x_i, $i = 1, \ldots n$, to a continuous univariate distribution with cumulative distribution function $F(x, \theta)$, where θ is a vector of known parameters.

The key advantage of the Moran test compared to other tests such as Kolmogorov-Smirnov test and Cramér-von Mises Test is that the asymptotic distribution of the Moran test statistic does not change substantially if instead of θ being known, it is estimated efficiently as $\hat{\theta}$, for

example using maximum likelihood estimation. This property was investigated and confirmed for a number of distributions in [52].

Assume $x_1 < x_2 < \ldots < x_n$, the spacing $D_i(\theta)$ is defined as

$$D_i(\theta) = y_i - y_{i-1} \quad (i = 1, \ldots, m), \tag{2.2}$$

where $y_i = F(x_i, \theta)$, $m = n + 1$, $y_0 = 0$ and $y_m = 1$. The Moran statistic is then

$$M(\theta) = -\sum_{i=1}^{n} \log D_i(\theta). \tag{2.3}$$

The test is clearly limited as the parameters are usually unknown in practice. It was extended to test for estimated parameters in [32] which is implemented in the package. The test statistic is:

$$T(\hat{\theta}) = (M(\hat{\theta}) + 1/2k - C_1)/C_2, \tag{2.4}$$

where $M(\hat{\theta})$, the Moran statistic, has mean γ_m and variance σ_m^2 expanded to order m^{-1} as

$$\begin{aligned} \gamma_m &= m(\log m + \gamma) - \frac{1}{2} - \frac{1}{12m} + \ldots, \\ \sigma_m^2 &= m\left(\frac{\pi^2}{6} - 1\right) - \frac{1}{2} - \frac{1}{6m} + \ldots, \end{aligned} \tag{2.5}$$

where $\gamma \approx 0.57722$ is Euler's constant. C_1 and C_2 are then defined as

$$\begin{aligned} C_1 &= \gamma_m - \sigma_m \sqrt{n/2} \\ C_2 &= \frac{\sigma_m}{\sqrt{2n}} \end{aligned} \tag{2.6}$$

(2.7)

This test has null hypothesis H_0: a random sample of n values of x comes from distribution $F(x, \theta)$. Here θ is expected to be the maximum likelihood estimate $\hat{\theta}$, an efficient estimate. The test rejects H_0 at significance level α if $T(\hat{\theta}) > \chi_n^2(\alpha)$.

The Moran test function takes the format as below:

```
moranTest(x, densFn, param = NULL, ...)
```

`x`

The data vector that is being tested.

`densFn`

The root name of the distribution to be tested, for instance 'norm' for the Normal distribution.

`param`

A vector giving the parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of the distribution are used instead

...

Additional arguments to allow specification of the parameters of the distribution other than specified by `param` argument.

2.2 General Routine Functions

As mentioned in Chapter 1, the general routine functions in the **distr** package approximate the distribution function and quantile function by sampling. On the other hand the algorithms investigated and implemented in the functions described in the following, are able to perform general routine tasks for any univariate unimodal continuous distribution by means of numerical calculation. Before we describe these two functions, we first introduce two general auxiliary functions developed to support the general CDF and quantile function approximation algorithms.

2.2.1 General Auxiliary Functions

The first general auxiliary function is the `distStepSize` function. This function implements an algorithm to determine the appropriate step size when determining the range of the specified unimodal distribution. The algorithm used is:

1. Simulate a small random sample of the specified unimodal distribution and appximate the median x_{median} of the distribution by sampling.

2. If the distribution specified is the skew hyperbolic Student's t distribution, i.e. skewhyp(β, δ, μ, ν), then the step size calculation is specially arranged. Recall the description of skew hyperbolic Student's t distribution in Section , when β is non-zero, the tails of skew hyperbolic Student's t distribution behave as one exponential and one polynomial. The step size of skew hyperbolic Student's t distribution is calculated as

   a) The step size of polynomial declining tail is $\delta|\beta|(\nu D)^{\frac{-2}{\nu}}$, where D is the current distance value. The step size of the exponentially declining tail is δ.

   b) If $\beta > 0$, the right tail is polynomial tail and the left tail is exponential tail, vice versa.

   c) If $\beta = 0$, the distribution is symmetric, and then the step size is $e^{\delta/\nu}$.

3. If the distribution specified is any other unimodal distribution then the step size for the left tail is $x_{\text{median}} - Q_{0.25}$ where $Q_{0.25}$ is the empirical 0.25-quantile from the random sample generated in step 1. The step size for the right tail is $Q_{0.75} - x_{\text{median}}$ where $Q_{0.75}$ is the empirical 0.75-quantile from the random sample generated in step 1.

The `distStepSize` function has the format

```
distStepSize(densFn, dist,
             param = NULL, side = c("right","left"), ...)
```

`densFn`

The root of the name of the density function for which the step size is required.

`dist`

Current distance value, for skew hyperbolic distribution only.

`param`

The parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of each distribution are used instead.

`side`

Define the side of distribution. "right" for a step to the right, "left" for a step to the left.

...

Additional arguments to allow specification of the parameters of the distribution other than specified by `param` argument.

Apart from the `distStepSize` function, a general algorithm to calculate the mode of a unimodal continuous distribution is implemented in the `distMode` function. The algorithm is

1. Approximate the median x_{median} of the specified distribution by sampling.
2. The lower bound of the interval x_{low} is found by iteratively performing $x_{\text{median}} - i * x_{\text{stepsizeL}}$ until the value of x_{low} ensures $\log f(x_{\text{low}}) < \log f(x_{\text{median}})$. $x_{\text{stepsizeL}}$ is determined by the application of `distStepSize` function on left tail.
3. The upper bound of the interval x_{high} is found by iteratively performing $x_{\text{median}} + i * x_{\text{stepsizeR}}$ until the value of x_{high} ensures $\log f(x_{\text{high}}) > \log f(x_{\text{median}})$. $x_{\text{stepsizeR}}$ is determined by the application of the `distStepSize` function on right tail.
4. The maximum density of that interval is claimed to be the distribution mode.

The `distMode` function has the format as

```
distMode(densFn, param = NULL, ...)
```

`densFn`

The root of the name of the density function for which the mode is required.

`param`

The parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of each distribution are used instead.

`...`

Additional arguments to pass to the `optimize` function.

2.2.2 Distribution Function Calculation

The general routine algorithm calculates the distribution function for all unimodal distributions that have a continuous density function defined as:

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t$$

For random variables with the density function $f(x)$, the distribution function is calculated as

$$F(x) = \begin{cases} \int_{-\infty}^{x} f(t)\,\mathrm{d}t & \text{if } t \le t_{\text{mode}} \\ 1 - \int_{x}^{\infty} f(t)\,\mathrm{d}t & \text{if } t > t_{\text{mode}} \end{cases}$$

The integration is achieved numerically by the `integrate` function from the **stats** package. The algorithm is implemented in the function `pDist` which has the format

```
pDist(densFn = "norm", q, param = NULL, subdivisions = 100,
      lower.tail = TRUE, intTol = .Machine$double.eps^0.25,
      valueOnly = TRUE, ...)
```

`densFn`

The root of the name of the density function for which the CDF is required.

`q`

The quantiles of the probabilities that are required to be computed.

`param`

The parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of each distribution are used instead.

`subdivisions`

The maximum number of subdivisions used to integrate the density and determine the accuracy of the distribution function calculation.

`lower.tail`

If `lower.tail = TRUE`, the cumulative distribution is taken from the lower tail.

`intTol`

The required relative accuracy of the numerical integration function `integrate`.

`valueOnly`

If `valueOnly = TRUE` calls to `pDist` function only return the value obtained for the integral. If `valueOnly = FALSE` an estimate of the accuracy of the numerical integration is also returned.

...

Additional arguments to allow specification of the parameters of the distribution other than specified by `param` argument.

This function is fairly flexible as the requested absolute accuracy for the function `integrate` can be adjusted by `intTol` in the argument list, therefore the accuracy of probabilities can be adjusted accordingly. The default `intTol` is `(.Machine$double.eps)^(0.25)` = 0.0001220703 on a 32-bit system, so the accuracy of probability calculation is expected to be around 10^{-4}. The output of this function contains a list of `p` and `error` where `p` is the probability required and `error` is the estimate of the accuracy of the approximation to the distribution function.

2.2.3 Quantile Function Calculation

The quantile function is the inverse of the distribution function, i.e. $F(x)^{-1}$ therefore the calculation of quantile function requires the use of the `pDist` function discussed in Section 2.2.2. The general algorithm for a unimodal continuous distribution consists of two approaches depending on priorities. If efficiency is the priority then the distribution function is approximated by spline interpolation, the Spline approach. Alternatively numerical integration of the density function, the Integrate approach, is used for precise calculation.

Given $p = F(x)^{-1}$ and $f(x)$ is the associated density function, both approaches use the one dimensional root finding function `uniroot` to find the root for $F(x) - p = 0$ over a finite interval. The main differences between these two approaches are the method of approximation of $F(x)$ and the method of determination of the appropriate interval.

The Integrate approach uses the `pDist` function to approximate $F(x)$ and the appropriate interval is determined as

1. Compute $p_{\text{mode}} = F(x_{\text{mode}})^{-1}$ where x_{mode} is obtained from the `distMode` function described in Section 2.2.1.

2. If $p \leq p_{\text{mode}}$ then $f(x)$ is numerically integrated iteratively from $x_{\text{mode}} - x_{\text{stepsizeL}}$ until $x \leq (x_{\text{mode}} - i * x_{\text{stepsizeL}})$.

3. If $p > p_{\text{mode}}$ then $f(x)$ is numerically integrated iteratively from $x_{\text{mode}} + x_{\text{stepsizeR}}$ until $x \geq (x_{\text{mode}} + i * x_{\text{stepsizeR}})$.

This approximation method involves a significant number of integrations, thus the time required is roughly linear in the number of quantiles being calculated, which can be a time-consuming task. The advantage of this approach is accuracy. The tolerance of the function

`uniroot` as well as the accuracy of function `pDist` can be adjusted by `uniTol` in the argument list. The default setting for `uniTol` is also `(.Machine$double.eps)^(0.25)` thus the accuracy is expected to be around 10^{-4}. It nevertheless can be set to 10^{-10} or 10^{-12} if the accuracy is needed to be around 10^{-8}.

The Spline method on the other hand uses spline interpolation via the `spline` function to approximate the major part of $F(x)$. The extreme tail approximation where the probabilities are $\leq 10^{-5}$ still uses the Integrate method.

The Spline approach is set out as

1. Compute the upper bound x_{high} and lower bound x_{low} of which the probabilities are $\leq 10^{-5}$.
2. Divide the interval $[x_{\text{low}}, x_{\text{high}}]$ into n sub-intervals.
3. Construct the function giving the spline interpolation within the interval. This function therefore approximates $F(x)$.

This approach is expected to be less efficient when a single quantile calculation is required. This is because the spline function approximates the interval with same number of sub-intervals regardless of the number of quantiles required. However with the requirement of large number of quantiles, the Spline approach is more efficient as the spline approximation is only done once provided that the probability is greater than 10^{-5}.

This general quantile approximation algorithm is implemented in function `qDist` which has the format

```
qDist(densFn = "norm", p, param = NULL,
      lower.tail = TRUE, method = "spline", nInterpol = 501,
      uniTol = .Machine$double.eps^0.25,
      subdivisions = 100, intTol = uniTol, ...)
```

`densFn`

The root of the name of the density function for which the quantile function is required.

`p`

The probabilities of the quantiles that are required to be computed.

`param`

The parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of each distribution are used instead.

`lower.tail`

If `lower.tail = TRUE`, the cumulative distribution is taken from the lower tail.

`method`

The approximation method.

`nInterpol`

Number of points used for cubic spline interpolation of the distribution function.

`uniTol`

The desired accuracy (convergence tolerance) of the one dimentional root search function `uniroot`.

`subdivisions`

The maximum number of subdivisions used to integrate the density and determine the accuracy of the distribution function calculation.

`intTol`

The required relative accuracy of the numerical integration function `integrate`.

...

Additional arguments to allow specification of the parameters of the distribution other than specified by `param` argument.

2.2.4 Moment Calculation

The function `momIntegrated` in this package implements the calculation of moments and absolute moments about a given location for any distribution that has a continuous density function. The algorithm of this function is based on

$$\mu^k = \int_{-\infty}^{\infty} (x-a)^k f(x)\,dx, \tag{2.8}$$

where μ^k denotes the kth moment about location a. When the absolute moment is required, $|x-a|^k$ replaces $(x-a)^k$ in (2.8).

The function `momIntegrated` calls the corresponding density function by the root for the distribution's name in the argument list and forms $(x-a)^k f(x)$. Then the function `integrate` is applied to perform the integration in (2.8).

The function `momIntegrated` has the format

```
momIntegrated(densFn, param = NULL, order, about = 0,
              absolute = FALSE, ...)
```

`densFn`

The root of the name of the density function for which the distribution function is required.

`param`

The parameter values for the distribution specified by `densFn`. If no `param` values are specified, then the default parameter values of each distribution are used instead.

`order`

The order of the moment or absolute moment to be calculated.

`about`

The point about which the moment is to be calculated.

`absolute`

Whether absolute moments or ordinary moments are to be calculated.

...

Additional arguments to allow specification of the parameters of the distribution other than specified by `param` argument.

2.3 Bessel Functions

There are two functions concerning Bessel functions. One is for incomplete Bessel K function calculation and the other is for Bessel function ratio calculation. The implementation of the Bessel function ratio uses exponential scaling of the Bessel function to avoid over- or underflow, but is otherwise unremarkable.

We will discuss the incomplete Bessel K function implementation in detail only because changes have been made to improve its performance.

The incomplete Bessel K function can be defined as

$$K_\nu(x,y) = \int_1^\infty \frac{e^{-xt-y/t}}{t^{\nu+1}} \mathrm{d}t, \tag{2.9}$$

and has been a computational challenge due to the integration complexity. Richard M. Slevinsky and Hassan Safouhi proposed the Slevinsky-Safouhi formulas I and II for analytical calculation of higher order derivatives in 2009 ([101]). They then extended and applied the Slevinsky-Safouhi formula I to a recursive algorithm to approximate the incomplete Bessel function analytically. They have also provided the original code of the function `incompleteBesselK` which it is claimed rapidly computes the incomplete Bessel function to pre-determined accuracies of between $\pm 1e^{-10}$ and $\pm 1e^{-15}$ over a range of `x` and `y` ([102]). This algorithm has been implemented by David J. Scott and Thomas Tran in `R` code which calls a Fortran routine to carry out the calculation. However it was found that the algorithm could fail to converge or gave incorrect results when either `x` or `y` take extremely small values. Furthermore, the `incompleteBesselK` function can only handle `x` and `y` as scalars. These shortcomings motivated the modification of the algorithm.

2.3.1 Modified Algorithm

The recursive algorithm, currently implemented in the `incompleBesselK` function ([102]), is set out as:

1. For $F(x) = \int_0^x f(t)\mathrm{d}t$, set:

$$N_0(x) = \frac{F(x)}{x^{\sigma_0} f(x)} \text{ and } D_0(x) = \frac{1}{x^{\sigma_0} f(x)} \tag{2.10}$$

2. For $n = 1, 2, \ldots$, $N_n(x)$ and $D_n(x)$:

$$D_n(x) = \left(x^2 \frac{d}{dx}\right)^n \frac{1}{x^{\sigma_0} f(x)} \text{ and } N_n(x) = \left(x^2 \frac{d}{dx}\right)^n \frac{F(x)}{x^{\sigma_0} f(x)} \tag{2.11}$$

The approximations for $D_n(x)$ and $N_n(x)$ are

$$\begin{aligned} \tilde{D}_n(x,y,\nu) &= \left(t^2 \frac{\mathrm{d}}{\mathrm{d}t}\right)^n (t^{\nu+1} e^{t+xy/t})\Big|_{t=x} \\ &= (-xy)^n x^{\nu+1} e^{x+y} \sum_{i=0}^{r} \binom{n}{r} (-y)^{-r} \sum_{i=0}^{r} A_r^i x^i \end{aligned} \tag{2.12}$$

$$\begin{aligned} \tilde{N}_n(x,y,\nu) &= N_n(x) - F(x) D_n(x) \\ &= \sum_{r=1}^{n} \binom{n}{r} \tilde{D}_{n-r}(x) \left(x^2 \frac{\mathrm{d}}{\mathrm{d}x}\right)^{r-1} \left(x^{\sigma_0} f(x)\right) \\ &= \frac{e^{-x-y}}{x^\nu y} \sum_{r=1}^{n} \binom{n}{r} \tilde{D}_{n-r}(x,y,\nu)(xy)^r \sum_{s=0}^{r-1} \binom{r-1}{s} y^{-s} \sum_{i=0}^{s} A_s^i (-x)^i \end{aligned} \tag{2.13}$$

3. The approximation to $K_\nu(x,y) = \int_x^\infty f(t)$ is

$$\tilde{G}_n^{(1)}(x,y,\nu) = x^\nu \frac{\tilde{N}_n(x,y,\nu)}{\tilde{D}_n(x,y,\nu)} \tag{2.14}$$

In this recursive algorithm, we have modified the calculation of numerator and denominator to reduce the impact of x and y when they take extreme values. In the modified algorithm we introduce a new variable $\tilde{D}_n^*$, where

$$\begin{aligned} \tilde{D}_n^* &= (-xy)^{-n} e^{-(x+y)} \tilde{D}_n(x,y,\nu) \\ &= x^{\nu+1} \sum_{i=0}^{r} \binom{n}{r} (-y)^{-r} \sum_{i=0}^{r} A_r^i x^i. \end{aligned} \tag{2.15}$$

Then

$$
\begin{aligned}
\tilde{N}_n(x,y,\nu) &= \frac{e^{-x-y}}{x^\nu y}\sum_{r=1}^{n}\binom{n}{r}\tilde{D}_{n-r}(x,y,\nu)(xy)^r\sum_{s=0}^{r-1}\binom{r-1}{s}y^{-s}\sum_{i=0}^{s}A_s^i(-x)^i \\
&= \frac{e^{-x-y}}{x^\nu y}\sum_{r=1}^{n}\binom{n}{r}(-xy)^{n-r}x^{\nu+1}e^{x+y}\sum_{r=0}^{n-r}\binom{n-r}{r}(-y)^{-r}\sum_{i=0}^{r}A_r^i x^i(xy)^r \\
&\quad \sum_{s=0}^{r-1}\binom{r-1}{s}y^{-s}\sum_{i=0}^{s}A_s^i(-x)^i \\
&= \frac{e^{-x-y}}{x^\nu y}\sum_{r=1}^{n}\binom{n}{r}(-xy)^{n}x^{\nu+1}e^{x+y}\sum_{r=0}^{n-r}\binom{n-r}{r}(-y)^{-r}\sum_{i=0}^{r}A_r^i x^i \\
&\quad (-1)^{-r}\sum_{s=0}^{r-1}\binom{r-1}{s}y^{-s}\sum_{i=0}^{s}A_s^i(-x)^i \\
&= \frac{(-xy)^n}{x^\nu y}\sum_{r=1}^{n}\binom{n}{r}(-1)^{-r}\tilde{D}^*_{n-r}(x,y,\nu)\sum_{s=0}^{r-1}\binom{r-1}{s}y^{-s}\sum_{i=0}^{s}A_s^i(-x)^i
\end{aligned}
\tag{2.16}
$$

thus

$$
\tilde{G}_n^{(1)}(x,y,\nu) = x^\nu\frac{\tilde{N}_n(x,y,\nu)}{\tilde{D}^*_n(x,y,\nu)(-xy)^n e^{(x+y)}} \tag{2.17}
$$

2.3.2 Exponential Scaled Algorithm

The incomplete Bessel function is exponentially scaled by $e^{2\sqrt{xy}}$ in order to avoid overflow or underflow issues. As Slevinksy and Safouhi proposed, the condition for applying the $G_n^{(1)}$ transformation is that the integrand $f(t) = e^{-t-xy/t}/t^{\nu+1}$ satisfies the first order linear homogeneous differential equation ([102]):

$$
f(t) = -\frac{t^2}{t^2 - xy + (\nu+1)t}f'(t) \tag{2.18}
$$

The new integrand $f(t) = e^{-t-xy/t+2\sqrt{xy}}/t^{\nu+1}$ can be shown to also satisfy that equation. As t is the variable in the integrand therefore $\frac{d}{dt}2\sqrt{xy} = 0$. Since $e^{2\sqrt{xy}}K_\nu(x,y)$ satisfies the condition for the $G_n^{(1)}$ transformation hence the approximation can use the exponentially scaled quantities $\tilde{D}_n(x,y,\nu)$ and $\tilde{N}_n(x,y,\nu)$ which have forms:

$$
\begin{aligned}
\tilde{D}^*_n(x,y,\nu) &= \left(t^2\frac{\mathrm{d}}{\mathrm{d}t}\right)^n (t^{\nu+1}e^{-2\sqrt{xy}})\Big|_{t=x} \\
&= x^{\nu+1}e^{-2\sqrt{xy}}\sum_{i=0}^{r}\binom{n}{r}(-y)^{-r}\sum_{i=0}^{r}A_r^i x^i
\end{aligned}
\tag{2.19}
$$

$$
\begin{aligned}
\tilde{N}_n(x,y,\nu) &= \sum_{r=1}^{n}\binom{n}{r}\tilde{D}^*_{n-r}(x)\left(x^2\frac{\mathrm{d}}{\mathrm{d}x}\right)^{r-1}\left(x^{\sigma_0}e^{2\sqrt{xy}}f(x)\right) \\
&= \frac{e^{2\sqrt{xy}}}{x^\nu y}(-xy)^n\sum_{r=1}^{n}\binom{n}{r}\tilde{D}^*_{n-r}(x,y,\nu)\sum_{s=0}^{r-1}\binom{r-1}{s}y^{-s}\sum_{i=0}^{s}A_s^i(-x)^i
\end{aligned}
\tag{2.20}
$$

This gives the exponential scaled version of the approximation $\tilde{G}_n^{(1)*}(x,y,\nu) = e^{2\sqrt{xy}}\tilde{G}_n^{(1)}(x,y,\nu)$.

2.3.3 The Vectorized Incomplete Bessel Function

The `incompleteBesselK` function uses five sub-routines:

`combinatorial`
: The binomial coefficients

`SSFcoef` **with** $\nu - 1$
: The Slevensky-Safouhi coefficient when $\nu = \nu - 1$

`SSFcoef` **with** $-\nu - 1$
: The Slevensky-Safouhi coefficient when $\nu = -\nu - 1$

`GNUM`
: The numerator $\tilde{N}_n(x, y, \nu)$

`GDENUM`
: The denominator $\tilde{D}_n(x, y, \nu)$

Three of these, including `combinatorial` and `SSFcoef` with both $\nu - 1$ and $-\nu - 1$, do not involve `x` and `y`. These can thus be removed from the main routine and calculated once only for all `x` and `y` in the vectorized function.

The vectorized incomplete Bessel function `incompleteBesselKV` has the form:

```
incompleteBesselKV (x, y, nu,
                    tol = (.Machine$double.eps)^0.5,
                    nmax = 90, expon.scaled = FALSE)
```

`x, y`
: Vector values of the first and second arguments of the incomplete Bessel K function.

`nu`
: ν, scalar value of the order of the incomplete Bessel K function.

`tol`
: The tolerance for the ratio of difference between successive approximations of the incomplete Bessel K function.

`nmax`
: The maximum iterations for the approximation.

`expon.scaled`
: If `TRUE` then $e^{-2\sqrt{xy}} K_\nu(x, y, \nu)$ is returned.

2.4 Examples and Discussion

In this section, we will illustrate examples for some of the utility functions and then discuss the modification of the Bessel function.

2.4.1 `momIntegrated` function

We will first illustrate the `momIntegrated` function by calculating central and raw moments for the normal distribution. The example first calculates the central moments of normal distribution with $\mu = 2$, $\sigma = 1$ by the central moment formula

$$E\left[(X-\mu)^k\right] = \begin{cases} 0 & \text{if } k \text{ is odd} \\ \sigma^k (k-1)!! & \text{if } k \text{ is even} \end{cases} \tag{2.21}$$

It then uses the `momChangeAbout` function in the **DistrUtils** package to obtain raw moments from (2.21). Afterwards, the central moments and raw moments are calculated again by the `momIntegrated` function and compared with the previous result. The comparison is only for 1st moment and 8th moment, carried out by the `checkEquals` function in the **RUnit** package. The function returns `TRUE` if both terms are equal, `FALSE` otherwise.

```
normalMom <- function(order, mean, sd){
    if (order%%2 == 0){
        nMom <- sd^order * sqrt(2^order/pi) *
                gamma((order - 1)/2 + 1)
    } else {
      nMom <- 0
    }
    return(nMom)
}
mean <- 2
sd <- 1
centralMom <- sapply(1:8, normalMom, mean = mean, sd = sd)
centralMom
[1]   0   1   0   3   0  15   0 105
rawMom <- momChangeAbout("all", centralMom, mean, 0)
rawMom
[1]    2    5   14   43  142  499 1850 7193
m1 <- momIntegrated("norm", order = 1, mean = mean, sd = sd,
                    about = 0)
m8 <- momIntegrated("norm", order = 8, mean = mean, sd = sd,
                    about = 0)
checkEquals(rawMom[1], m1)
[1] TRUE
checkEquals(rawMom[8], m8)
```

```
[1] TRUE
cm1 <- momIntegrated("norm", order = 1, mean = mean, sd = sd,
                     about = m1)
cm8 <- momIntegrated("norm", order = 8, mean = mean, sd = sd,
                     about = m1)
checkEquals(centralMom[1], cm1)
[1] TRUE
checkEquals(centralMom[8], cm8)
[1] TRUE
```

2.4.2 pDist and qDist function

The examples for pDist function comprise the calculation of selected distribution functions with a variety of parameters, plus comparisons between the pDist function and implementations of the specific distribution function. The distribution included in the example is the gamma distribution with shape = 3 and tolerance = $10^{(-12)}$. The example first illustrates the output of the pDist function, then the comparison between pDist function and pgamma function result.

```
> n = 10
> x = rgamma(n, shape = 3)
> p = pDist("gamma", q = x, shape = 3, intTol = 10^(-12))
> p - pgamma(x, shape = 3)
 [1] -1.110223e-16  0.000000e+00  1.110223e-16  8.326673e-17  3.330669e-16
 [6]  0.000000e+00  1.110223e-16  2.803313e-15  0.000000e+00  5.551115e-17
```

The integration error, in the example, is no more than 10^{-8}. This is as expected in Section 2.2.2. The differences between the pDist function and the specific distribution function for the gamma distribution is barely more than 10^{-15}. The result indicates the pDist function is fairly accurate for at least the gamma distribution.

The examples for qDist function comprise the calculation of selected quantile functions using both methods, plus comparison between qDist function and implementations of the specific quantile function. The distribution included in the examples is again the gamma distribution.

Both examples use gamma distribution with shape = 3 but with different methods. We will start with the default method, that is, the Spline method.

```
> qs <- c(0.001, 0.025, 0.05, 0.1, 0.5, 0.9, 0.95, 0.975, 0.999)
> q <- qDist("gamma", p = qs, shape = 3)
> q - qgamma(qs, shape = 3)
[1]  1.703598e-05 -2.282323e-06  6.257252e-06 -9.850411e-07  1.500261e-05
[6]  1.205959e-05  2.187905e-05  1.610092e-05 -2.847725e-05
```

This example not only shows the output generated by the qDist function but also compares it with the specific quantile function of gamma distribution with shape = 3. The difference is around

10^{-5}. The Integration method with small tolerance would be expected to reduce this difference. Therefore `qDist` uses the Integrate method with 10^{-12} tolerance in the next example.

```
> q <- qDist("gamma", p = qs, shape = 3,
             method = "integrate", uniTol = 10^(-12))
> q - qgamma(qs, shape = 3)
[1]  9.660328e-13 -2.220446e-16 -2.220446e-16 -4.884981e-15  0.000000e+00
[6]  8.881784e-16 -1.065814e-14  8.881784e-16 -2.504663e-13
```

The differences in the example have significantly decreased to less than 10^{-12}. This illustrates the accuracy for `qDist` function is as expected in Section 2.2.3.

There are other distributions have been tested: the normal distribution, GHyp distribution, and hyperbolic distribution, to name a few. The `pDist` and `qDist` function appear to handle most of the distributions reasonably well. However the performance of `qDist` declines as the skew hyperbolic Student's t distribution ν decreases even when the step sizes for this distribution are carefully chosen, i.e. as mentioned in Section 2.2.3 when the tail is declining exponentially the step is just a linear function of the current distance from the mode. If the tail is declining only as a power of x, an exponential step is used. The main difficulty is that when the skew hyperbolic Student's t distribution $\nu \leq 4$, the central moments which relate to the variance, the skewness and the kurtosis, can be undefined, as discussed in detail in Chapter 3.

2.4.3 Incomplete Bessel K Function

The discussion for the incomplete Bessel K function has two aspects, efficiency and accuracy. The reference values for accuracy tests are generated from *Maple*, symbolic computation software which also performs highly accurate integration.

The efficiency tests are carried out by comparing the evaluation time of modified vectorized, exponential scaled vectorized and the original function. The test vector x comprises 4833 values within a normal value range, from 1×10^{-4} to 1000. The test vector y also contains 4833 values within a normal value range, from 2×10^{-6} to 10000. The ratios of the x and y values are designed to be contained in $\{0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50\}$. The value of ν is set as a fixed number. First we compare the `incompleteBesselK` function with the modified vectorized function `incompleteBesselKV`.

```
system.time(
  for(i in 1:length(x)){
    xval <- x[i]
    yval <- y[i]
    incompleteBesselK(xval, yval, nu,
                      tol = (.Machine$double.eps)^(0.85))
   }
)
```

```
  user  system elapsed
81.439  0.048 81.506

system.time(
  for(i in 1:length(x)){
    val <- x[i]
    yval <- y[i]
    incompleteBesselK(xval, yval, nu)
  }
)

  user  system elapsed
54.361  0.035 54.407
```

The current function evaluation takes 81.506 seconds with tolerance `(.Machine$double.eps)^(0.85)` and 54.407 seconds with tolerance `(.Machine$double.eps)^(0.5)`.

```
system.time(
  incompleteBesselKV(x,y,nu,
                     tol = (.Machine$double.eps)^(0.85))
)

  user  system elapsed
79.819  0.045 79.879

system.time(
  incompleteBesselKV(x,y,nu,
                     tol = (.Machine$double.eps)^(0.5))
)

  user  system elapsed
52.996  0.023 53.026
```

The modified function evaluation takes 79.879 seconds with tolerance `(.Machine$double.eps)^(0.85)` and 53.026 seconds with tolerance `(.Machine$double.eps)^(0.5)`. The vectorization slightly improves the performance of function. We then comparing the current function with vectorized and exponentially scaled function.

```
system.time(
  incompleteBesselKV(x,y,nu,
                     tol = (.Machine$double.eps)^(0.85),
                     expon.scaled = TRUE)
)
```

```
   user  system elapsed
 68.777   0.036  68.823

system.time(
  incompleteBesselKV(x,y,nu,
                     expon.scaled = TRUE)
)

   user  system elapsed
 30.724   0.014  30.743
```

It is found that the exponential scaling improves the efficiency of the calculation. The exponential scaled function evaluation takes 68.823 seconds with tolerance `(.Machine$double.eps)^(0.85)` and 30.743 seconds with tolerance `(.Machine$double.eps)^(0.5)`.

As mentioned in 2.3.1 and 2.3.2, the modifications are motivated by the instability of algorithm when x and ytake extreme values. Therefore, we will compare the robustness and accuracy for boundary values. The relative percentage error, to assess the performance of our functions, is calculated by

$$\left|\frac{\text{Maple IBF} - \text{IBF}}{\text{Maple IBF}}\right| * 100\% \tag{2.22}$$

We first assign x and y extreme values that would cause current published `incompleteBesselK` function to fail, i.e.

x	y	nu	MapleIBF
1e−08	1e−06	2	0.4999997
1e−07	1e−06	2	0.4999996
1e−05	1e−08	−10	3.630000e+55
1e−04	1e−08	−10	3.630000e+45
1e−03	1e−08	−10	3.630000e+35
1e−02	1e−08	−10	3.630000e+25
1e−01	1e−08	−10	3.630000e+15
2e+00	1e−08	−10	3.543585e+02

The `incompleteBesselK` and modified `incompleteBesselKV` functions give results as follows:

IBF	Error(%)	ScaledIBF	Error(%)	UnscaledIBF	Error(%)
NA	NA	0.4995116	0.097%	0.4984131	0.32%
NA	NA	0.4999997	2e−5%	0.4996338	0.073%
NA	NA	3.628800e+55	0.031%	3.628800e+55	0.031%
NA	NA	3.628800e+45	0.031%	3.628800e+45	0.031%
NA	NA	3.628800e+35	0.031%	3.628800e+35	0.031%

NA	NA	3.628800e+25	0.031%	3.628800e+25	0.031%
NA	NA	3.628800e+15	0.031%	3.628800e+15	0.031%
NA	NA	3.543585e+02	1.1e−8%	3.543585e+02	1.2e−8%

The results show that the `incompleteBesselK` function does not converge when either x or y is small. Our modified algorithm function not only converges but also gives results fairly close to the reference incomplete Bessel function values. The percentage errors are less than 0.1% for all scaled function results and almost all unscaled function results. However we have noticed that the percentage errors for scaled function are almost the same the unscaled for most x and ys.

We then assign x and y values that cause the current published `incompleteBesselK` function to converge to an incorrect result.

x	y	nu	MapleIBF
20	20	−10	5.40e−18
20	20	−2	1.10e−18
20	20	−0.5	8.95e−19
20	20	0	8.39e−19
20	20	0.5	7.89e−19
20	20	1	7.44e−19
20	20	2	6.64e−19
10	20	10	2.17e−14
10	20	20	7.71e−15
50	50	−0.5	4.85e−45
50	50	1	4.31e−45
50	50	2	4.00e−45
50	50	10	2.44e−45

The `incompleteBesselK` and modified `incompleteBesselKV` function give results as follows:

IBF	Error(%)	UnscaledIBF	Error(%)
−2.655221e−19	104.92%	5.401687e−18	0.031%
1.062089e−19	90.34%	1.099504e−18	0.045%
2.606352e−19	70.88%	8.946629e−19	0.038%
3.034539e−19	63.83%	8.392878e−19	0.034%
3.398683e−19	56.92%	7.890999e−19	0.013%
3.694221e−19	50.35%	7.435078e−19	0.066%
4.084956e−19	38.48%	6.640409e−19	0.006%
8.783273e−15	59.52%	2.167917e−14	0.096%
2.029216e−13	2531.93%	7.713683e−15	0.048%
9.230958e−46	80.97%	4.848076e−45	0.040%
1.403802e−45	67.43%	4.307802e−45	0.051%
1.660748e−45	58.48%	3.998809e−45	0.030%
2.084525e−45	14.57%	2.438327e−45	0.069%

The modified function appears to be more precise when the incomplete Bessel function values are small. Although the current version has supposedly converged, the percentage error can be as high as 2531.93% while the corresponding percentage error for modified function is only 0.048%.

The modified incomplete Bessel function function performance in the GIG distribution CDF approximation is discussed in Chapter 3.

CHAPTER 3

PROBABILITY FUNCTION AND QUANTILE FUNCTION ESTIMATION

3.1 The Generalized Inverse Gaussian Distribution

As pointed out in Chapter 1 the CDF approximation algorithm, proposed in [102] and implemented in the function `pgig`, has problems for extreme values of the arguments. These problems are the primary motivation to improve the current incomplete Bessel function calculation approach, discussed in Chapter 2. In the extreme tail of the GIG distribution, the current incomplete Bessel function approach tends to overflow/underflow as either x or y take extremely small values. Moreover, the incomplete Bessel function calculation approach is an approximation of the GIG distribution tail probability and can perform badly towards the centre of the distribution. To naively apply the approach as proposed in [102] can cause problems. To see this, recall the density function $f(x)$ of the GIG distribution

$$f(x) = \frac{1}{K_\lambda(\chi,\psi)} x^{\lambda-1} e^{-(\chi x^{-1}+\psi x)/2} \tag{3.1}$$

which can be re-written as

$$f(x) = \frac{(\psi/\chi)^{\lambda/2}/(2\sqrt{\chi\psi})}{K_\lambda(\sqrt{\chi\psi})} x^{\lambda-1} e^{-(\chi x^{-1}+\psi x)/2} \tag{3.2}$$

Therefore the CDF $F(x)$ is

$$F(x) = 1 - \frac{(\psi/\chi)^{\lambda/2}/(2\sqrt{\chi\psi})}{K_\lambda(\sqrt{\chi\psi})} \int_x^\infty t^{\lambda-1} e^{-(\chi t^{-1}+\psi t)/2} \tag{3.3}$$

It was shown in [103] that by variable substitution $t' = \psi t/2$, the CDF can be expressed as

$$F(x) = 1 - \frac{(\psi/\chi)^{\lambda/2}/(2\sqrt{\chi\psi})}{K_\lambda(\sqrt{\chi\psi})} K_{-\lambda}\left(\frac{\psi z}{2}, \frac{\chi}{2z}\right), \tag{3.4}$$

which can be approximated as [102]

$$F(x) = 1 - \frac{(\psi/\chi)^{\lambda/2}/(2\sqrt{\chi\psi})}{K_\lambda(\sqrt{\chi\psi})}\tilde{G}_n^{(1)}(\frac{\psi z}{2}, \frac{\chi}{2z}, -\lambda), \tag{3.5}$$

where $K_{-\lambda}\left(\frac{\psi z}{2}, \frac{\chi}{2z}\right)$ is the incomplete Bessel function with $x = \psi z/2$, $y = \chi/(2z)$, and $\tilde{G}_n^{(1)}$ is the $\tilde{G}_n^{(1)}$ transformation discussed in Chapter 2. Therefore when the quantiles fall into the tail area, i.e. $z \to 0$ or $z \to \infty$, either x or y of the incomplete Bessel function becomes extremely small.

For instance, let $p = 0.3$ for the GIG distribution with $\chi = 0.1$, $\psi = 0.1$ and $\lambda = -0.5$. The corresponding quantile obtained using Maple is therefore $q = 0.08290312$. The probability given by the function `pgig` however is

```
> pgig(0.08290312, param = c(0.1, 0.1, -0.5))
[1] 1
```

The `pgig` function has the format as

```
pgig(q, chi = 1, psi = 1, lambda = 1,
     param = c(chi, psi, lambda), lower.tail = TRUE,
     ibfTol = .Machine$double.eps^(0.85), nmax = 200)
```

`q`

The quantiles of the probabilities that are required to be computed.

`chi`

χ is the shape parameter of the distribution.

`psi`

ψ is the shape parameter of the distribution.

`lambda`

λ is the shape parameter of the distribution.

`param`

Parameter vector taking the form `c(chi, psi, lambda)`.

`lower.tail`

If `TRUE`, probabilities are $P(X \le x)$, otherwise as $P(X > x)$.

`ibfTol`

Value of tolerance to be passed to function `incompleteBesselK`.

`nmax`

Value of maximum order of the approximating series to be passed to `incompleteBesselK`.

As illustrated in Section 2.4.3, the modified incomplete Bessel function calculation approach proposed in Section 2.3 significantly increases the performance of the algorithm when either x or y is extremely small. Recalling the discussion in Section 2.3, the incomplete Bessel function can be approximated as

$$K_\nu(x,y) = e^{-2\sqrt{xy}}\tilde{G}_n^{(1)*}(x,y,\nu) \tag{3.6}$$

where $\tilde{G}_n^{(1)*}(x,y,\nu) = e^{2\sqrt{xy}}\tilde{G}_n^{(1)}(x,y,\nu)$.

Therefore the tail area of the GIG distribution can be approximated as

$$P(X>x) = \frac{(\psi/\chi)^{\lambda/2}/(2\sqrt{\chi\psi})}{K_\lambda(\sqrt{\chi\psi})} e^{-2\sqrt{\psi\chi/4}}\tilde{G}_n^{(1)*}(\frac{\psi z}{2},\frac{\chi}{2z},-\lambda) \tag{3.7}$$

However this modification is not sufficient to improve the bad performance when estimating a probability near the median of the GIG distribution. [98] has proposed a new algorithm using the G transformation which provides a remedy for the CDF approximation problem of the GIG distribution. This algorithm starts applying the G transformation on the CDF of GIG distribution instead of the incomplete Bessel function. The advantages of this algorithm are

- The overflow problem is avoided since the density function $f(x)$ will always have integral less than 1, i.e. $F(x) = \int_0^x f(t)\mathrm{d}t < 1$.
- The underflow problem is not a concern given the algorithm is approximating the tail probability.

In order to achieve these advantages, the algorithm first identifies the empirical median by sampling, then decides the respective tails that the required quantiles fall into and at last approximate the CDF in the correct tail. This algorithm is implemented in the function `pgigRAccel` which was provided by David Scott.

3.1.1 Examples

[98] showed that the function `pgigRAccel` performs reasonably well in 875 cases for a range of parameter values and integration limit x for both upper and lower tails of the GIG distribution. In this section, we will examine the performance of the function `pgigRAccel` and compare it to the `pgig` function. The `pgigRAccel` function computes upper/lower probabilities for 11 quantile values of the reference probabilities p, i.e.

$$p \in \{10^{-12}, 10^{-6}, 10^{-2}, 0.1, 0.3, 0.5, 0.8, 0.9, 1-10^{-2}, 1-10^{-6}, 1-10^{-12}\}$$

over a more comprehensive range of parameters which contains 1100 sets of values for (χ,ψ,λ), i.e. 12100 parameter and quantile combinations. For the 1100 sets of parameters, the values of χ and ψ were chosen from

$$\{0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}$$

and the value of λ was chosen from

$$\{-2,-1,-0.5,0,0.1,0.2,0.5,1,2,5,10\}.$$

Quantiles calculated by the black-box algorithm implemented in the **Runuran** package were used as reference. To assess the performance, the absolute percentage error

$$\text{abs.tol} = |\Pr(X)_{\text{RGIG}} - \Pr(X)_{\text{Reference}}| \times 100\% \tag{3.8}$$

is used.

Of these 12100 cases, when tolerance was set at 10^{-4}, the `pgigRAccel` function failed to converge at quantiles for which probabilities are 10^{-12} and $1-10^{-12}$ for only 4 sets of parameters, i.e. 8 cases.

Table 3.1 shows the maximum absolute percentage errors for each of the 11 quantiles considered for the remaining 12092 cases where convergence was achieved. The parameter set for which the maximum was achieved is also shown. The table indicates that the performance of the method is best in the tails of the distribution. Nonetheless even at the centre of the distribution, for $p = 0.5$, the absolute percentage error is less than 1%.

Table 3.1: Maximum absolute percentage error in the CDF using `pgigRAccel` when tol = 10^{-4} for reference probabilities of the GIG distribution

χ	ψ	λ	Probability (p)	Quantile (q)	Max(abs.tol) (%)
20	0.2	-1	10^{-12}	0.4000092	4.67991×10^{-9}
0.1	0.5	-2	10^{-6}	2.993899×10^{-3}	2.993899×10^{-3}
0.1	0.1	2	0.01	3.019218	7.797682×10^{-4}
0.1	0.1	1	0.1	2.306777	1.479599×10^{-2}
0.1	0.1	1	0.3	7.3732	8.305225×10^{-2}
0.2	1	-2	0.5	5.802001×10^{-2}	0.6446068
0.1	0.1	-1	0.8	0.213302	4.312542×10^{-2}
0.1	0.1	-1	0.9	0.433505	1.4796×10^{-2}
0.1	0.1	-2	0.99	0.331212	7.797694×10^{-4}
0.1	0.1	-2	$1-10^{-6}$	166.982	1.1262×10^{-5}
0.1	0.1	-2	$1-10^{-12}$	586.8214	4.552692×10^{-10}

When the tolerance for`pgigRAccel` was set to 10^{-6}, the function failed to converge for 32 cases which are the combination of 9 quantiles and 29 sets of parameters. Table 3.2 shows the maximum percentage error for the remaining 12068 cases, for the different quantiles examined. The pattern of results here is not as clear, with the maximum error occurring for $p = 0.1$. The maximum error though does not exceed 0.2%.

Table 3.2: Maximum absolute percentage error in the CDF using `pgigRAccel` when tol = 10^{-6} for reference probabilities of the GIG distribution

χ	ψ	λ	Probability (p)	Quantile (q)	Max(abs.tol) (%)
20	0.2	-1	10^{-12}	0.4000092	4.67991×10^{-9}
1	0.5	-2	10^{-6}	2.976499×10^{-2}	7.288361×10^{-6}
0.1	0.2	-2	0.01	7.526397×10^{-3}	0.171875
0.1	0.2	2	0.1	5.366639	2.933638×10^{-4}
0.2	0.1	2	0.3	2.204487	1.232702×10^{-3}
0.1	0.1	-2	0.5	6.047018×10^{-2}	2.407494×10^{-2}
5	5	-1	0.8	1.196581	1.560539×10^{-3}
0.1	0.2	-2	0.9	9.316819×10^{-2}	2.933754×10^{-4}
0.5	0.1	2	0.99	133.0102	6.25×10^{-2}
0.1	0.5	2	$1 - 10^{-6}$	668.0242	4.632568×10^{-6}
5	0.5	5	$1 - 10^{-12}$	149.9604	4.659051×10^{-10}

The results using `pgig` on the other hand are not so satisfactory. Although `pgig` converged for all sets of parameters when the requested incomplete Bessel function tolerance was set to 10^{-4}, the maximum absolute percentage error, is shown in Table 3.3, can be as large as 100%.

Table 3.3: Maximum absolute percentage error in the CDF using `pgig` when ibfTol = 10^{-4} for reference probabilities of the GIG distribution

χ	ψ	λ	Probability (p)	Quantile (q)	Max(abs.tol) (%)
0.1	0.1	10	10^{-12}	7.02772405	100
0.1	50	10	10^{-6}	0.05629494	99.9999
50	5	10	0.01	3.25999826	99.88508
50	50	10	0.1	1.01770786	97.07505
100	100	10	0.3	1.04835382	76.9508
100	100	10	0.5	1.10462577	38.39736
100	100	10	0.8	1.20107663	3.884678
100	100	10	0.9	1.25462441	0.8943765
100	100	10	0.99	1.39083186	1.822789×10^{-2}
2	0.2	-2	$1 - 10^{-6}$	43.77389765	4.199617×10^{-7}
0.2	20	10	$1 - 10^{-12}$	4.83144353	4.501954×10^{-10}

When the tolerance of incomplete Bessel function was set to 10^{-6}, the performance of `pgig` decreased for the lower tail area, even though convergence was achieved for all parameter sets. The maximum percentage error identified was 4.610571×10^{178}% as shown in Table 3.4.

Table 3.4: Maximum absolute percentage error in the CDF using `pgig` when ibfTol = 10^{-6} for reference probabilities of the GIG distribution

χ	ψ	λ	Probability (p)	Quantile (q)	Max(abs.tol) (%)
0.5	1	10	10^{-12}	0.7294163	5.563319×10^{172}
0.5	10	10	10^{-6}	0.2814747	4.610571×10^{178}
10	10	10	0.01	1.2293079	100.0956
50	50	10	0.1	1.01770786	97.07505
100	100	10	0.3	1.04835382	76.9508
0.1	0.1	0	0.5	1	50
0.1	0.1	-0.5	0.8	0.846621	20
100	100	10	0.9	1.25462441	0.8943765
100	100	10	0.99	1.39083186	1.822789×10^{-2}
100	100	10	$1-10^{-6}$	1.760015	1.324515×10^{-7}
0.2	20	10	$1-10^{-12}$	4.831445	4.501954×10^{-10}

From the comparisons in this section, we can conclude that the new CDF approximation approach for the GIG distribution provides a significant improvement in performance. In addition, as pointed out in [38], the Runuran algorithm becomes numerically unstable in the tail area whereas the algorithm behind the `pgigRAccel` function performs best in the tail.

3.2 The Generalized Hyperbolic Distribution

As mentioned in Chapter 1, the GHyp CDF and quantile function approximation algorithms implemented in the `pghyp` and `qghyp` functions of the **GeneralizedHyperbolic** package version 0.2-0 respectively have several problems. These concerns are addressed and some solutions found in my research. This section illustrates the methodology and test results of the updated `pghyp` and `qghyp` functions.

Before the discussion of the improvements from my research, it is worthwhile to describe the `pghyp` and `qghyp` functions of the **GeneralizedHyperbolic** package version 0.2-0.

Let x denote values for which the CDF $F(x)$ is to be evaluated and $f(x)$ denote the probability density function. Then the CDF $F(x)$ approximation algorithm is:

1. Break $f(x)$ into 8 regions where the cut-off points are x_{Tiny}, x_{Small}, x_{Low}, x_{Mode}, x_{High}, x_{Large} and x_{Huge}. These cut-off points are found as follows:

   a) x_{Tiny} and x_{Huge} are calculated by solving $f(x_{\text{Tiny}}) = f(x_{\text{Huge}}) = 10^{-10}$.

   b) x_{Small} and x_{Large} are calculated by solving $f(x_{\text{Small}}) = f(x_{\text{Large}}) = 10^{-6}$.

   c) x_{mode} is the distribution mode found by maximizing $\log f(x)$.

d) x_{Low} is determined by the derivative of $f(x_{\text{Low}})$, $f'(x_{\text{Low}}) = c * \max[f'(x_{\text{Mode}}), f'(x_{\text{Small}})]$, where c is a scaling factor with default value 0.3.

e) x_{High} is determined by the derivative of $f(x_{\text{High}})$, $f'(x_{\text{High}}) = c * \max[f'(x_{\text{Large}}), f'(x_{\text{Mode}})]$, where c is a scaling factor with default value 0.3.

2. $F(x \leq x_{\text{Tiny}}) = 0$ and $F(x \geq x_{\text{Huge}}) = 1$.

3. Each region is numerically integrated using the `integrate` function to obtain the CDF.

The pghyp function has the format

```
pghyp(q, mu = 0, delta = 1, alpha = 1, beta = 0, lambda = 1,
      param = c(mu, delta, alpha, beta, lambda),
      small = 10^(-6), tiny = 10^(-10),
      deriv = 0.3, subdivisions = 100, accuracy = FALSE, ...)
```

`q`

The quantiles of the probabilities that are required to be computed.

`mu`

μ, the location parameter of the distribution.

`delta`

δ, the scale parameter of the distribution.

`alpha`

α, the tail parameter of the distribution.

`beta`

β, the skewness parameter of the distribution.

`lambda`

λ, the shape parameter of the distribution.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`small`

Size of a small difference between the density function and zero or one i.e. $f(x_{\text{Small}})$ and $f(x_{\text{Large}})$.

`tiny`

Size of a tiny difference between the density function and zero or one i.e. $f(x_{\text{Small}})$ and $f(x_{\text{Large}})$.

`deriv`

Value of $c \in [0,1]$. It determines the point where the derivative becomes substantial, compared to its maximum value.

`accuracy`

Uses the accuracy calculated by the `integrate` function to determine the accuracy of the distribution function calculation.

`subdivisions`

The maximum number of subdivisions used to integrate the density returning the CDF.

`...`

Passes arguments to the one-dimensional root searching function `uniroot`.

The GHyp distribution quantile function approximation, implemented in qghyp, adopts a similar approach to the CDF approximation algorithm in breaking the real line into eight regions. Denote the probabilities for which quantiles are required by p and the quantile function by $F^{-1}(x)$. The algorithm is

1. Break the real line into 8 regions where the cut-off points are x_{Tiny}, x_{Small}, x_{Low}, x_{Mode}, x_{High}, x_{Large} and x_{Huge} as for the CDF.
2. Sort the p values into appropriates region $[0, F(x_{\text{Tiny}})]$, $(F(x_{\text{Tiny}}), F(x_{\text{Small}})]$, $(F(x_{\text{Small}}), F(x_{\text{Low}})]$, $(F(x_{\text{Low}}), F(x_{\text{Mode}})]$, $(F(x_{\text{Mode}}), F(x_{\text{High}})]$, $(F(x_{\text{High}}), F(x_{\text{Large}})]$, $(F(x_{\text{Large}}), F(x_{\text{Huge}})]$, $[F(x_{\text{Huge}}), 1]$
3. Set $F^{-1}(p) = -\infty$ if $p \in [0, F(x_{\text{Tiny}})]$
4. Set $F^{-1}(p) = \infty$ if $p \in [F(x_{\text{Huge}}), 1]$
5. In the 6 inner regions the `splinefun` function is used to perform cubic spline interpolation of values of $F(x)$ in each region. The quantiles are then found by solving the equation `splinefun`$(x) - p = 0$ using the one dimensional root searching function `uniroot`.

The qghyp function has the format

```
qghyp(p, mu = 0, delta = 1, alpha = 1, beta = 0, lambda = 1,
      param = c(mu, delta, alpha, beta, lambda),
      small = 10^(-6), tiny = 10^(-10),
      deriv = 0.3, nInterpol = 100, subdivisions = 100, ...)
```

`p`

The probabilities of the quantiles that are required to be computed.

`mu`

μ, the location parameter of the distribution.

`delta`
: δ, the scale parameter of the distribution.

`alpha`
: α, the tail parameter of the distribution.

`beta`
: β, the skewness parameter of the distribution.

`lambda`
: λ, the shape parameter of the distribution.

`param`
: Parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`small`
: Size of a small difference between the density function and zero or one i.e. $f(x_{\text{Small}})$ and $f(x_{\text{Large}})$.

`tiny`
: Size of a tiny difference between the density function and zero or one i.e. $f(x_{\text{Small}})$ and $f(x_{\text{Large}})$.

`deriv`
: Value of $c \in [0,1]$. It determines the point where the derivative becomes substantial, compared to its maximum value.

`nInterpol`
: The number of points used for cubic spline interpolation.

`subdivisions`
: The maximum number of subdivisions used to integrate the density returning the CDF.

...
: Passes arguments to the one-dimensional root searching function `uniroot`.

The CDF and quantile approximation approaches described above break the real line into an unnecessarily large number of intervals without achieving great accuracy. For instance, let $p = 10^{-15}$, so that we should have $F(F^{-1}(p)) = 10^{-15}$. However the result of using the pghyp and qghyp functions is

```
> qghyp(10^{-15})
[1] -22.81846
> pghyp(qghyp(10^{-15}))
[1] 0
```

```
> dghyp(qghyp(10^{-15}))
[1] 9.999973e-11
```

When $f(F^{-1}(p)) < 10^{-10}$, pghyp($F^{-1}(p)$) is set equal to 0 without calculation. In addition, the qghyp function approximation results are not reliable either:

```
> qghyp(10^{-16})
[1] -22.81846
> qghyp(10^{-17})
[1] -22.81846.
```

3.2.1 Methodology

3.2.1.1 The CDF Approximation Approach

The discussion of the approaches used for the approximate GHyp distribution CDF and quantile functions is mainly concentrated on the quantile function. However the discussion of the CDF approximation approach is still necessary as the quantile function approximation will not be stable without a well-behaved CDF approximation.

Let X be a GHyp distributed random variate with density function $f(x)$. The CDF $F(x)$ approximation approach is as follows.

1. Calculate the mode of the GHyp distribution x_{Mode} and divide the $f(x)$ into 2 parts at x_{Mode}.

2. If $x \leq x_{\text{Mode}}$, $F(x)$ is numerically integrated over $[-\infty, x]$ using the `integrate` function.

3. If $x > x_{\text{Mode}}$, $F(x)$ is numerically integrated over $[x, \infty]$ using the `integrate` function.

The improved pghyp function has the format

```
pghyp(q, mu = 0, delta = 1, alpha = 1, beta = 0, lambda = 1,
      param = c(mu, delta, alpha, beta, lambda),
      lower.tail = TRUE, subdivisions = 100,
      intTol = .Machine$double.eps^0.25, valueOnly = TRUE, ...)
```

`q`

The quantiles of the probabilities required to be computed.

`mu`

μ, the location parameter of the distribution.

`delta`

δ, the scale parameter of the distribution.

`alpha`
: α, the tail parameter of the distribution.

`beta`
: β, the skewness parameter of the distribution.

`lambda`
: λ, the shape parameter of the distribution.

`param`
: Parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`lower.tail`
: If `TRUE`, $F(X \leq x)$ otherwise $F(X > x)$.

`subdivisions`
: The maximum number of subdivisions used to integrate the density and determine the accuracy of the CDF calculation.

`intTol`
: The requested relative accuracy of `integrate` function.

`valueOnly`
: If `valueOnly = TRUE` calls to pghyp only return the value obtained for the integral. If `valueOnly = FALSE` an estimate of the accuracy of the numerical integration is also returned.

...
: Passes arguments to the one dimensional root searching function `uniroot`.

3.2.1.2 The Quantile Function Approximation Approach

Since the CDF of the GHyp distribution cannot be specified explicitly, efficient and accurate approximatiion of the GHyp quantile function are issues for the qghyp function. In fact, there are numerous situations where only efficiency or accuracy is required. For instance, the Q-Q plot requires that the qghyp function efficiently generates a large number of quantiles but since its purpose is for plotting only, it is not required to be extremely accurate. For this reason, the improved GHyp distribution quantile function approximation consists of two approaches, the Integrate approach and the Spline approach, for different situations.

The Integrate approach is accurate but time consuming with a large number of quantiles. Let p denote the probabilities for which quantiles are required, $F(x)$ denote the CDF, $f(x)$ the density function, and $F^{-1}(x)$ the quantile function. The algorithm is then as follows:

1. Compute the GHyp distribution mode x_{Mode}.

2. Determine the searching interval for the one-dimensional root search function `uniroot`. This is the time consuming part, as the pghyp function needs to be applied to each candidate value i.e. numerical integration is carried out at each step.

   a) If $p \leq F(x_{\text{Mode}})$, the interval is $[F(x_{\text{Low}}), F(x_{\text{Mode}})]$. The lower bound x_{Low} is obtained by iteratively subtracting $\sqrt{\text{Var}(X)}$ from x_{Mode} until $F(x_{\text{Low}}) < p$.

   b) If $p > F(x_{\text{Mode}})$, the interval is $[F(x_{\text{Mode}}), F(x_{\text{High}})]$. The upper bound x_{High} is obtained by iteratively adding $\sqrt{\text{Var}(X)}$ from x_{Mode} until $F(x_{\text{High}}) > p$.

3. Apply the `uniroot` function over the interval to search for the roots of $F_{\text{Spline}}(x) - p = 0$ for the different values of p, i.e. the required quantiles.

An alternative method to compute the GHyp quantile function is the Spline approach which uses the `splinefun` function to interpolate the major part of the GHyp distribution. However one feature of the GHyp distribution is the semi-heavy tails where the density is extremely sparse. Within the extreme tails, the quantiles are always approximated by the Integrate approach as the `splinefun` function does not work well when the distribution is very flat. The Spline approach may not be as accurate as the Integrate approach, however the advantage of this approach is the efficiency. Again, let p be the probabilities for which quantiles are required. The algorithm is then as follows:

1. Compute the cut-off bounds x_{Low}, x_{High} for the `splinefun` function. Any quantiles falling outside of the range are approximated by the Integrate approach. The criteria for x_{Low} and x_{High} are that x_{Low} (x_{High}) is the largest (smallest) x for which $f(x_{\text{Low}}) \leq 10^{-5}$ ($f(x_{\text{High}} \leq 10^{-5}$).

2. Divide the interval $[x_{\text{Low}}, x_{\text{High}}]$ into a predetermined number of sub-intervals by x_i, $i = 1, \ldots, n$ where $x_1 = x_{\text{Low}}$ and $x_n = x_{\text{High}}$.

3. Compute the $F(x_i)$ using the pghyp function.

4. Perform cubic spline interpolation of the points $(x_i, F(x_i))$ using the `splinefun` function.

5. Apply the `uniroot` function over the interval to search for the roots of $F_{\text{Spline}}(x) - p = 0$ for the different values of p, i.e. the required quantiles.

The improved qghyp function has the format

```
qghyp(p, mu = 0, delta = 1, alpha = 1, beta = 0, lambda = 1,
      param = c(mu, delta, alpha, beta, lambda),
      lower.tail = TRUE, method = c("spline","integrate"),
      nInterpol = 501, uniTol = .Machine$double.eps^0.25,
      subdivisions = 100, intTol = uniTol, ...)
```

`p`

The probabilities of the quantiles required to be computed.

`mu`

μ, the location parameter of the distribution.

`delta`

δ, the scale parameter of the distribution.

`alpha`

α, the tail parameter of the distribution.

`beta`

β, the skewness parameter of the distribution.

`lambda`

λ, the shape parameter of the distribution.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`lower.tail`

If `TRUE`, $F(X \leq x)$ otherwise $F(X > x)$.

`method`

The character string to specify the quantile approximation approach.

`nInterpol`

Number of points used for cubic spline interpolation of the CDF.

`uniTol`

The convergence tolerance of `uniroot` function.

`subdivisions`

The maximum number of subdivisions used to integrate the density and determine the accuracy of the CDF calculation.

`intTol`

The requested relative accuracy of `integrate` function.

`...`

Passes arguments to the one-dimensional root searching function `uniroot`.

The improved CDF and quantile approximation approaches, implemented in the modified pghyp and qghyp functions, can be shown to provide more accurate results. Recall the example which illustrates the problem of existing approaches earlier in this section. The result of the improved pghyp and qghyp functions is

```
> qghyp(10^{-15})
[1] -34.33912
> pghyp(qghyp(10^{-15}))
[1] 9.999916e-16
> dghyp(qghyp(10^{-15}))
[1] 9.996018e-16
```

```
> qghyp(10^{-16})
[1] -36.64257
> pghyp(qghyp(10^{-16}))
[1] 9.999985e-17
```

```
> qghyp(10^{-17})
[1] -38.94592
> pghyp(qghyp(10^{-17}))
[1] 9.999971e-18
```

The improved pghyp function is able to estimate the probabilities in the extreme tails of the distribution, which were set to 0 or 1 in the previous approach. Moreover the improved quantile approximation approach estimates the quantiles within the extreme tails fairly accurately. In the next section, more extensive test results are presented to illustrate the accuracy and efficiency of the improved approximation approach.

3.2.2 Result

3.2.2.1 Accuracy Tests

The basic idea of testing the accuracy is to compare the difference between given p and `pghyp(qghyp(p))`, and the difference between given q and `qghyp(pghyp(q))`.

The test is designed to be as comprehensive as possible. Thus the comparisons use several sets of distribution parameters and different values of tolerance of the `integrate` function. Moreover, the given vector p comprises the probabilities $\{0, 0.001, 0.025, 0.3, 0.5, 0.7, 0.975, 0.999, 1\}$ and the given vector q comprises the quantiles $\{1, -\infty, 0, -1, +\infty, q_{\text{le}}, q_{\text{ue}}, q_{\text{mode}}\}$, where q_{le} and q_{ue} are randomly generated extreme values in the lower tail and upper tail respectively.

For each given p or q, the test returns the maximum difference from testing the distribution parameters sets under each tolerance. From Table 3.5, there is not a lot of difference between the Spline method at default tolerance. However the accuracy gap increases as the tolerance for integration increases. At 10^{-12} tolerance, the integration method can be as accurate as 10^{-16}. On

the other hand, the spline method can only be as accurate as 10^{-11}. Overall, the accuracy of both methods is satisfactory.

Table 3.5: Maximum Difference between Actual p (or q) and Approximate p (or q)

	Default Tol		10^{-8} Tol		10^{-10} tol		10^{-12} Tol	
	Spline	Integrate	Spline	Integrate	Spline	Integrate	Spline	Integrate
p = 0	0	0	0	0	0	0	0	0
p = 0.001	7.98×10^{-8}	8.77×10^{-8}	8.40×10^{-9}	8.49×10^{-12}	8.40×10^{-19}	6.94×10^{-14}	8.40×10^{-9}	9.81×10^{-16}
p = 0.025	3.02×10^{-6}	2.14×10^{-6}	2.89×10^{-8}	1.27×10^{-10}	2.89×10^{-8}	1.16×10^{-12}	2.89×10^{-8}	1.25×10^{-14}
p = 0.3	1.99×10^{-5}	1.15×10^{-5}	5.05×10^{-7}	1.64×10^{-9}	5.05×10^{-7}	1.40×10^{-11}	5.05×10^{-7}	1.00×10^{-13}
p = 0.5	1.47×10^{-5}	1.26×10^{-5}	1.34×10^{-7}	1.11×10^{-9}	1.35×10^{-7}	8.20×10^{-12}	1.35×10^{-7}	9.81×10^{-14}
p = 0.7	1.73×10^{-5}	1.15×10^{-5}	3.13×10^{-7}	1.07×10^{-9}	3.13×10^{-7}	1.40×10^{-11}	3.13×10^{-7}	1.00×10^{-13}
p = 0.975	1.37×10^{-6}	1.44×10^{-6}	2.45×10^{-9}	8.29×10^{-11}	2.45×10^{-9}	5.40×10^{-13}	2.45×10^{-9}	1.05×10^{-14}
p = 0.999	5.31×10^{-8}	5.40×10^{-8}	4.09×10^{-9}	1.77×10^{-12}	5.35×10^{-11}	1.52×10^{-14}	5.35×10^{-11}	3.33×10^{-16}
p = 1	0	0	0	0	0	0	0	0
q = $-\infty$	0	0	0	0	0	0	0	0
q = -1	3.05×10^{-5}	3.02×10^{-5}	2.14×10^{-6}	2.45×10^{-9}	2.14×10^{-6}	2.35×10^{-11}	2.14×10^{-6}	2.44×10^{-13}
q = 0	2.83×10^{-5}	3.02×10^{-5}	7.40×10^{-7}	2.33×10^{-9}	7.40×10^{-7}	2.42×10^{-11}	7.40×10^{-7}	2.34×10^{-13}
q = 1	2.94×10^{-5}	3.05×10^{-5}	7.06×10^{-7}	2.49×10^{-9}	7.06×10^{-7}	2.39×10^{-11}	7.06×10^{-7}	2.39×10^{-13}
q = $+\infty$	0	0	0	0	0	0	0	0
q = q_{le}	2.98×10^{-5}	3.04×10^{-5}	1.39×10^{-6}	2.49×10^{-9}	1.39×10^{-6}	2.49×10^{-11}	1.39×10^{-6}	1.79×10^{-13}
q = q_{ue}	3.01×10^{-5}	3.04×10^{-5}	8.23×10^{-8}	2.40×10^{-9}	8.39×10^{-8}	3.35×10^{-10}	8.39×10^{-8}	3.11×10^{-10}
q = q_{mode}	2.83×10^{-5}	0	6.3×10^{-7}	0	6.3×10^{-7}	0	6.3×10^{-7}	0

3.2.2.2 Efficiency Tests

As mentioned in Section 3.2.1, efficiency is the major concern which promoted the use of the spline function in the first place. Therefore the `system.time` function is used in the test program to time how long it takes to evaluate the qghyp function with different quantile sizes.

Based on the algorithms, the Integrate method is expected to be more efficient for very small quantile sizes since the Spline method simulates the probability function i.e. performs numeric integration at least 501 times. However, once the probability function is approximated, the Spline method becomes very efficient. As a matter of fact, it is more efficient than the Integrate method for larger quantile sizes. The test results in Table 3.6 are as expected. The Spline method apparently makes significant improvement in terms of efficiency when the sample size ≥ 500. In addition, the existing code for the qghyp function is tested for comparison. Table 3.6 shows that the existing method is slower than both the Spline and Integrate methods when the quantile size is small. Although its evaluation becomes more rapid than the Integrate method as the number of quantiles increases, it is still significantly slower than Spline method. The modification of the qghyp function therefore is an improvement.

Table 3.6: Comparison of Time Required to Calculate Various Numbers of Quantiles (in seconds)

Quantile Size	Existing Method	Spline Method	Integrate Method
10	2.37	0.28	0.21
50	2.51	0.29	0.86
100	2.59	0.33	1.71
500	2.76	0.67	8.14
1000	3.04	1.11	15.94

3.3 The Skew Hyperbolic Student-*t* Distribution

The approximation of the CDF and quantile function of the skew hyperbolic Student-*t* distribution has always been challenging due to the complexity of the density function and the special behaviour of its tails. The second, third, and fourth central moments of the distribution, described in [97], can be expressed as

$$\bar{\mathrm{M}}_2 = \frac{2\delta^4\beta^2}{(\nu-2)^2(\nu-4)} + \frac{\delta^2}{\nu-2} \tag{3.9}$$

when $\nu > 4$,

$$\bar{\mathrm{M}}_3 = \frac{16\delta^6\beta^3}{(\nu-2)^2(\nu-4)(\nu-6)} + \frac{6\delta^4\beta}{(\nu-2)^2(\nu-4)} \tag{3.10}$$

when $\nu > 6$,

$$\bar{\mathrm{M}}_4 = \frac{12\delta^8\beta^4(\nu+10)}{(\nu-2)(\nu-4)(\nu-6)(\nu-8)} + \frac{12\delta^6\beta^2(\nu+2)}{(\nu-2)^3(\nu-4)(\nu-6)} + \frac{3\delta^4}{(\delta-2)(\delta-4)} \tag{3.11}$$

when $\nu > 8$.

Therefore when $\nu \leq 4$, the variance, skewness and kurtosis are undefined or infinite. However it is worthwhile to investigate this distribution as those properties are useful for finance asset returns, which have both heavy and semi-heavy tails, and often feature skewness.

As mentioned in Chapter 1, the current approaches to approximate the CDF and the quantile function do not perform well when ν is small. A brief description of these approaches and functions is helpful at this point.

The CDF and the quantile function of the skew hyperbolic Student-*t* distribution are approximated by the same approaches as the GHyp distribution described in Section 3.2.1. The only difference is the step-size when determining the search interval for the one-dimensional root search function `uniroot` in the Integrate approach for quantile function approximation. Recall that for the GHyp distribution quantile function approximation, the search interval is determined by

1. If $p \leq F(x_{\text{Mode}})$, the interval is $[F(x_{\text{Low}}), F(x_{\text{Mode}})]$. The lower bound x_{Low} is obtained by iteratively subtracting $\sqrt{\text{Var}(X)}$ from x_{Mode} until $F(x_{\text{Low}}) < p$.

2. If $p > F(x_{\text{Mode}})$, the interval is $[F(x_{\text{Mode}}), F(x_{\text{High}})]$. The upper bound x_{High} is obtained by iteratively adding $\sqrt{\text{Var}(X)}$ to x_{Mode} until $F(x_{\text{High}}) > p$.

In this case the step-size is $\sqrt{\text{Var}(X)}$. For the skew hyperbolic Student-t distribution, the step-size is determined as follows:

$$\begin{array}{ll} \delta, & \text{when in the exponential tail} \\ \delta|\beta|(\nu\delta)^{-2/\nu}, & \text{when in the polynomial tail} \\ e^{\delta/\nu}, & \text{when } \beta = 0 \end{array}$$

The `pskewhyp` function which implements the CDF function has the format

```
pskewhyp(q, mu = 0, delta = 1, beta = 1, nu = 1,
         param = c(mu, delta, beta, nu), log.p = FALSE,
         lower.tail = TRUE, subdivisions = 100,
         intTol = .Machine$double.eps^0.25,
         valueOnly = TRUE, ...)
```

The arguments are

`q`
: The quantiles of the probabilities required to be computed.

`mu`
: The location parameter μ, of the distribution.

`delta`
: The scale parameter δ, of the distribution.

`beta`
: The skewness parameter β, of the distribution.

`nu`
: The shape parameter ν, of the distribution.

`param`
: The parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`log.p`
: If `TRUE`, the log probability is calculated

`lower.tail`
: If `TRUE`, $F(X \leq x)$, otherwise $F(X > x)$.

`subdivisions`

The maximum number of subdivisions used to integrate the density and determine the accuracy of the CDF calculation.

`intTol`

The requested relative accuracy of `integrate` function.

`valueOnly`

If `valueOnly = TRUE` calls to `pghyp` only return the value obtained for the integral. If `valueOnly = FALSE` an estimate of the accuracy of the numerical integration is also returned.

`...`

Passes arguments to the one-dimensional root searching function `uniroot`.

The `qskewhyp` function which implements the quantile function has the format

```
qskewhyp(p, mu = 0, delta = 1, beta = 1, nu = 1,
         param = c(mu,delta, beta, nu),
         lower.tail = TRUE, log.p = FALSE,
         method = c("spline","integrate"),
         nInterpol = 501, uniTol = .Machine$double.eps^0.25,
         subdivisions = 100, intTol = uniTol, ...)
```

The arguments are

`p`

The probabilities of the quantiles required to be computed.

`mu`

μ, the location parameter of the distribution.

`delta`

δ, the scale parameter of the distribution.

`beta`

β, the skewness parameter of the distribution.

`nu`

ν, the shape parameter of the distribution.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta, lambda)`.

`lower.tail`

If `TRUE`, $F(X \leq x)$ otherwise $F(X > x)$.

`log.p`
: If `TRUE`, `p` are treated as `log(p)`.

`method`
: The character string to specify the quantile approximation approach.

`nInterpol`
: Number of points used for cubic spline interpolation of the CDF.

`uniTol`
: The convergence tolerance of the `uniroot` function.

`subdivisions`
: The maximum number of subdivisions used to integrate the density and determine the accuracy of the CDF calculation.

`intTol`
: The requested relative accuracy of `integrate` function.

`...`
: Passes arguments to the one-dimensional root searching function `uniroot`.

Below is an example to illustrate the performance of these two functions when ν is small. Recall the tail behavior from **??**. When $\beta > 0$, the right hand tail (as $x \to \infty$) is the polynomially decaying tail which is the heavy tail. However with ν small, the quantile function approximation approach fails even for $p = 0.9$:

```
> qskewhyp(0.9, param = c(0, 1, 2, 0.5))
Error in integrate(dskewhypInt, q[i], Inf,
            subdivisions = subdivisions,  :
  the integral is probably divergent
```

As mentioned in Chapter 1, there is an alternative approach the so-called black-box approach, proposed in [38], implemented in the **Runuran** package. Although this approach has the ability to provide a reliable approximation result, as pointed out in Chapter 4, it requires rather complicated machinery to at least store the fairly large table. Moreover, the approximation procedure of this approach is relatively complex, i.e. prior to evaluation of the approximate CDF, the `UNU.RAN` object which is the critical part of the algorithm is required to be created. Since the black-box approach was primarily developed as a universal random number generater, the algorithm and the functions which implement the algorithm are reviewed in detail in Chapter 4. In this section, we will only introduce one function which evaluates the approximate CDF of the `UNU.RAN` object for a continuous or discrete distribution. The function description is as follows.

```
up(obj, x)
```

The arguments are

`obj`

The `UNU.RAN` object for a continuous or discrete distribution.

`x`

The quantiles of the probabilities required to be computed.

In this section, we propose a new approach which transforms the density function by the split t transformation, then integrates the transformed integrand numerically by the Gaussian quadrature with Richardson extrapolation or the `integrate` function. This approach is straightforward, relatively stable, and addresses some of the previous concerns.

3.3.1 The Split-*t* Transformation

The split-t transformation technique used in our approach was proposed by A. Genz and R. E. Kass for integrands that have a dominant peak ([49]). The integration over an infinite interval is transformed to a finite interval, and should be easier to compute. In this section, this technique is modified to consider only one side of the distribution. This is because the skew hyperbolic distribution possesses the property that the two sides of the distribution curve can be swapped around by manipulating the β and x values. To prove this, consider x in the lower tail, i.e. $x < \mu$. Let $\hat{x} = 2\mu - x$ and $\hat{\beta} = -\beta$. Then x and $\hat{x}$ are symmetrical around μ. Let C be the normalizing constant in the density function, that is,

$$C = \frac{2^{(1-\nu)/2}\delta^{\nu}|\hat{\beta}|^{(\nu+1)/2}}{\gamma(\nu/2)\sqrt{\pi}}. \tag{3.12}$$

The density function is

$$\begin{aligned}
f(\hat{x}) &= C\frac{e^{\hat{\beta}(\hat{x}-\mu)}K_{(\nu+1)/2}(\sqrt{\hat{\beta}^2(\delta^2+(\hat{x}-\mu)^2)})}{(\sqrt{\delta^2+(\hat{x}-\mu)^2})^{(\nu+1)/2}} \\
&= C\frac{e^{-\beta(2\mu-x-\mu)}K_{(\nu+1)/2}(\sqrt{(-\beta)^2(\delta^2+(2\mu-x-\mu)^2)})}{(\sqrt{\delta^2+(2\mu-x-\mu)^2})^{(\nu+1)/2}} \\
&= C\frac{e^{-\beta(\mu-x)}K_{(\nu+1)/2}(\sqrt{(-\beta)^2(\delta^2+(\mu-x)^2)})}{(\sqrt{\delta^2+(\mu-x)^2})^{(\nu+1)/2}} \\
&= C\frac{e^{\beta(x-\mu)}K_{(\nu+1)/2}(\sqrt{(\beta)^2(\delta^2+(x-\mu)^2)})}{(\sqrt{\delta^2+(x-\mu)^2})^{(\nu+1)/2}} \\
&= f(x)
\end{aligned} \tag{3.13}$$

Therefore we will only implement the transformation for positive x. The algorithm is set out as follows. Consider the skew hyperbolic distribution CDF of interest

$$F(x) = \int_{-\infty}^{x} f(t)\mathrm{d}t, \tag{3.14}$$

where $f(t)$ is the density function as defined as $f(t;\mu,\delta,\beta,\nu)$ given by

$$f(t) = \frac{2^{(1-\nu)/2}\delta^{\nu}|\beta|^{(\nu+1)/2}e^{\beta(t-\mu)}K_{(\nu+1)/2}\left(\sqrt{\beta^2[\delta^2+(t-\mu)^2]}\right)}{\gamma(\nu/2)\sqrt{\pi}\left[\sqrt{\delta^2+(t-\mu)^2}\right]^{(\nu+1)/2}} \tag{3.15}$$

Adjust t with location parameter μ and scale parameter δ as

$$\begin{aligned} t^* &= \frac{t-\mu}{\delta} \\ \beta^* &= \beta\delta. \end{aligned} \tag{3.16}$$

The probability density function $f(t)$ is therefore simplified as $f^*(t^*)$ by considering the location- and scale-invariant parameters ν, β^*. The density function of $f^*(t^*;0,1,\beta^*,\nu)$ is therefore

$$f^*(t^*) = \frac{|\beta^*|^{(\nu+1)/2}e^{\beta^* t^*}K_{(\nu+1)/2}\left(\sqrt{(\beta^*)^2[1+(t^*)^2]}\right)}{\gamma(\nu/2)\sqrt{\pi}\left[\sqrt{1+(t^*)^2}\right]^{(\nu+1)/2}} \tag{3.17}$$

The CDF can then be written as

$$F(x) = \int_{-\infty}^{(x-\mu)/\delta} f^*(t^*)\mathrm{d}t^*. \tag{3.18}$$

[49] proposed that $f^*(t^*)$ can be modelled by the scaled Student-t distribution density function $g(t^*)$ to take account of tail behavior, i.e.

$$g(t^*) = K\left[1+\frac{(t^*)^2}{\hat{\nu}\hat{\delta}^2}\right]^{-(\hat{\nu}+1)/2} \tag{3.19}$$

where K is the normalizing constant, $\hat{\nu}>0$ is the degree of freedom of the Student-t distribution and $\hat{\delta}$ is the scale parameter. Let $G(t^*)$ be the CDF of the $\hat{\delta}$ scaled Student-t distribution with degrees of freedom $\hat{\nu}$. Then the CDF of the skew hyperbolic distribution can be derived as

$$F(x) = \hat{\delta}\int_0^{z_0} \frac{f^*(\hat{\delta}G^{-1}(t^*))}{g(G^{-1}(t^*))}\mathrm{d}t^* \tag{3.20}$$

where $z_0 = G\left(\frac{(x-\mu)/\delta}{\hat{\delta}}\right)$. The determination of the values for $\hat{\delta}$ and $\hat{\nu}$ based on several evaluations of $g(t^*)$ is a nonlinear approximation problem that can be solved using a variety of methods. A. Genz and R. E. Kass in [49] propose examining the value of $f(t^*)$ at several points t^* then fitting values $\hat{\delta}$ and $\hat{\nu}$. The value of $\hat{\delta}$ is approximated by solving

$$\log\left(\frac{f^*(\alpha\hat{\delta})}{f^*(0)}\right) = -1.25 \tag{3.21}$$

In practice, since we are only interested in the integral of one side, this is equivalent to minimizing

$$|\log(f^*(\alpha\hat{\delta})) - \log(f^*(0)) + 1.25|. \tag{3.22}$$

The value of α is determined in [49] to be $\sqrt{2.5}$. To see this, recall that $f^*(t^*)$ is proposed to be modeled by $g(t^*)$, thus $\log\left(f^*(\alpha\hat{\delta})/f^*(0)\right)$ can be modeled by

$$\log\left(\frac{g(\alpha\hat{\delta})}{g(0)}\right). \tag{3.23}$$

Substituting equation (3.19) into equation (3.23), we obtain

$$\begin{aligned}
&\log\left[K\left(1+\frac{(\alpha\hat{\delta})^2}{\hat{\nu}\hat{\delta}^2}\right)^{-(\hat{\nu}+1)/2}\right]-\log\left[K\left(1+\frac{(0)^2}{\hat{\nu}\hat{\delta}^2}\right)^{-(\hat{\nu}+1)/2}\right]\\
&=\log(1+\frac{(\alpha\hat{\delta})^2}{\hat{\nu}\hat{\delta}^2})^{-(\hat{\nu}+1)/2}\\
&=-\frac{(\hat{\nu}+1)}{2}\log(1+\frac{\alpha^2}{\hat{\nu}})=-1.25.
\end{aligned} \tag{3.24}$$

[49] pointed out that equation (3.24) when evaluated at $y=\sqrt{2.5}\,\delta$, i.e. $\alpha=\sqrt{2.5}$ is

$$-(\hat{\nu}+1)\log(1+2.5/\hat{\nu})=-1.25, \tag{3.25}$$

has less than 5% relative error for all $\hat{\nu}>0.6$. Therefore a good value of $\hat{\delta}$ can be approximated irrespective of $\hat{\nu}$ with $\alpha=\sqrt{2.5}$.

At this stage, the value of $\hat{\delta}$ is claimed be able to enable $g(t^*)$ to provide a good approximation to $f^*(t^*)$ at $t^*=\sqrt{2.5}\hat{\delta}$ for all $\hat{\nu}>0.6$ and at $t^*=0$. The determination of the value of $\hat{\nu}$ enables $g(t^*)$ to provide a good approximation at other t^* values. Likewise, the value of $\hat{\nu}$ is determined by solving

$$\begin{aligned}
\log\left(\frac{f^*(\hat{\delta})}{f^*(0)}\right)&=\log\left(\frac{g(\hat{\delta})}{g(0)}\right)\\
&=-(\hat{\nu}+1)\frac{\log(1+1/\hat{\nu})}{2},
\end{aligned} \tag{3.26}$$

where $\hat{\delta}$ was found from equation (3.22). In practice, solution of equation (3.26) is accomplished by minimizing

$$\left|\log\left(\frac{f^*(\hat{\delta})}{f^*(0)}\right)-(\hat{\nu}+1)\frac{\log(1+1/\hat{\nu})}{2}\right| \tag{3.27}$$

The approximation of the value of $\hat{\delta}$ and $\hat{\nu}$ completes the transformation of the skew hyperbolic Student-t distribution density function.

This approach is implemented in the `pskewhyp` function which is given in the Appendix.

3.3.2 Examples

In this section, we will illustrate the approximation result of the proposed approach by examining the CDF tail plots. Then the proposed approach is compared with the black-box approach

mentioned in Section 1.1.3. As in described in Section 3.2, the skew hyperbolic distribution has 4 parameters where β is the skewness parameter and ν is the scale parameter. When $\beta = 0$, the distribution is symmetric and when $|\beta|$ increases, the skewness of the distribution increases. When $\nu \to 0$, the distribution tends to have one extremely heavy tail. Those are the cases when the CDF approximation becomes extremely challenging. In the examples, the parameters $\delta = 1$ and $\mu = 0$ since these two parameters do not have an impact on the shape of the tails.

To illustrate the approach, we start with a CDF tail plot of a symmetric skew hyperbolic Student-t distribution when $\nu = 2, 5, 10$. The shape of the tails in Figure 3.1 are consistent with expectation, i.e. two exponential decaying tails when $\beta = 0$, indicating that the approach works well when the distribution of interest is symmetric.

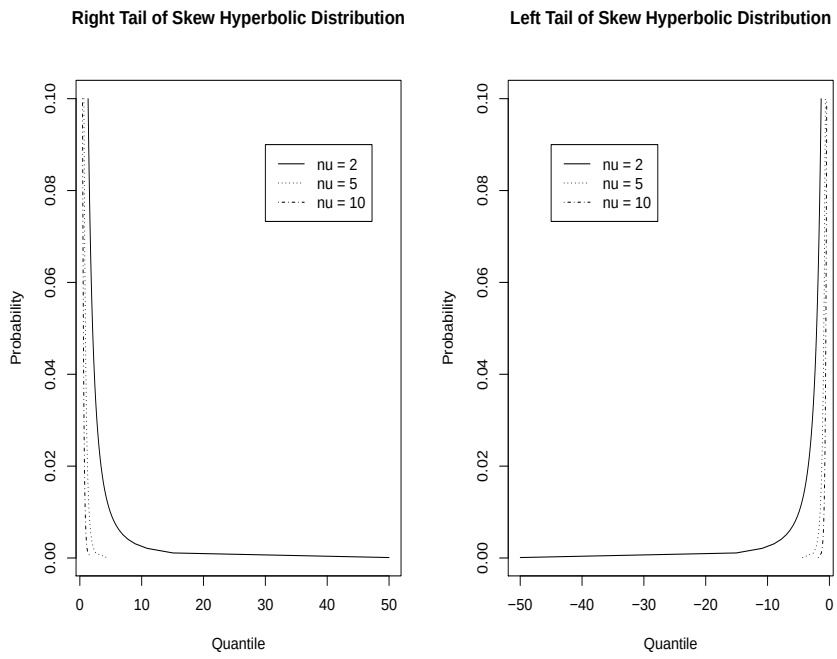

Figure 3.1: The tails of symmetric skew hyperbolic Student-t distribution CDF with different ν values

As the approach satisfactorily approximates a symmetric CDF, we then demonstrate the approach using some parameters which produce an extremely skewed distribution, $\beta = 5$ and $\nu = 1$ for instance. With these parameters the previous approach fails, even when $F(x) = 0.9$. We first use the CDF tail plot to examine the performance.

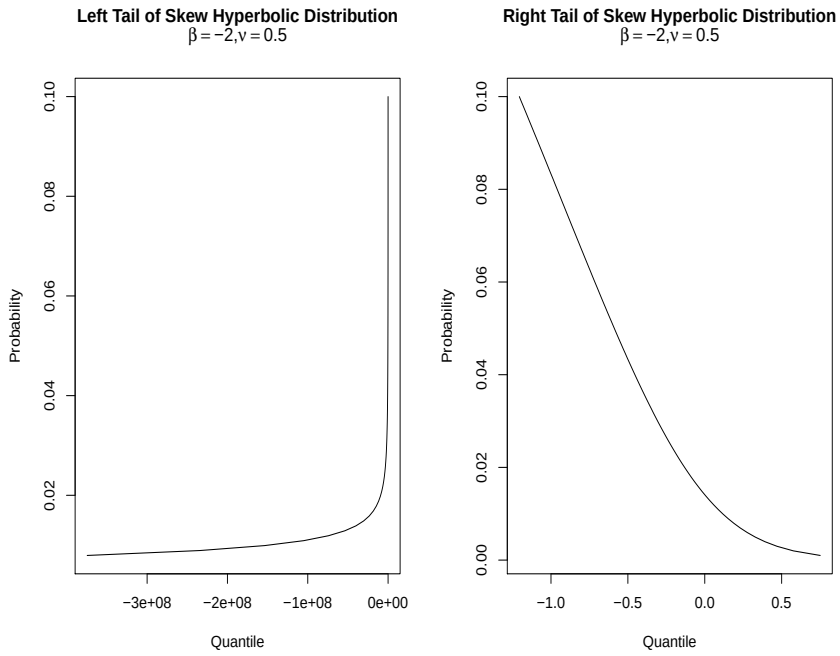

Figure 3.2: The tails of a non-symmetric skew hyperbolic Student-t distribution CDF with $\beta = -2$ and $\nu = 0.5$

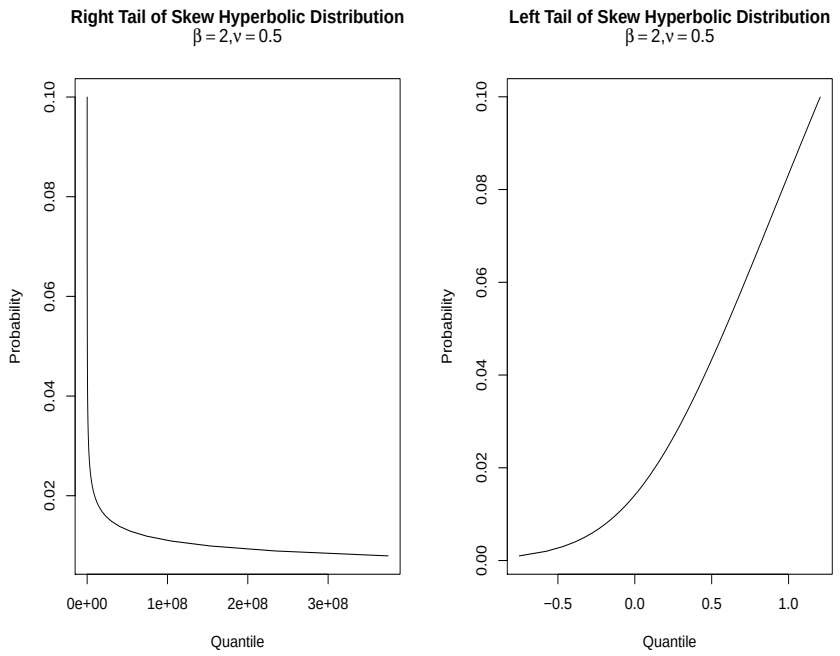

Figure 3.3: The tails of a non-symmetric skew hyperbolic Student-t distribution CDF with $\beta = 2$ and $\nu = 0.5$

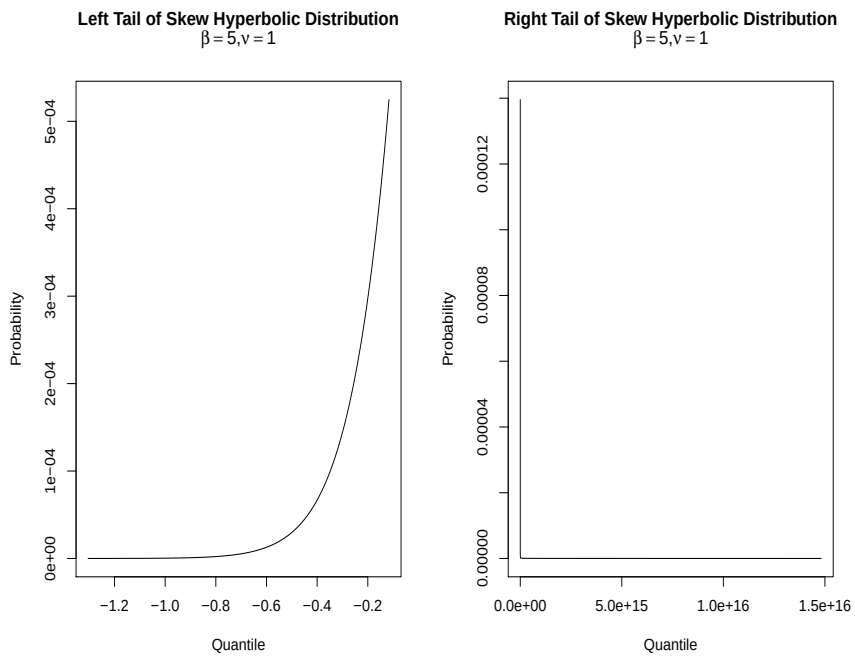

Figure 3.4: The tails of non-symmetric skew hyperbolic Student-t distribution CDF with $\beta = 5$ and $\nu = 1$

Figures 3.2, 3.3 and 3.4 shows that the approximated tails are consistent with expectation, i.e. exponential left tail and polynomial right tail when $\beta > 0$ or polynomial left tail and exponential right tail when $\beta < 0$, indicating that the approach works well when the distribution of interest is extremely skewed.

After examining the performance of the proposed approach, it is then compared with the black-box approach. We first use the `rskewhyp` function to generate 10000 random observations from the skew hyperbolic Student-t distribution with $\mu = 0$, $\delta = 1$, $\beta = -5$, $\nu = 1$.

```
n <- 10000
param <- c(0, 1, -5, 1)
x <- rskewhyp(n, param = param)
```

Then we approximate the CDF $F(x)$ using the proposed approach and the black-box approach respectively. The function `pinv.new` creates the `UNU.RAN` object as mentioned in Section 3.3.1.

```
pSplitT <- pskewhypGK(x, param = param, method = "integrate")
gen <- pinv.new(pdf = skewhyppdf, lb = -Inf, ub = Inf)
pRunuran <- up(gen, x)
```

The absolute difference is calculated as

$$|\Pr(X)_{\text{Runuran}} - \Pr(X)_{\text{SplitT}}| \tag{3.28}$$

and the absolute ratio is calculated as

$$\left|\frac{\Pr(X)_{\text{Runuran}}}{\Pr(X)_{\text{SplitT}}}\right| \tag{3.29}$$

Table 3.7: The absolute difference between the proposed approach and the black-box approach approximation of the skew hyperbolic Student-t distribution CDF when $\beta = -5$, $\nu = 1$

	Min.	1st Quantile	Median	Mean	3rd Quantile	Max.
Abs. Diff.	0.000	2.7035×10^{-8}	2.384×10^{-5}	1.438×10^{-5}	3.140×10^{-5}	6.736×10^{-5}
Abs. Ratio	0.9999	1.0000	1.0000	1.0000	1.0000	1.0000

Table 3.7 shows the summary statistics of absolute differences and the ratios of the probabilities calculated by these two approaches. The absolute differences of the two approaches are fairly small and the ratios are quite close to 1. It appears the split-t approach has improved the CDF approximation for extreme parameter values.

After the comparison under the extreme parameter but not-so-extreme probabilities, we will demonstrate the approach with more extreme probabilities i.e. $\Pr(X) \leq 0.1$ when $\beta < 0$ and $\Pr(X) \geq 0.9$ when $\beta > 0$, as the current approach performs poorly in this polynomial tail area. We still use some parameters which produce skewed distributions, namely $\nu = (0.5, 1)$ and $\beta = (-2, 2, 5)$. The quantiles used to approximate the CDF are calculated by the black-box approach. Sample code illustrates the procedure for obtaining the quantiles.

```
p <- c(0.9, 0.99, 0.999, 0.1, 0.01, 0.001)
gen <- pinv.new(pdf = skewhyppdf, lb = -Inf, ub = Inf)
xRunuran <- uq(gen, p)
```

We usse the absolute percentage error to measure the accuracy of the approach, calculated as

$$100 \times \left|\frac{\Pr(X)_{\text{ref}} - \Pr(X)_{\text{new}}}{\Pr(X)_{\text{ref}}}\right| \tag{3.30}$$

Table 3.8: The absolute percentage error between the proposed approach and the black-box approximation of the skew hyperbolic Student-*t* distribution CDF

β	ν	$\Pr(X)_{\text{ref}}$	X_{Runuran}	$\Pr(X)_{\text{SplitT}}$	Absolute Percentage Error (%)
2	0.5	0.9	14814.74	0.900000055	6.10×10^{-6}
2	0.5	0.99	1.481548×10^{8}	0.99	5.13×10^{-10}
-2	0.5	0.01	-1.481548×10^{8}	0.01	5.02×10^{-8}
-2	0.5	0.1	-1.481474×10^{4}	0.09999995	4.97×10^{-5}
5	1	0.99	3.182973×10^{4}	0.9904864	3.25×10^{-6}
5	1	0.999	3.183097×10^{6}	0.9989999998	1.64×10^{-8}
5	1	0.9999	3.183099×10^{8}	0.9999048	4.77×10^{-4}
-5	1	0.001	-3183097.25	0.001	1.08×10^{-5}
-5	1	0.01	3.1829.37	0.00999967	3.32×10^{-4}

The absolute percentage errors in Table 3.8 are fairly small. The largest percentage error is only $3.32 \times 10^{-4}\%$ and the tail can be precisely estimated as far as $x = 10^{8}$. The result indicates that this approach is fairly accurate and robust in extreme tail areas, which is a significant improvement on the current approach.

CHAPTER 4

RANDOM NUMBER GENERATION OF GENERALIZED HYPERBOLIC AND HYPERBOLIC DISTRIBUTION

The generation of pseudo-random variates which are distributed in accordance with a given continuous distribution is critical for investigating any continuous distribution including the GHyp distribution. In this chapter three major general methods for generation of univariate continuous distribution random variates proposed in the literature are discussed, namely,

- Inversion Sampling
- Envelope Rejection Sampling
- Slice Sampling.

Inversion sampling is a basic and simple method for generating random variates from a given distribution. Consider a distribution with strictly monotone CDF $F(x)$. If

$$X = F^{-1}(U) = \inf\{x : F(x) \geq U,\ 0 < u < 1\} \tag{4.1}$$

where U is on $(0,1)$ and $F^{-1}(x)$ is the quantile function, then X is an observation from the distribution given by the CDF $F(x)$.

This would appear to solve the problem in general, however it is often difficult to calculate the quantile function because the CDF either is not invertible or does not possesses a closed form. Hence development of other methods is necessary.

As mentioned in Chapter 1, one of the attractions of the `UNU.RAN` project (discussed in a later section) is to provide universal non-uniform random number generators using the numerical inversion approach. Although the generators perform well, the approach does require rather complicated machinery including creation and storage of a fairly large table. Therefore the search for simple and reliable stand-alone algorithms remains important.

The rejection sampling or so called acceptance-rejection algorithm is a Monte Carlo method which relies on repeated sampling. Let $f(x)$ be the density function of a continuous distribution for which the CDF $F(x)$ is difficult to invert. Assume $f(x)$ is bounded and non-zero on a finite interval $[\alpha,\beta]$ and $M = \max\{f(x) : \alpha \le x \le \beta\}$. To obtain $V \sim U[\alpha,\beta)$, simply set $V = \alpha + (\beta - \alpha)U$ where $U \sim U[0,1)$. Then the formal statement of the algorithm is

1. Generate $V_1 \sim U[\alpha,\beta)$
2. Generate $V_2 \sim U[0,M)$
3. Accept $X = V_1$ if $V_2 \le f(V_1)$ otherwise return to step 1

The chief drawbacks of this algorithm are

- The sampling range of $f(x)$ must be finite.
- The acceptance probability is $1/\left[M(\beta-\alpha)\right]$ which can be very inefficient.

To overcome these deficiencies, the use of an envelope or instrumental density $g(x)$ has been introduced. For this method, $g(x)$ must satisfy $f(x) < Mg(x)$ where $M > 1$. The algorithm becomes more efficient as $M \to 1$, thus the choice of $g(x)$ is crucial for this algorithm. There is an extensive literature proposing and discussing $g(x)$ that are relatively easy to compute and close enough to $f(x)$ $\forall x$ for distributions with different characteristics. Some examples are [50] for distributions with a log-concave density function, [3] for the GIG and GHyp and hyperbolic distributions (discussed in Section 4.1.1 and 4.2), [59] for T-concave density functions (discussed in Section 4.2.4), and [70] (discussed in Section 4.2.2) for continuous distributions.

The ratio-of-uniforms approach, proposed in [70], appears most useful for our work as the algorithms which utilize this technique are often short and their performance is comparable with more complicated algorithms. The approach is described as follows:

Definition 8. *Let X be a random variable with density function $f(x)$ with support (b_l, b_r) not necessarily finite. If (U, V) is uniformly distributed in $C_f = \{(u,v) : 0 \le u \le \sqrt{f(v/u+\mu)}, b_l < v/u + \mu < b_r\}$, then $X = V/U + \mu$ has the desired density function $f(x)$.*

To generate U, V uniformly over the region C_f, one must use the rejection method to generate U, V uniformly over the minimal enclosing rectangle D, defined as

$$D = \{(u,v) : 0 \le u \le u_+, v_- \le v \le v_+\} \tag{4.2}$$

where

$$\begin{aligned} u_+ &= \sup_x \sqrt{f(x)} \\ v_- &= \inf_x (x-\mu)\sqrt{f(x)} \\ v_+ &= \sup_x (x-\mu)\sqrt{f(x)}. \end{aligned} \tag{4.3}$$

It is shown in [104] that rejection from the minimal enclosing rectangle is equivalent to rejection from the envelope distribution

$$h(x) = \begin{cases} (v_-/x)^2 & \text{for } x < v_-/u_+ \\ (u_+)^2 & \text{for } v_-/u_+ \le x \le v_+/u_+ \\ (v_+/x)^2 & \text{for } x > v_+/u_+. \end{cases} \tag{4.4}$$

Apart from the two random variate generation techniques described above, slice sampling takes a simple form and can be applied to a wide variety of distributions. The fundamental idea of the slice sampling approach is if you can sample uniformly from the two-dimensional region under the probability density function $f(x)$, the horizontal coordinates of sample points will then have the desired density. However, it is often difficult to sample uniformly over the two-dimensional region, consequently, the latent variable Y is introduced. Let $f(x)$ be the probability density function of the required distribution. Given an observation x of the random variable X with density $f(x)$, the latent variable Y has distribution $Y|x \sim U(0, g(x))$ where $g(x) = cf(x)$. Set

$$\begin{aligned} f(x,y) &= \begin{cases} f(x)/g(x) & 0 \le y \le g(x) \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} 1/c & 0 \le y \le g(x) \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{4.5}$$

Since X and Y are uniformly distributed over the region $\{(x,y) : 0 \le y \le g(x)\}$, $f(x|y)$ must also be uniformly distributed. Then a sequence of (X_i, Y_i) is generated using the Gibb's sampling technique discussed in [35]. The ith iteration of the algorithm is

- Simulate $Y_i \sim f(y|x_{i-1}) = U(0, g(X_{i-1}))$
- Simulate $X_i \sim f(x|y_i) = U(S(y_i))$ where $S(y) = \{x : g(x) \ge y\}$.

The X_i are discarded and Y_i comprise a set of correlated variates from the desired distribution. Adjustments, mentioned in Section 4.2.3 and discussed in [106], can then be applied to obtain independent variates.

After this brief discussion of three general methods of random variate generation, in the later sections, specific approaches to generate random variates from generalized inverse Gaussian distribution which are critical for the GHyp distribution random variate generation are investigated. Thereafter approaches for the hyperbolic distribution, the sub-case of the GHyp distribution, are reviewed and investigated. Finally the performance of the methods is compared to decide on the best approach.

4.1 Generalized Inverse Gaussian Distribution

The generalized inverse Gaussian distribution GIG(λ, χ, ψ), defined in Section 3.2, possesses an important property for generating random variates from GHyp distribution. The GHyp

distribution, as mentioned in Section 3.2, is defined as a generalized inverse Gaussian distribution-variance-mean mixture of the normal distribution ([86]). Let $\chi = \delta^2$ and $\psi = \alpha^2 - \beta^2$, if $w \sim \mathrm{GIG}(\lambda, \delta^2, \alpha^2 - \beta^2)$, then the random variate with conditional normal distribution $Y \sim N(\mu + \beta w, w)$ is distributed as the GHyp distribution. The algorithms for normal distribution random variate generation are well-established, such as [71], [26] and [2], and are widely implemented in software. Besides the univariate GHyp distribution, the mixture property is valuable when generating random variates from the multivariate GHyp distribution. It is shown in [86], that the d-dimensional GHyp random variates $Y_i \sim \mathrm{GHyp}_d(\lambda, \alpha, \beta, \delta, \mu, \Delta)$ can be obtained as:

$$y_i = \sqrt{W_j}\left(X_{d(j-1)+1}, \ldots, X_{dj}\right) D + \mu + W_j \beta \Delta, \tag{4.6}$$

where for $\{1 \le j \le n\}$, $W_j \sim \mathrm{GIG}(\lambda, \delta^2, \alpha^2 - h_0' \Delta h_0)$ and for $\{1 \le i \le d_n\}$, X_i are distributed as standard normal. In addition, D is the Cholesky decomposition of the matrix Δ.

For these reasons we use the mixing property to first generate random variates W from the GIG distribution, then generate random variates from the normal distribution with parameters $\mu = \mu + \beta W$ and $\sigma = W$ to obtain random variates from the GHyp distribution. Therefore it is critical to generate random variates with the GIG distribution efficiently.

4.1.1 Atkinson Rejection Sampling Method

A. C. Atkinson in [3] proposed using the rejection sampling technique to generate random variates from the GIG distribution. The algorithm is named GIG and it is applicable for $\lambda \in \mathbb{R}$. This algorithm is described in Richard Trendall's honours project ([106]) in detail, and the brief description below is drawn from that.

Let the GIG density function $f(x) = ce(x; \lambda, \chi, \psi)$, where

$$e(x; \lambda, \chi, \psi) = x^{\lambda - 1} \exp\left[-\frac{1}{2}(\chi x^{-1} + \psi x)\right] \tag{4.7}$$

The algorithm GIG uses rejection with two part-envelopes divided at the mode m, given by

$$m(\lambda, \chi, \psi) = \begin{cases} [(\lambda - 1) + \sqrt{(1-\lambda)^2 + \chi\psi}]/\psi & \psi > 0 \\ \chi/[2(1-\lambda)] & \psi = 0. \end{cases} \tag{4.8}$$

The envelope, shown in Figure 4.1 is

$$g(x) = \begin{cases} k_1 d_1(x) & 0 \le x \le m \\ k_2 d_2(x) & x > m \end{cases} \tag{4.9}$$

where $d_1(x) = e^{sx}$, $d_2(x) = e^{-px}$ and $g(x)$ is required to be a density function, that is

$$\begin{aligned} \int_0^\infty g(x)\,\mathrm{d}x &= k_1 \int_0^t d_1(x)\,\mathrm{d}x + k_2 \int_t^\infty d_2(x)\,\mathrm{d}x \\ &= k_1 \Delta_1 + k_2 \Delta_2 = 1 \end{aligned} \tag{4.10}$$

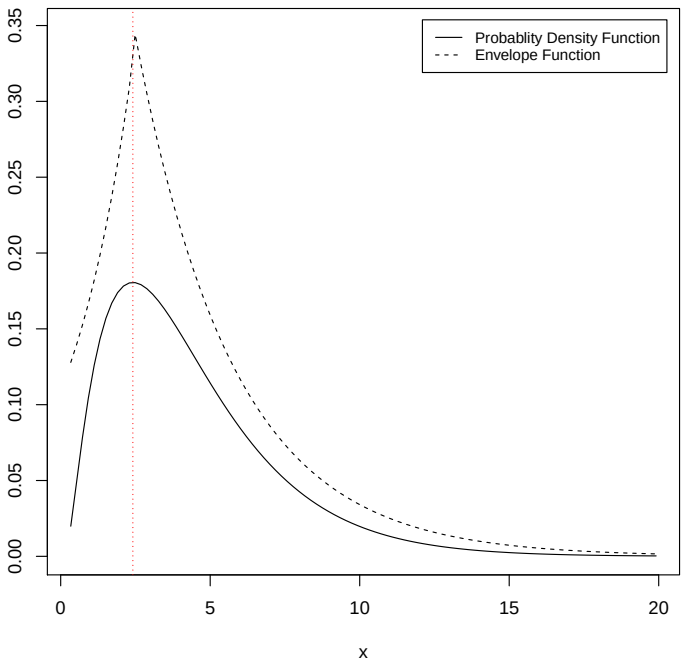

Figure 4.1: The Atkinson Two Part Rejection Envelope when $\beta = \omega = 1$, $\lambda = 2$

The rejection functions are proportional to the GIG density:

$$\begin{aligned} h_1(x) &= \frac{e(x;\lambda,\chi,\psi)}{d_1(x)} \\ &= e^{-sx}x^{\lambda-1}e^{-\frac{1}{2}(\chi x^{-1}+\psi x)} \\ &= x^{\lambda-1}e^{-\frac{1}{2}(\chi x^{-1}+(\psi+2s)x)} \\ &= e(x;\lambda,\chi,\psi+2s). \end{aligned} \tag{4.11}$$

Similarly $h_2(x) = e(x;\lambda,\chi,\psi-2p)$. The values of k_1 and k_2 are calculated by letting the ratios of the acceptance probability of any given x from both sides of the mode m proportional to the density $f(x)$ be the same, i.e.

$$\begin{aligned} k_1(x)d_1(x)\frac{h_1(x)}{S_1} &= k_2(x)d_2(x)\frac{h_2(x)}{S_2} \\ \frac{k_1}{S_1} &= \frac{k_2}{S_2} = F \end{aligned} \tag{4.12}$$

where S_i are the suprema of the rejection functions and F is the ratio of the acceptance probability to the density $f(x)$.

Let $k = k_1S_1 = k_2S_2$ then because $g(x)$ is a density function

$$k = \frac{S_1 S_2}{S_1\Delta_1 + S_2\Delta_2} \tag{4.13}$$

and the efficiency

$$\begin{aligned} E &= \frac{F}{c} \\ &= \frac{k_1}{cS_1} \\ &= \frac{1}{c(S_1\Delta_1 + S_2\Delta_2)}. \end{aligned} \tag{4.14}$$

Therefore to achieve maximum efficiency, it is essential to minimize the denominator $S_1\Delta_1 + S_2\Delta_2$. Although $S_1\Delta_1$ and $S_2\Delta_2$ can be minimized independently, Atkinson in ([3]) points out that these two functions can not be solved analytically. For this reason, the approach is potentially unstable when applied over an extensive parameter range.

The algorithm GIG is implemented in the function `rgigAtkin` which has the format

```
rgigAtkin(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(chi, psi, lambda)`.

Table 4.1 shows the acceptance probabilities of the Atkinson rejection method for various λ and β values. The acceptance probabilities are fairly reasonable apart from when λ and ω are both extremely small. As $\lambda \to 0$ and $\omega \to 0$ simultaneously, the acceptance probability declines substantially.

Table 4.1: Acceptance Probabilities of the GIG Algorithm

	λ			
$\beta = \omega$	0.01	0.1	1	10
0.01	0.0410	0.0455	1.0000	0.7634
0.1	0.2132	0.2114	0.9615	0.7299
1	0.6329	0.6369	0.9091	0.7042
10	0.8333	0.8264	0.8264	0.7874

4.1.2 The Ratio-of-Uniforms with Shifted Mode Method

The default random number generation function `rgig` for the GIG distribution in the **GeneralizedHyperbolic** package, as mentioned in Section 1.1.2, uses the ratio-of-uniforms with shifted mode method. It is based on the algorithm `GENINV` proposed by J. S. Dagpunar ([33]). Dagpunar first rewrites the probability density function of the scaled relocated variate z in the alternative parameterization $\alpha = 1/\eta$, $\beta = \omega$ as

$$f_z(z) = \frac{(m+z)^{\lambda-1} e^{-0.5\beta(z+m+1/(z+m))}}{2K_\lambda(\beta)} \tag{4.15}$$

where $\lambda \geq 0$, $\beta > 0$, $z = \eta x - m$ and m is the mode of ηx which is given by

$$m = \frac{\lambda - 1 + \sqrt{(\lambda-1)^2 + \beta^2}}{\beta}. \tag{4.16}$$

The algorithm then works with the quasi-density function

$$h_z(z) = (m+z)^{\lambda-1} e^{-0.5\beta(z+m+1/(z+m))}. \tag{4.17}$$

A random variate $Z = z$ is generated by the ratio-of-uniforms method. Let $y = \eta x$. The bounds of minimal enclosing rectangle are proven to be $u_+ = \sqrt{h(0)}$, $v_- = \min((y-m)\sqrt{h(y-m)})$ and $v_+ = \max((y-m)\sqrt{h(y-m)})$. The values of v_+ and v_- are found by setting the derivative of $(y-m)\sqrt{h(y-m)}$ with respect to y equal to 0. It was pointed out in [33] that this is equivalent to solving the following equation in y:

$$\frac{1}{2}\beta y^3 - y^2\left(\frac{1}{2}\beta m + \lambda + 1\right) + x\left(|\lambda - 1| m - \frac{1}{2}\beta\right) + \frac{1}{2}\beta m = 0. \tag{4.18}$$

The real roots y_- and y_+ define v_- and v_+ according to

$$\begin{aligned} v_- &= (y_- - m)\sqrt{h(y_- - m)} \\ v_+ &= (y_+ - m)\sqrt{h(y_+ - m)}. \end{aligned} \tag{4.19}$$

To obtain y_- and y_+ the R function `uniroot` can be used.

To draw samples from the GIG distribution, we first generate two independent uniformly distributed random variates R_1 and R_2 over $[0,1]$, then set

$$\begin{aligned} V &= R_2(v_+ - v_-) + v_- \\ U &= R_1 u_+. \end{aligned} \tag{4.20}$$

Since $Z = V/U$, X can be generated as

$$\begin{aligned} X &= \frac{U}{V} + m \\ &= \frac{R_2(v_+ - v_-) + v_-}{R_1 u_+}. \end{aligned} \tag{4.21}$$

The prospective X is accepted if satisfies the condition:

$$R_1 u_+ \leq \sqrt{h(Z)}, \tag{4.22}$$

and the acceptance probability is

$$P = \frac{0.5 \int_0^\infty h(z)\mathrm{d}z}{u_+(v_+ - v_-)}. \tag{4.23}$$

The `rgig` function which implements this algorithm has the format

```
rgig(n, chi = 1, psi = 1, lambda = 1,
     param = c(chi, psi, lambda))
```

`n`

Number of random variates to be generated.

`chi`

χ, The parameter of GIG distribution that by default holds a value of 1.

`psi`

ψ, The parameter of GIG distribution that by default holds a value of 1.

`lambda`

λ, The parameter of GIG distribution that by default holds a value of 1.

`param`

Parameter vector taking the form `c(chi, psi, lambda)`.

Table 4.2 shows acceptance probabilities of ratio of uniforms with shifted mode approach. The acceptance probabilities are at satisfactory level apart from when λ and β are both extremely small. Although not as extreme as for Atkinson's approach, as $\lambda \to 0$ and $\beta \to 0$ simultaneously, the acceptance probability declines fairly significantly.

Table 4.2: Acceptance Probabilities of the GENINV Method

	λ			
β	0.01	0.1	1	10
0.01	0.3891	0.3049	0.6452	0.7407
0.1	0.5319	0.5263	0.7576	0.7634
1	0.6667	0.7092	0.7092	0.7752
10	0.7299	0.7194	0.7143	0.6757

4.1.3 Dagpunar Rejection Sampling Method

As mentioned in Section 4.1.2 and 4.1.1, the acceptance probability declines rapidly as $\lambda \to 0$ and $\beta \to 0$ simultaneously. Dagpunar in 2007 proposed a rejection sampling method using a gamma distributed hat function which provides a remedy for this issue. This algorithm is applied to the scaled density function. Let $X \sim \text{GIG}(\alpha, \beta, \lambda)$ then define $X^2 = \alpha X$, the scaled pdf of x is

$$h(x) = x^{\lambda-1} e^{-\beta(x+1/x)/2}. \tag{4.24}$$

The envelope is $g(x) = Kr(x)$ as shown in Figure 4.2 when $\beta = 0.8$ and $\lambda = 1$.

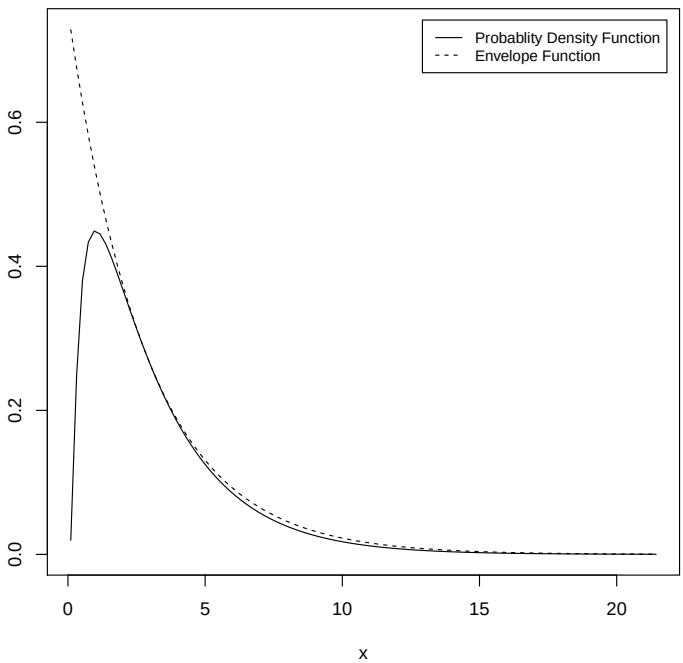

Figure 4.2: The Gamma Distributed Rejection Envelope when $\beta = 0.8$, $\lambda = 1$

The function $r(x)$ is given by ([34]) as

$$r(x) = x^{\lambda-1}e^{-\frac{\gamma x}{2}}, \tag{4.25}$$

and

$$K = \max\left(\frac{h(x)}{r(x)}\right) = e^{-\sqrt{\beta(\beta-\gamma)}}. \tag{4.26}$$

To maximize the acceptance probability

$$\gamma = \frac{2\lambda^2(\sqrt{1+\frac{\beta^2}{\lambda^2}}-1)}{\beta} \tag{4.27}$$

giving the acceptance probability

$$\frac{\int_0^\infty h(x)\mathrm{d}x}{\int_0^\infty g(x)\mathrm{d}x} = \frac{e^{\sqrt{\beta(\beta-\gamma)}}(\gamma/2)^\lambda \int_0^\infty x^{\lambda-1}e^{-\beta(x+1/x)/2}\mathrm{d}x}{\Gamma(\lambda)} \tag{4.28}$$

This algorithm is implemented in the function `rgigGamma` which has the format

```
rgigGamma(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(chi, psi, lambda)`.

Figure 4.3 presents a histogram of random variates from the GIG distribution with $\beta = 0.1$, $\lambda = 0.5$ generated by `rgigGamma`. The density curve is superimposed.

Table 4.3 shows acceptance probabilities for the Dagpunar rejection sampling approach. The acceptance probabilities are fairly reasonable when $\lambda \geq 1$. However as $\lambda \to 0$ or $(\beta - \lambda) \to \infty$, the acceptance probability declines substantially.

Table 4.3: Acceptance Probabilities of the Gamma Hat Function Method

	λ			
β	0.01	0.1	1	10
0.01	0.0530	0.3984	0.9853	1.0000
0.1	0.0340	0.2714	0.9372	1.0000
1	0.0200	0.1714	0.7599	0.9950
10	0.0128	0.1046	0.5155	0.9747

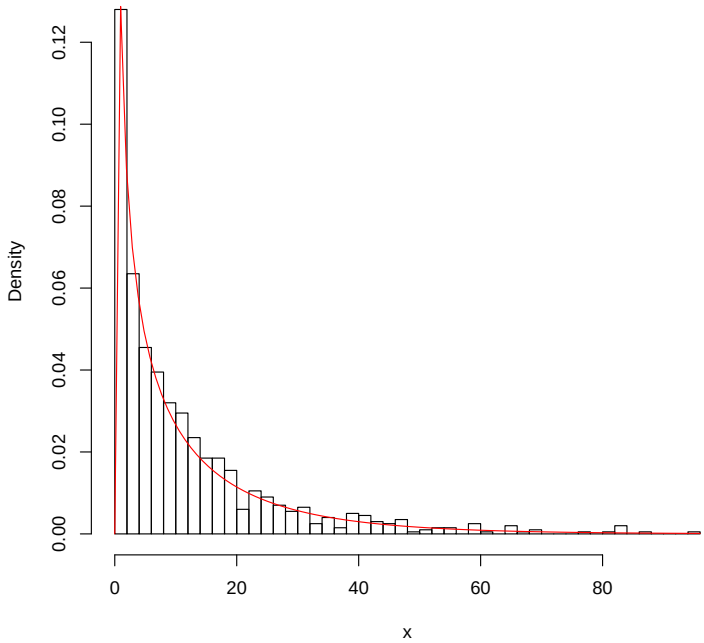

Figure 4.3: Histogram of random variates from the GIG distribution generated by Dagpunar's envelope

4.1.4 Hörmann's Rejection Sampling Method

Wolfgang Hörmann also addressed the concern mentioned in Section 4.1.2 in a conference talk and proposed some remedies ([60]). One of these, presented in [79], is the use of a rejection method with a universally bounded three-part hat function for $0 \le \lambda < 1$ and $\beta \le \min(1/2, 2/(3\sqrt{1-\lambda}))$.

Consider $X \sim \mathrm{GIG}(\lambda, \delta^2, \alpha^2 - \beta^2)$ which has the quasi-density function $f(x)$. The envelope $h(x)$, shown in Figure 4.4, has the form ([79])

$$h(x) = \begin{cases} f(m) & \text{if } x \le x_0 \\ x^{\lambda-1} e^{-\beta} & \text{if } x_0 < x \le x_* \\ {x_*}^{\lambda-1} e^{-x\beta/2} & \text{if } x > x_* \end{cases} \tag{4.29}$$

where $x_0 = \beta/(1-\lambda)$, $x_* = \max(x_0, 2/\beta)$. Note that there is inconsistency of the envelope function $h(x)$ as proposed in [79] and in [60]. Our implementation uses $h(x)$ as in [79].

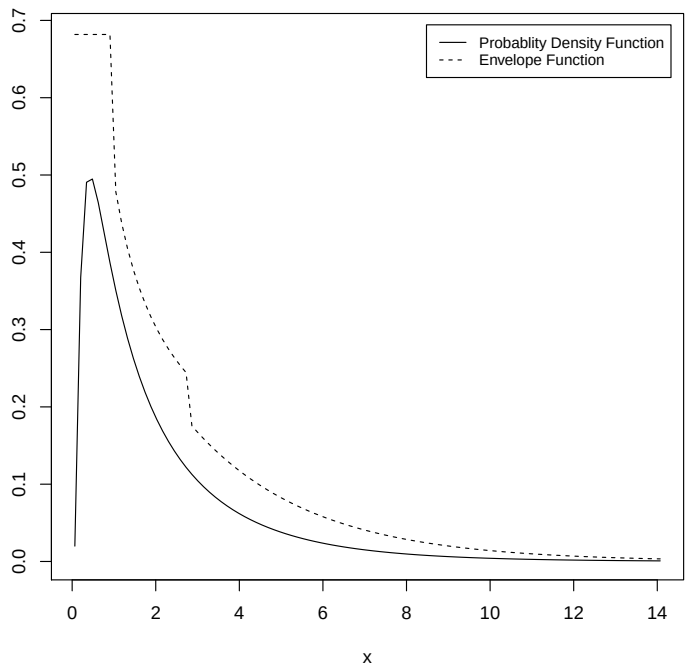

Figure 4.4: The Hörmann Rejection Envelope when $\beta = 0.5$, $\lambda = 0.3$

Areas under the three-part envelope are

$$
\begin{aligned}
\Delta_1 &= \int_0^{x_0} f(m)\,\mathrm{d}x \\
&= f(m)\frac{\beta}{1-\lambda} \\
\Delta_2 &= \int_{x_0}^{x_*} \frac{x^{\lambda-1}}{\lambda} e^{-\beta}\,\mathrm{d}x \\
&= \frac{{x_*}^{\lambda} - (\frac{\beta}{1-\lambda})^{\lambda}}{\lambda} e^{-\beta} \\
\Delta_3 &= \int_{x_*}^{-\infty} {x_*}^{\lambda-1} e^{\beta/2x}\,\mathrm{d}x \\
&= \frac{\beta}{2} {x_*}^{\lambda-1} e^{-\beta/2-x_*}
\end{aligned}
\tag{4.30}
$$

The suprema of the rejection functions are

$$
\begin{aligned}
s_1 &= f(x_0) \\
s_2 &= x^{\lambda-1} e^{-1-\beta^2/4} \\
s_3 &= f(x_*) e^{((\lambda-1)/x_* - \beta/2 + \beta/2x_*^2)(x-x_*)}
\end{aligned}
\tag{4.31}
$$

Therefore the rejection constants, claimed to be bounded by e, are

$$
\begin{aligned}
\rho_1 &= \frac{\Delta_1}{\int_0^{x_0} s_1(x)\,\mathrm{d}x} \\
&= \frac{f(m)x_0}{f(x_0)(x_0-m)} \\
&= \frac{f(m)}{f(x_0)}\frac{1}{1-m/x_0} \\
&< 2.72604, \\
\rho_2 &= \frac{\Delta_2}{\int_{x_0}^{2/\beta} s_2(x)\,\mathrm{d}x} \\
&= \frac{e^{-\beta}}{e^{-1-\beta^2/4}} \\
&= e^{1-\beta+\beta^2/4} \\
&< e, \\
\rho_3 &= \frac{\Delta_3}{\int_{x_*}^{\infty} s_3(x)\,\mathrm{d}x} \\
&= \frac{x_*^{\lambda-1}(2/\beta)e^{-\beta x_*/2}}{-[(\lambda-1)1/x_* - \beta/2 + \beta/(2x_*^2)]^{-1} x_*^{\lambda-1} e^{-\beta/2(x_*+1/x_*)}} \\
&= \left[(1-\lambda)\frac{2}{\beta x_*} - \frac{1}{x_*^2} + 1\right] e^{\beta/(2x_*)} \\
&< 2e^{1/16} \\
&< 2.129 \qquad (4.32)
\end{aligned}
$$

This algorithm is implemented in the function `rgigHoer` which has the format

```
rgigHoer(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(chi, psi, lambda)`.

Figure 4.5 presents a histogram of random variates from the GIG distribution with $\beta = 0.1$, $\lambda = 0.5$ generated by `rgigHoer`. The density curve is superimposed.

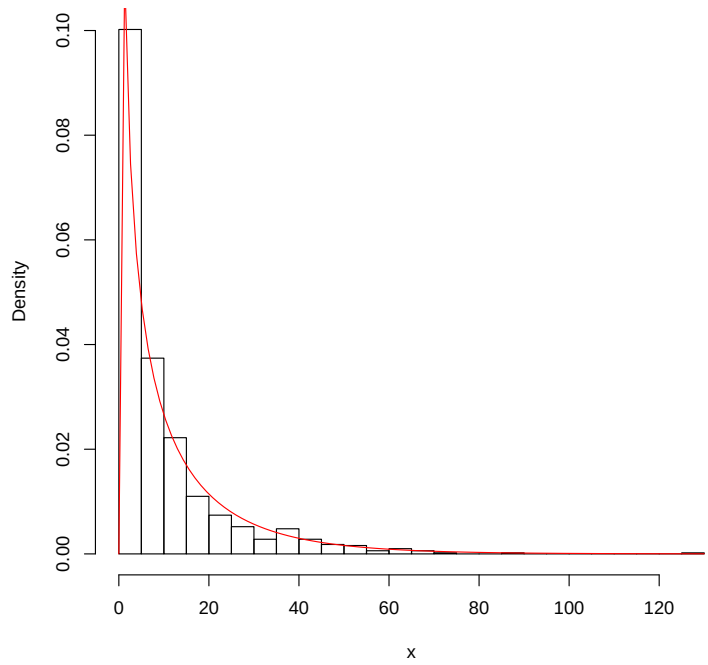

Figure 4.5: Histogram of random variates from the GIG distribution generated by the Hörmann envelope

Table 4.4 shows the acceptance probabilities of Hörmann's rejection sampling approach. The acceptance probabilities are at a satisfactory level throughout the table.

Table 4.4: Acceptance Probabilities of Hörmann's Rejection Sampling Method

	λ			
β	0.1	0.2	0.5	0.8
0.001	0.8264	0.7463	0.6897	0.7246
0.005	0.885	0.7813	0.7194	0.7692
0.01	0.7813	0.8621	0.6849	0.7937
0.05	0.8000	0.8475	0.8065	0.7299

4.1.5 Fast Inversion Method

As mentioned in Chapter 1 and earlier in this Chapter, the `UNU.RAN` project provides an approach for fast numerical inversion and can therefore generate random variates using the approximated quantile function. This 'black-box' algorithm, proposed in ([78]), is implemented in the R package **Runuran** but the important code is written in the Ch language to achieve optimal performance.

The algorithm consists of two steps, set-up and sampling. In the set-up step, $F(X)$ is evaluated at several points x_i then interpolation at the nodes $(u_i = F(x_i), x_i)$ is used to create a setup table to approximate the quantile function $F_a^{-1}(u)$ for given u in the sampling part, by means of polynomials. The error of the approximate quantile function $F_a^{-1}(u)$ is defined as $|u - F(F_a^{-1}(u))|$ ([38]). The criterion for the size of the approximation errors in this algorithm is aimed to be

$$\varepsilon_u(u) \geq \sup_{u\in(0,1)} |u - F(F_a^{-1}(u))|. \tag{4.33}$$

Since the set-up is the core part of this algorithm, it is worthwhile to give a description of methods used in this step. Given X with CDF $F(x)$ and probability density function $f(x)$, the set-up procedure has the following four steps. For a detailed explanation and full proof refer to [38].

1. Find the computationally relevant domain $[b_l, b_r]$ of the distribution. The algorithm is only applicable to bounded continuous distributions and it is claimed to be numerically unstable if the quantile function is very steep. Therefore it is essential to find appropriate cut-off points that are close enough to 0 and 1 but not too close to cause numerical problems. The cut off points are chosen at probability of $0.05\varepsilon_u$ for both tails. The quantiles of those points are approximated by a recursive method using formula as below ([38]):

$$p^* = \begin{cases} p + \frac{f(p)}{cf'(p)}\left[\left(\frac{0.05\varepsilon_u|f'(p)|(1+c)}{f(p)^2}\right)^{c/(1+c)} - 1\right] & \text{if } c \neq 0 \\ p + \frac{f(p)}{f'(p)}\log\frac{\varepsilon|f'(p)|}{f(p)^2} & \text{if } c = 0, \end{cases} \tag{4.34}$$

   where $c = n$ if $\mathrm{sgn}(n)f(x)^n$ is concave and $c = 0$ when $f(x)$ is log concave ([59]).

2. Divide the relevant domain $[b_l, b_r]$ into k intervals $[a_{i-1}, a_i]$ where $i = 1, \ldots k$. These intervals are then further divided by n construction points for the interpolation in step 4. The length of $[a_{i-1}, a_i]$ is decided by computing the interpolating polynomial and estimating the error on some tentative intervals, shortening if the error is larger than the required error $0.9\varepsilon_u$ and extending if the error is smaller than the required error $0.9\varepsilon_u$.

3. Compute $\int_{x_{j-1}}^{x_j} f(x)\mathrm{d}x$ for $j = 1, \ldots, n$ on each interval $[a_{i-1}, a_i]$ to approximate the CDF $F(x)$ using (adaptive) Gauss-Lobatto quadrature.

4. Interpolate the construction points $(F(x_j), x_j)$ where $a_{i-1} = x_0 < x_1 <, \ldots, < x_n = a_i$ to approximate the inverse CDF F^{-1} on each interval $[F(a_i - 1), F(a_i)]$ which is obtained from step 2. This is accomplished by using Newton's recursion for the interpolating polynomial with a fixed number of points. This interpolation method is selected for its accuracy and robustness.

After setting up the table of interpolation points, the sampling part is relatively straightforward. Indexed search is chosen to find the correct interval together with evaluation of the interpolation

polynomial ([38]). Rejection constant and acceptance probability are not relevant concepts for this algorithm.

In the case of the GIG distribution, the implementation of the method in R uses three functions, namely `udgig`, `pinvd.new` and `ur`. The purpose of these three functions is to create a `UNU.RAN` object for the GIG distribution which approximates the inverse of the CDF of the distribution by means of Newton interpolation, to allow the generation of random variates.

The function `udgig` has the format

```
udgig(theta, psi, chi, lb = 0, ub = Inf)
```

`theta`

θ The parameter of generalized inverse Gaussian distribution.

`psi`

ψ The parameter of generalized inverse Gaussian distribution.

`chi`

χ The parameter of generalized inverse Gaussian distribution.

`lb`

The lower bound of the (truncated) distribution.

`ub`

The upper bound of the (truncated) distribution.

The function `pinvd.new` has the format

```
pinv.new(pdf, cdf, lb, ub, islog = FALSE, center = 0,
         uresolution = 1.e-10, smooth = FALSE, ...)
```

`pdf`

The probability density function of required distribution.

`cdf`

The CDF of the required distribution.

`lb`

The lower bound of the required distribution domain.

`ub`

The upper bound of the required distribution domain.

`islog`

If set to `TRUE`, the density function or CDF are given by their corresponding logarithms.

`center`

Typical point of the required distribution.

`uresolution`

The maximal acceptable u-error.

`smooth`

If the inverse CDF is differentiable.

`...`

Optional arguments for `pdf` and `cdf`.

The function `ur` has the format

```
ur(unr, n)
```

`unr`

The `UNU.RAN` object.

`n`

The required sample size.

Figure 4.6 presents a histogram of random variates from the GIG distribution with $\beta = 0.1$, $\lambda = 0.5$ generated by the `udgig`, `pinvd.new` and `ur` functions together. The density curve is superimposed.

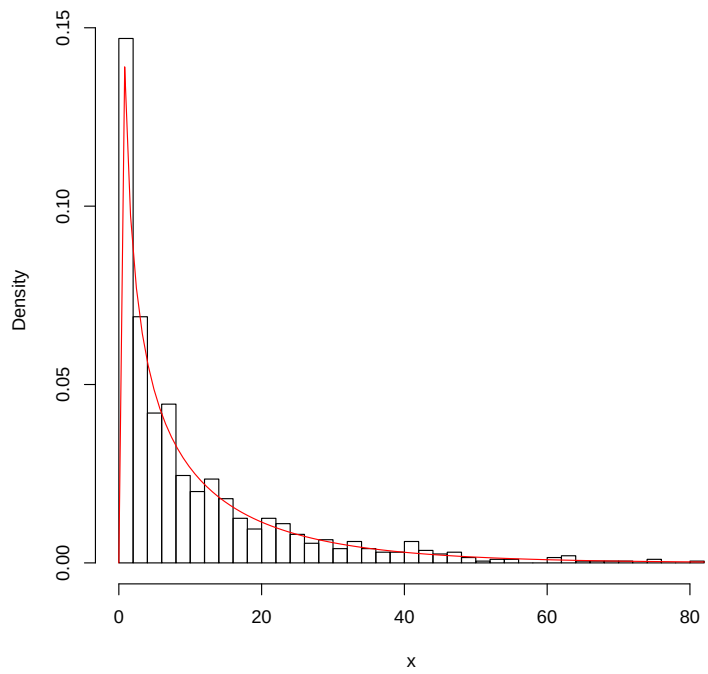

Figure 4.6: Histogram of random variates from the GIG distribution generated by the black-box algorithm

4.1.6 Other Methods

Apart from the above successfully implemented algorithms, two other approaches were investigated. The first approach is the slice sampler. The basic theory of this approach was described earlier in this chapter, and it is applied to the hyperbolic distribution in Section **??** following.

However in order to apply the slice sampler to the GIG distribution, we need to compute the bounds on the two uniform densities generating $f(y|x_{i-1})$ and $f(x|y_i)$ where Y is the latent variable with conditional distribution,

$$Y|x_{i-1} \sim U\left(0, x^{\lambda-1}e^{-\frac{1}{2}(\chi x^{-1}+\psi x)}\right), \tag{4.35}$$

where $x^{\lambda-1}e^{-\frac{1}{2}(\chi x^{-1}+\psi x)}$ is the quasi-density of the GIG distribution.

The bounds on $f(x|y_i)$ are the two roots x that satisfy the equation:

$$\begin{aligned} y_i &= x^{\lambda-1}e^{-\frac{1}{2}(\chi x^{-1}+\psi x)} \\ \Rightarrow \qquad \log(y_i) &= (\lambda-1)\log(x) - \frac{1}{2}(\chi x^{-1}+\psi x). \end{aligned} \tag{4.36}$$

This equation can only be solved explicitly for $\lambda = 1$. Since efficiency is critical for random number generation, the slice sampler is not appropriate for the entire parameter range of GIG distribution.

The second approach investigated is the transformed density rejection method proposed in [59]. This approach is also applied to the hyperbolic distribution in a later section.

A description of the method is given in Section 4.2.4. The transformed density rejection approach requires the distribution to be T-concave i.e $T(f(x))$ is concave. However the GIG distribution is not T-concave in the entire parameter domain. Consider $X \sim \text{GIG}(\lambda, \delta^2, \alpha^2 - \beta^2)$ which has the quasi-density function

$$f(x) = x^{\lambda-1} e^{-\beta/2(x^{-1}+x)}. \tag{4.37}$$

[59] has pointed out that $f(x)$ is concave with transformation $T(f(x)) = \log(f(x))$ if $\lambda \geq 1, \beta > 0$ and $T(f(x)) = \sqrt{f(x)}$ if $\lambda > 0, \beta \geq 0.5$, i.e.

$$T(f(x)) = \begin{cases} -\beta/2(x^{-1}+x) + (\lambda-1)\log(x) & (\lambda \geq 1, \beta > 0) \\ \sqrt{x^{\lambda-1} e^{-\beta/2(x^{-1}+x)}} & (\lambda > 0, \beta \geq 0.5), \end{cases} \tag{4.38}$$

The parameter domain that has not been covered by this approach is the area where the low acceptance probability occurs, therefore this approach may not appropriate to use.

4.2 Hyperbolic Distribution

In 2004, Richard Trendall ([106]) investigated methods for random number generation from the hyperbolic distribution. There were four approaches identified in his work. A brief description is given in this section for each method and an alternative method proposed in [59] is also investigated.

4.2.1 Atkinson Rejection Sampling Method

A. C. Atkinson ([3]) proposed the algorithm HYP which uses rejection sampling to generate random variates from the hyperbolic distribution. This algorithm uses a three-part envelope with the uniform distribution in the center and exponential distributions in the tails. The algorithm does not require optimization. The cut points separating the regions are at $t = -\sqrt{\gamma/\phi}$ and $w = \sqrt{\phi/\gamma}$, and the envelope, shown in Figure 4.7, is

$$g(x) = \begin{cases} ke^{\phi x} & (x \leq t) \\ ke^{\theta} & (t < x \leq w) \\ ke^{-\gamma x} & (x > w), \end{cases} \tag{4.39}$$

where $\theta = f(x_{\text{mode}})$ and $f(x)$ is the quasi-density function of the hyperbolic distribution.

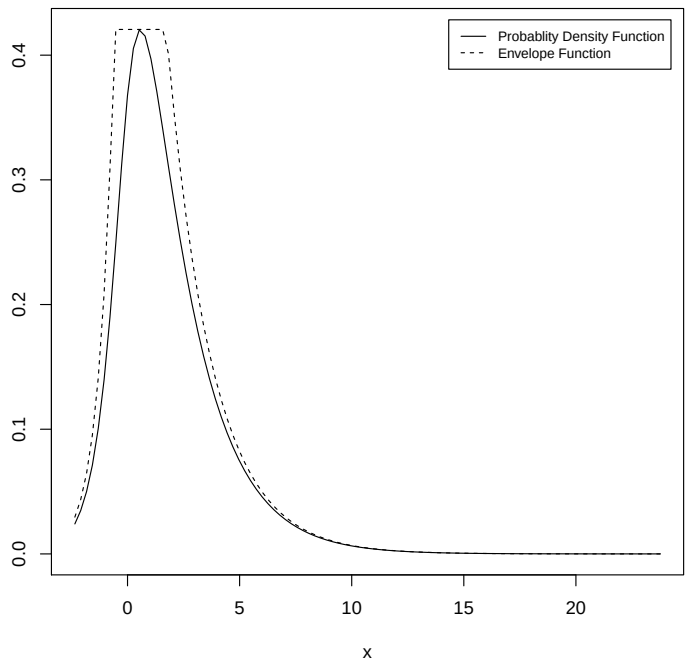

Figure 4.7: The Atkinson Rejection Envelope for the Hyperbolic Distribution when $\alpha = 1$, $\beta = 0.5$

The function $g(x)$ must be a density function and

$$\Delta_1 = \frac{e^{\phi t}}{\phi}, \quad \Delta_2 = (w-t)e^{\theta}, \quad \Delta_3 = \frac{e^{-\gamma w}}{\gamma}. \tag{4.40}$$

Therefore

$$\begin{aligned} 1 &= \int_{-\infty}^{\infty} g(x) \\ &= k\Delta_1 + k\Delta_2 + k\Delta_3 \\ \Longrightarrow k &= \frac{1}{e^{\theta}(1/\theta + w - t) + (1/\gamma)e^{-\gamma w}}. \end{aligned} \tag{4.41}$$

The algorithm is implemented in the function `rhypAtkin` which has the format

```
rhypAtkin(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta)`.

Table 4.5 shows the acceptance probabilities of the Atkinson rejection approach for various α and β values. The probabilities are in general at a satisfactory level but as α increases, the acceptance probability declines slightly.

Table 4.5: Acceptance Probabilities of the HYP Algorithm

	α			
β	0.5	1	2	5
0	0.9615	0.8850	0.6494	0.495
0.2α	0.9174	0.8475	0.6757	0.4651
0.8α	0.9524	0.8403	0.7463	0.5988

4.2.2 Ratio of Uniforms Method

The basic theory of the ratio of uniforms sampling technique was discussed earlier in this chapter. To apply the theory to the hyperbolic distribution, suppose $X \sim \text{hyp}(\mu, \delta, \alpha, \beta$ has the quasi-density function when $\mu = 0$ and $\delta = 1$,

$$g(x) = e^{-\alpha\sqrt{1+x^2}+\beta x}. \tag{4.42}$$

The bounds of the minimal enclosing rectangle are $v_- = 0$, $v_+ = \sqrt{g(x_+)}$, $u_\pm = y_\pm\sqrt{g(y_\pm)}$ where x_+ is the root of equation

$$-\alpha x/\sqrt{1+x^2} + \beta = 0. \tag{4.43}$$

The $y_\pm$ are the roots of the equation

$$\frac{e^{-\alpha\sqrt{1+y^2}+\beta y/2}(2\sqrt{1+x^2} + \alpha x^2 - x\beta\sqrt{1+x^2})}{2\sqrt{1+x^2}} = 0. \tag{4.44}$$

Figure 4.8 illustrates the equivalent envelope function of the minimal enclosing rectangle.

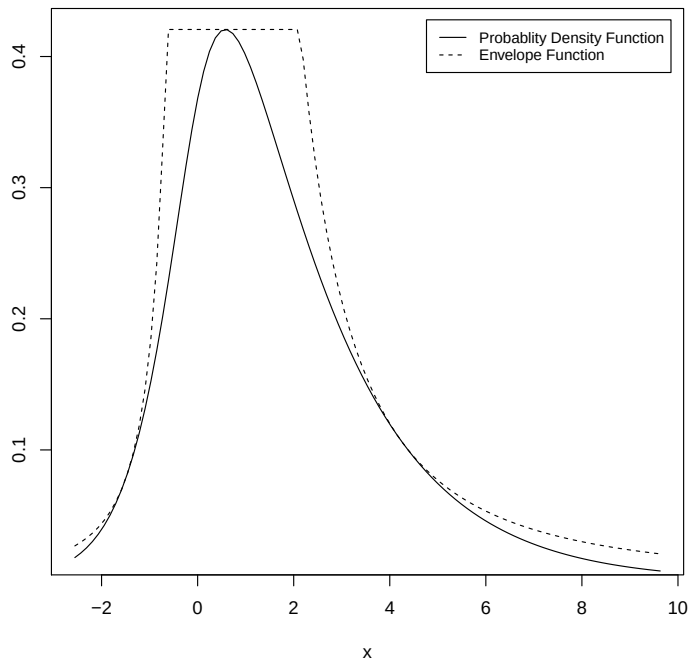

Figure 4.8: The Ratio-of-Uniforms Equivalent Rejection Envelope for the Hyperbolic Distribution when $\alpha = 1$, $\beta = 0.5$

The algorithm is implemented in the function `rhypRoU` which has the format

```
rhypRoU(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta)`.

Table 4.6 shows the acceptance probabilities for ratio of uniforms approach. The probabilities are in general at a satisfactory level and there does not appear any substantial decline as α increases.

Table 4.6: Acceptance Probabilities of the Ratio of Uniforms Method

	α			
β	0.5	1	2	5
0	0.7463	0.7042	0.6944	0.7519
0.2α	0.7752	0.7634	0.7692	0.7752
0.8α	0.7463	0.7092	0.7353	0.6711

4.2.3 Slice Sampler Method

The slice sampler method is also described in Richard Trendall's honours project. Therefore only a brief description is given below and for a detailed discussion see [106].

Let $f(x)$ be the probability density function of the hyperbolic distribution with the second parameterization $(\mu, \delta, \zeta, \pi)$. Define the latent variable Y such that $Y|x \sim U(0, g(x))$ where $g(x) = cf(x) = e^{-\zeta(\sqrt{1+\pi^2}\sqrt{1+x^2}-\pi)}$. The algorithm uses the Gibb's sampling technique to recursively simulate

$$Y_i \sim f(y|X_i = x_{i-1}) = U(0, e^{-\zeta\sqrt{1+\pi^2}\sqrt{1+x_{i-1}^2}-\pi x_{i-1}}) \tag{4.45}$$

and

$$\begin{aligned} X_i &\sim f(x|Y_i = y_i) \\ &= U\Bigg(-\frac{\pi}{\zeta}\log(y_i) - \frac{1}{2}\sqrt{\left[\frac{2\pi}{\zeta}\log(y_i)\right]^2 - 4\left[1+\pi^2-\left(\frac{1}{\zeta}\right)^2\log(y_i)^2\right]}, \\ &\quad -\frac{\pi}{\zeta}\log(y_i) + \frac{1}{2}\sqrt{\left[\frac{2\pi}{\zeta}\log(y_i)\right]^2 - 4\left[1+\pi^2-\left(\frac{1}{\zeta}\right)^2\log(y_i)^2\right]}\Bigg). \end{aligned} \tag{4.46}$$

The X_i are then correlated variates from the density $f(x)$. The start value for the X_i can be any arbitrary value. Although the approach appears to be relatively simple, a "burn-in" period is required and in addition, only n^{th} variates are taken to treat the serial correlation issue. These two requirements increase the time to generate random variates.

The algorithm is implemented in the function `rhypSS` and has the format

```
rhypSS(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta)`.

4.2.4 Transformation Density Rejection Method

The transformed density rejection (TDR) method was proposed in [59]. The basic idea of the TDR algorithm is fairly intuitive i.e., given a suitable transformation $T(x)$ of the desired density function $f(x)$, if $l(x)$ is a piecewise linear function where $l(x) \geq T(f(x))\,\forall x$, then $T^{-1}(l(x))$ must be a dominating function for $f(x)$. The approach can also incorporate a squeeze function $s(x)$, which is an easy-to-compute lower bound of $f(x)$ used for a preliminary test of whether a proposed point lies beneath $f(x)$ when the computation of $f(x)$ is complex. To illustrate the idea, Figure 4.9 compares the density function with envelope in T transformed scale and the original scale. Figure 4.10 exhibits a similar comparison with a squeeze function added in. The transformation $T(f(x)) = \log(f(x))$ provides an appropriate T transformation for the hyperbolic distribution as $\log(f(x))$ satisfies the four conditions required by this algorithm. The proof is provided later in this section.

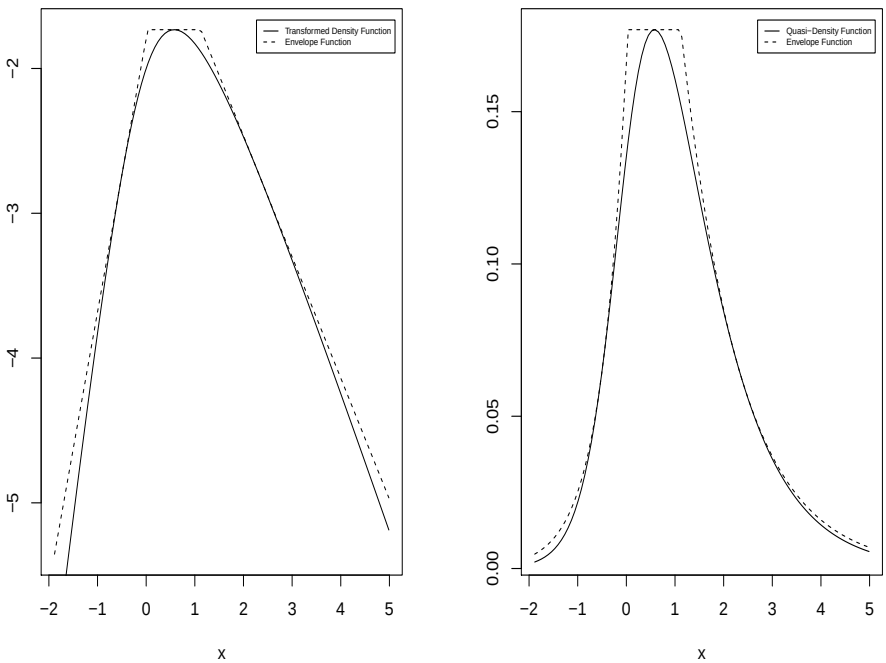

Figure 4.9: The Transformation Density Rejection Envelope when $\mu = 0$, $\delta = 1$, $\alpha = 2$, $\beta = 1$

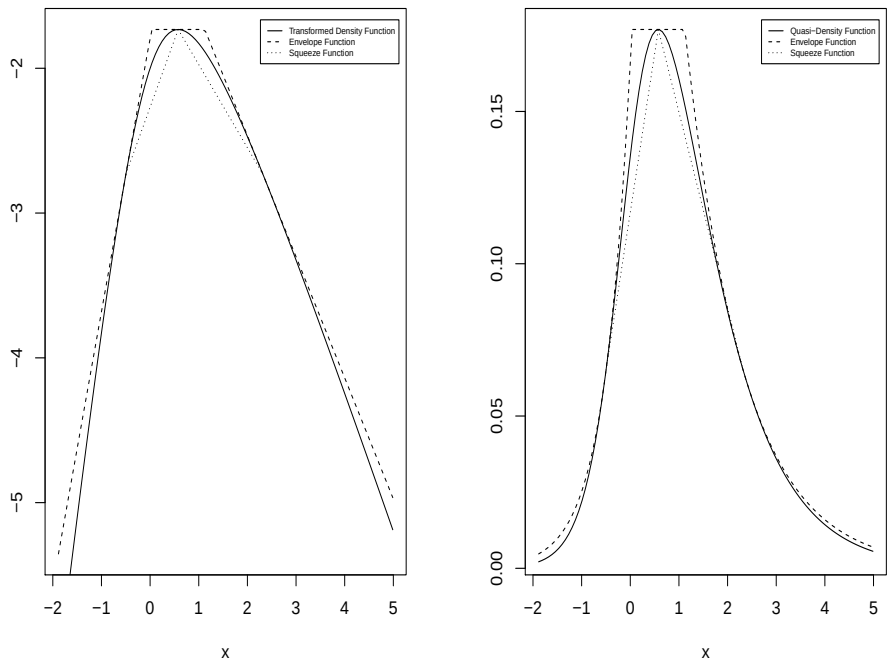

Figure 4.10: The Transformation Density Rejection Envelope with Squeeze Function when $\mu = 0$, $\delta = 1$, $\alpha = 2$, $\beta = 1$

Suppose $h(x) = T(f(x))$ is a transformed quasi-density function of the desired distribution, the algorithm in [59] is as follows:

1. Prepare a function $f(x)$ that returns values proportional to the density function of the distribution and a function $h'(x)$

2. Set $m \leftarrow$ mode of the distribution and

$$i_l \leftarrow \inf\{x | f(x) > 0\}$$
$$i_r \leftarrow \sup\{x | f(x) > 0\}$$

   where i_l and i_r need not be finite.

3. Choose x_l in the interval (i_l, m) and x_r in the interval (m, i_r)

4. Set

$$
\begin{aligned}
b_r &\leftarrow x_r + \frac{-\zeta - h(x_r)}{h'(x_r)} \\
b_l &\leftarrow x_l + \frac{-\zeta - h(x_l)}{h'(x_l)} \\
v_l &\leftarrow \frac{F(h(m)) - F(h'(x_l)(i_l - x_l) + h(x_l))}{h'(x_l)} \\
v_c &\leftarrow f(m)(b_r - b_l) \\
v_r &\leftarrow \frac{F(h'(x_r)(i_r - x_r) + h(x_r)) - F(h(m))}{h'(x_l)}
\end{aligned}
$$

5. If a squeeze function is to be used, then two squeeze functions are defined as

$$
\begin{aligned}
s_l &\leftarrow \frac{h(m) - h(x_l)}{m - x_l} \\
s_r &\leftarrow \frac{h(m) - h(x_r)}{m - x_r}
\end{aligned}
$$

6. Generate a uniform random number U and set

$$U \leftarrow U \cdot (v_l + v_c + v_r)$$

7. If $U \leq v_l$ set

$$
\begin{aligned}
X &\leftarrow \frac{F^{-1}(-Uh'(x_l) + F(h(m))) - h(x_l)}{h'(x_l)} + x_l \\
l_x &\leftarrow T^{-1}(h'(x_l)(X - x_l) + h(x_l))
\end{aligned}
$$

8. Generate a uniform random number V and set

$$V \leftarrow V \cdot l_x$$

9. If $V \leq f(X)$ return X, else go to generate U step, If a squeeze function is added then there is one more return condition: If $X < m$

   a) If $X > x_l$ and

   $$V \leq T^{-1}(h(m) - (m - x) * s_l)$$

   return X

   b) Else if $X < x_r$ and

   $$V \leq T^{-1}(h(m) - (m - x) * s_r)$$

   return X

The algorithm works with the quasi-density function of the hyperbolic distribution

$$f(x) = e^{-\alpha\sqrt{1+x^2}+\beta x} \tag{4.47}$$

As mentioned at the beginning of this section, the log transformation is a suitable T transformation. Then

$$T(f(x)) = -\alpha\sqrt{1+x^2} + \beta x \tag{4.48}$$

The log transformation is chosen since it satisfies the following prerequisite four conditions of this algorithm,

1. $\lim_{x\to 0} T(x) = -\infty$
2. $T(x)$ is differentiable and $T'(x) \geq 0$
3. $\int_0^\infty T^{-1}(h(m)-x)\mathrm{d}x < \infty$
4. $h(x) = T(f(x))$ is concave

The function $\log(x)$ for $x \geq 0$ meets the above four conditions:

$$\lim_{x\to 0} \log(x) = \log(0) = -\infty, \tag{4.49}$$

$$T'(x) = \frac{1}{x} > 0 \quad \forall x \geq 0, \tag{4.50}$$

$$\begin{aligned} \int_0^\infty T^{-1}(h(m)-x)\mathrm{d}x &= \int_0^\infty e^{(h(m)-x)}\mathrm{d}x \\ &= [-e^{(h(m)-x)}]_0^\infty \\ &= e^{h(m)} < \infty, \end{aligned} \tag{4.51}$$

$$\begin{aligned} T(f(x)) &= -\alpha\sqrt{1+x^2} + \beta x \\ \Rightarrow T'(f(x)) &= -\alpha\frac{x}{\sqrt{1+x^2}} + \beta \\ \Rightarrow T''(f(x)) &= \frac{-\alpha(1+x^2) - \alpha x^2}{\sqrt{(1+x^2)^3}} \\ &= \alpha\frac{-2x^2-1}{\sqrt{(1+x^2)^3}}. \end{aligned} \tag{4.52}$$

Since $\alpha > 0$, $-2x^2 - 1 < 0$ $\forall x \in \mathbb{R}$ then $T''(f(x)) < 0$. Therefore $T(f(x))$ is concave.

The algorithm is implemented in the functions `rhypTDR` and `rhypTDRsq` respectively with and without the squeeze function, with the formats

```
rhypTDR(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta)`.

```
rhypTDRsq(n, param)
```

`n`

Number of random variates to be generated.

`param`

Parameter vector taking the form `c(mu, delta, alpha, beta)`.

Figures 4.11 and 4.12 present histograms of random variates from the hyperbolic distribution with $\mu = 0$, $\delta = 1$, $\alpha = 1$, $\beta = 0.5$ generated by `rhypTDR` and `rhypTDRsq` respectively. The density curves are superimposed.

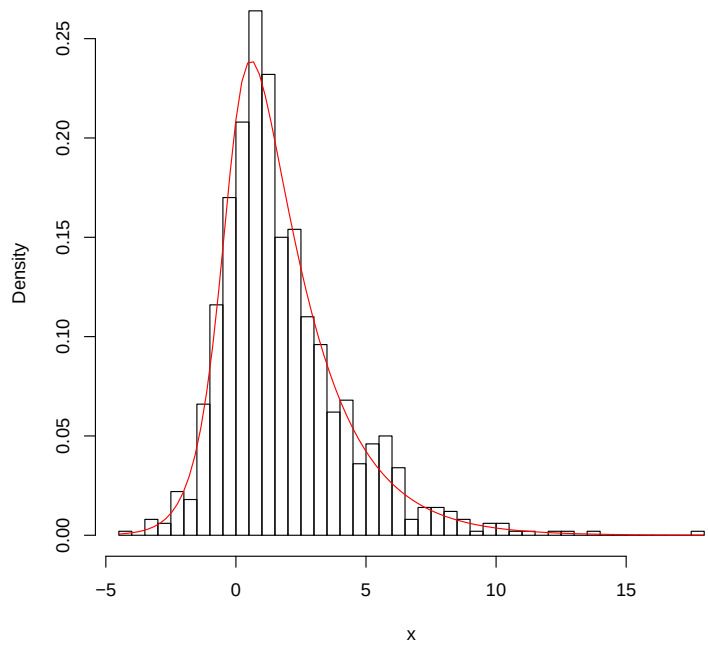

Figure 4.11: Histogram of random variates from the hyperbolic distribution generated by the TDR algorithm

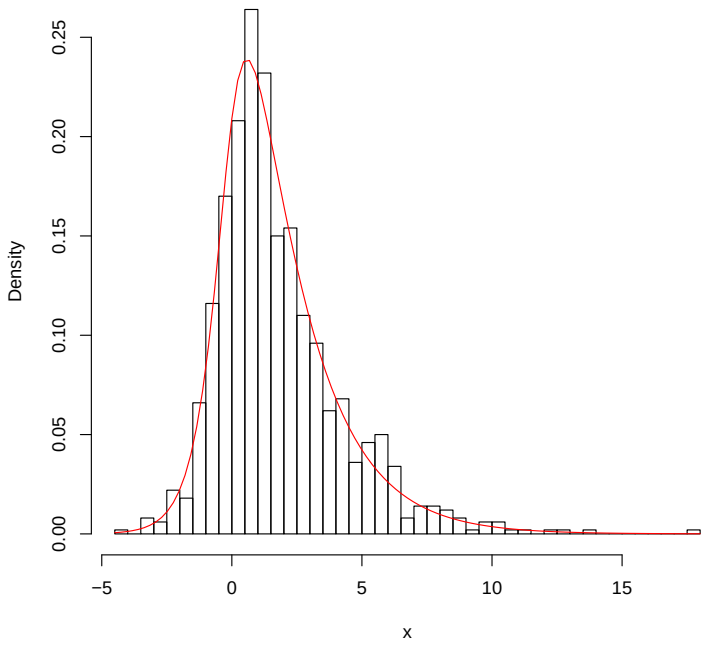

Figure 4.12: Histogram of random variates from the hyperbolic distribution generated by the TDR algorithm with squeeze function

Table 4.7 shows acceptance probabilities for the transformation density rejection approach. The probabilities are in general at satisfactory level.

Table 4.7: Acceptance Probabilities of the TDR Method

	α			
β	0.5	1	2	5
0	0.9346	0.9346	0.9259	0.9346
0.2α	0.9709	0.9259	0.8772	0.8696
0.8α	0.9804	0.9346	0.9615	0.8772

Table 4.8 shows acceptance probabilities of the transformation density rejection approach with squeeze function. The probabilities are at satisfactory level.

Table 4.8: Acceptance Probabilities of the TDR Method with Squeeze Function

	α			
β	0.5	1	2	5
0	0.9524	0.9615	0.9091	0.8929
0.2α	0.9615	0.9259	0.9009	0.9259
0.8α	0.9804	0.9709	0.9346	0.9009

4.3 Analysis

The focus in this section is the comparison of the time taken to generate a single random variate using the different the algorithms discussed previously for sampling from the GIG distribution and the hyperbolic distribution.

4.3.1 Generalized Inverse Gaussian Distribution

For parameter values within the commonly used range the functions to compare for the GIG distribution are `rgigAtkin`, `rgig`, `rgigGamma`, and `ur(pinv.new(ugig))`. The function `rgigHoer` is not included in this first comparison as it is only applicable when $0 < \lambda < 1$, $\beta \leq \min(1/2, 2/(3\sqrt{1-\lambda}))$.

Table 4.9 shows the ratio of the average time taken to generate one random variate by function `rgigAtkin` compared to the default function `rgig`. The ratio of uniforms with shifted mode approach appears to be superior to the Atkinson rejection method. The time that `rgigAtkin` function takes to generate one random variate is at least 1.71 times that of the `rgig` function.

Table 4.9: Ratio of `rgigAtkin` and `rgig`

	λ			
β	0.01	0.1	1	10
0.01	11.54	12.51	1.71	2.22
0.1	4.90	5.06	2.51	2.56
1	2.60	2.36	2.07	2.06
10	2.20	2.08	2.16	2.10

Table 4.10 shows the ratio of the average time taken to generate one random variates by function `rgigGamma` compared to the default function `rgig`. Dagpunar's rejection method appears to faster when $\lambda >= 1$. However the method has variable performance. When $\lambda \to 0$, the efficiency of this method declines and the `rgig` function becomes faster.

Table 4.10: Ratio of `rgigGamma` and `rgig`

	λ			
β	0.01	0.1	1	10
0.01	2.23	0.55	0.52	0.51
0.1	8.74	1.25	0.56	0.66
1	16.45	2.23	0.68	0.51
10	28.59	3.61	0.93	0.53

Table 4.11 shows the ratio of the average time taken to generate one random variate by the function `ur(pinv.new(ugig))` compared to the default function `rgig`. The fast inversion method appears to be superior to the ratio of uniforms with shifted mode method when either $\lambda > 0.01$ or $\beta > 0.01$ and performs very reliably. Note however that the comparison between these two method is not strictly fair. As mentioned in Section 4.1.5, the function `ur(pinv.new(ugig))` is written in Ch, an embeddable C/C++ interpreter for cross-platform scripting, which is a much more efficient language than R. In addition, the fast inversion method requires setting up a fairly large table to store the interpolation points, therefore implementing the method using R would not achieve the efficiency that the `ur(pinv.new(ugig))` function has achieved at present.

Table 4.11: Ratio of `ur(pinv.new(ugig))` and `rgig`

	λ			
β	0.01	0.1	1	10
0.01	1.07	0.145	0.24	0.19
0.1	0.19	0.22	0.24	0.24
1	0.25	0.25	0.25	0.21
10	0.26	0.22	0.25	0.24

From the comparisons above, the ratio of uniforms with shifted mode method appears to be best method in terms of the combination of stability, efficiency and simplicity over the commonly used range of parameter values. However, the method has a known problem when λ and β both become extremely small, of order 10^{-7}. Hence the following comparisons will focus on extremely small parameter values in order to find the best complement to the `rgig` algorithm for this case.

The candidate algorithms are the fast inversion method, Dagpunar rejection sampling method and Hörmann's rejection sampling method. Table 4.12 shows the ratio of average time taken to generate one random variate by the function `ur(pinv.new(ugig))` compared to the function `rgigHoer`. The Hörmann rejection sampling method approach appears to be superior to the fast inversion method. The time that `ur(pinv.new(ugig))` function takes to generate one random variate is at least 4.39 times of the `rgigHoer` function. In addition, as mentioned earlier in this

section, the function `ur(pinv.new(ugig))` is mainly written in the Ch language which offers a speed advantage compared to code written in R.

Table 4.12: Ratio of `ur(pinv.new(ugig))` and `rgigHoer`

	λ		
β	0.01	0.1	0.5
10^{-14}	10.16	11.57	4.39
10^{-12}	6.39	10.63	5.00
10^{-10}	7.50	7.09	8.88

Table 4.13 shows the ratio of the average time taken to generate one random variate by the function `rgigGamma` compared to the function `rgigHoer`. The Dagpunar rejection sampling method appears to be faster than the Hörmann rejection sampling method approach. However the `rgigGamma` function appears to be unstable in that it has difficulty in generating random variates at some parameter values.

Overall the `rgigHoer` function is the best method to complement the ratio of uniforms with shifted mode method in terms of stability and efficiency.

Table 4.13: Ratio of `ur(pinv.new(ugig))` and `rgigHoer`

	λ		
β	0.01	0.1	0.5
10^{-14}	∞	∞	0.26
10^{-12}	0.57	∞	0.60
10^{-10}	0.79	0.86	∞

Figure 4.13 and 4.14 present a histogram of random variates from the GIG distribution with $\beta = 10^{-7}$, $\lambda = 0.5$ generated by `rgigHoer`. The density curve is superimposed. Since the GIG distribution with $\beta = 10^{-7}$ and $\lambda = 0.5$ is extremely skewed, the random variates of left tail presented in Figure 4.13 visually appear to form a greater spike than the density curve. However the zoomed-in left tail which is plotted in Figure 4.14 illustrates that the random variates generated by `rgigHoer` follow the density curve reasonably well in the extreme tail area.

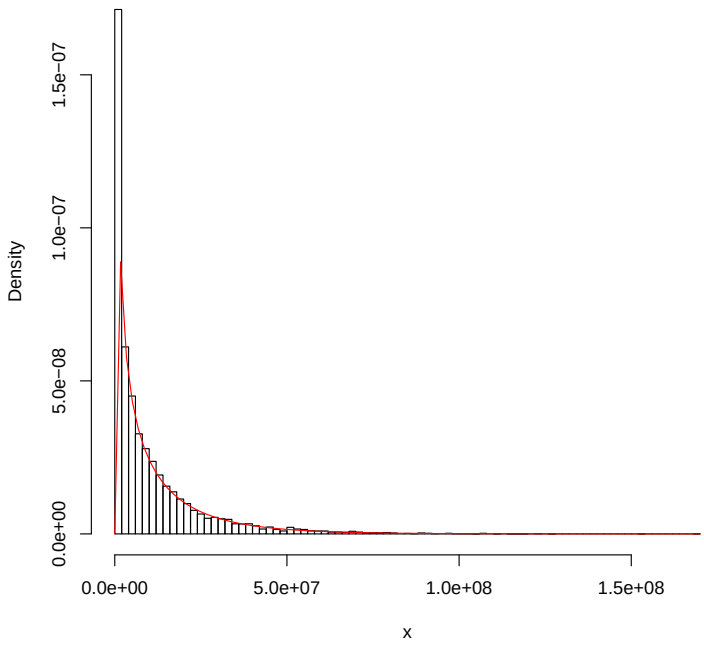

Figure 4.13: Histogram of random variates from the GIG distribution generated by Hörmann envelope

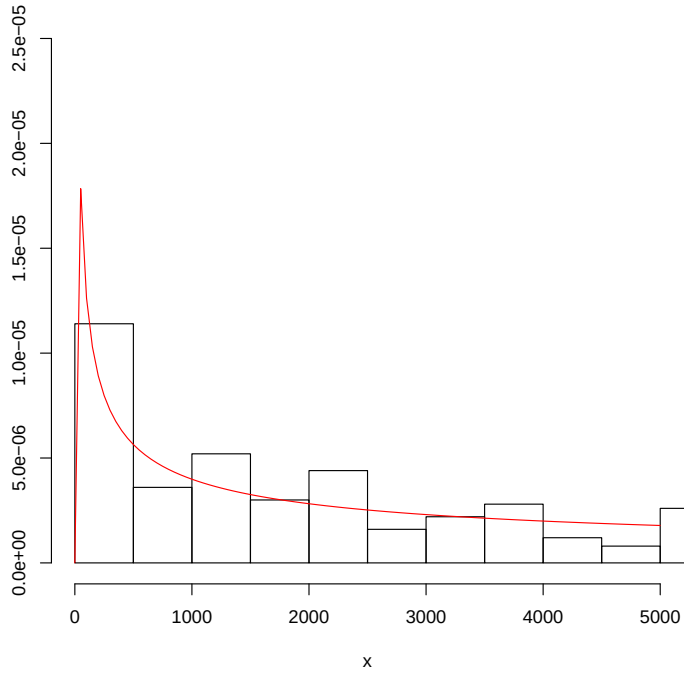

Figure 4.14: The left tail of a histogram of random variates from the GIG distribution generated by Hörmann envelope

4.3.2 Hyperbolic Distribution

The comparisons in this section are between the functions `rhyperb`, `rhyperbTDR` and `rhyperbTDRsq`. As stated in Section 3.2, the hyperbolic distribution is the subclass of the GHyp distribution with $\lambda = 1$, therefore random variates can be generated by using the GHyp distribution with $\lambda = 1$. This is implemented in the function `rhyperb` which uses the mixture property of the GIG distribution along with the ratio of uniforms with shifted mode method. This algorithm was claimed to be superior to the ratio of uniforms method, the slice sampler and the Atkinson three part envelope rejection method by Richard Trendall in his honours project.

Table 4.14 shows the ratio of the average time taken to generate one random variate by function `rhyperbTDR` compared to the function `rhyperb`. Table 4.15 shows the same comparison between the function `rhyperbTDRsq` and `rhyperb`. The ratio of uniforms with shifted mode method appears to be more efficient than the transform density function method regardless of the use of a squeeze function.

Table 4.14: Ratio of `rhyperbTDR` and `rhyperb`

	α					
β	0.1	0.2	0.5	1	2	5
0	10.49	11.43	13.59	11.54	12.71	12.26
0.1α	11.77	12.20	13.13	15.52	12.29	11.77
0.2α	10.31	11.85	12.29	14.47	13.43	14.35
0.5α	9.93	9.35	11.53	12.84	13.37	14.87
0.8α	8.57	10.21	11.84	11.64	11.57	12.46
0.9α	9.71	10.01	10.70	10.42	13.60	11.89

Table 4.15: Ratio of `rhyperbTDRsq` and `rhyperb`

	α					
β	0.1	0.2	0.5	1	2	5
0	10.57	11.48	13.54	11.51	12.63	12.27
0.1α	11.72	12.29	13.25	15.54	12.35	11.86
0.2α	10.28	11.75	12.38	14.69	13.33	14.50
0.5α	9.97	9.51	11.56	12.98	13.34	14.81
0.8α	8.55	10.13	11.92	11.80	11.55	12.54
0.9α	9.65	10.96	10.88	10.53	13.62	11.85

4.3.3 Conclusion

From the comparison results in Section 4.3.1 the ratio of uniforms with shifted mode method for the GIG distribution with $\lambda \geq 1$ or $\beta > \min[1/2, 2/(3\sqrt{1-\lambda})]$ and Hörmann's rejection sampling method for GIG distribution with $0 < \lambda < 1$ and $\beta \leq \min[1/2, 2/(3\sqrt{1-\lambda})]$ forms the best solution to generate random variates from the GIG distribution.

Recall from Section 4.1, random variates from the GHyp $(\alpha, \beta, \delta, \mu)$ distribution can be generated using the representation of the GHyp distribution as a mixture of the normal distribution and the GIG distribution. Therefore the best solution for generating random variates from the GHyp distribution is mixing a normally distributed random variate and a GIG distributed variate with $\lambda \geq 1$ or $\beta > \min[1/2, 2/(3\sqrt{1-\lambda})]$ random variates from the ratio of uniforms with shifted mode or the GIG distributed with $0 < \lambda < 1$ and $\beta \leq \min[1/2, 2/(3\sqrt{1-\lambda})]$ from Hörmann's rejection sampling method.

From the comparison results in Section 4.3.2 the random variates of the GHyp distribution with $\lambda = 1$ that are generated by mixing the normal distributed random variates and the GIG distributed random variates that are generated by the ratio of uniforms with shifted mode method

provide the best performance for hyperbolic distribution random variate generation. The special case of the GIG distribution with $0 < \lambda < 1$ and $\beta \leq \min(1/2, 2/(3\sqrt{1-\lambda}))$ is not applicable to the hyperbolic distribution because the value of λ for the hyperbolic distribution is 1.

CHAPTER 5

Robust Linear Modeling using the Hyperbolic Distribution

Linear regression is a very important statistical tool in many fields. However, when the data violates the underlying assumptions, the results of the analysis can be misleading. One of the major assumptions of linear regression is that the error terms are normally distributed.

There is considerable interest in the literature in regards to the robust procedures to model non-normally distributed data, in particular data with longer than normal tails. Comprehensive reviews of the robust inferences are given in [76], [62], [54], [63].

There are numbers of approaches to handling data with such characteristics. Applying a transformation to the response variable appears to be the most common approach. There are a number of potential transformations including but not limited, to the inverse hyperbolic sine transformation discussed in [29], the well-known Box-Cox transform discussed in [25] and its extensions discussed in [110] and [65]. However it has been pointed out in [46] that there are three major drawbacks to this approach,

1. The resulting fitted regression coefficients can be difficult to interpret.

2. The transformation affects other aspects of the model, for instance heteroscedaticity.

3. Some scholars have negative views on optimizing on the transformed scale implicitly.

A different approach is the replacement of normal-theory estimators and/or their standard errors by methods suitable for non-normal errors. [61] and [84] proposed the use of M-estimation, a *'maximum likelihood type'* estimation, as an alternative to least squares estimation. This method is robust to outliers in the response variable, but turned out not to be resistant to leverage points. Other proposals for robust estimators are the L-estimator, a *linear combinations*

of order statistics and the R-estimator, an *estimate derived from rank test* discussed in [62]. Apart from the use of alternative estimators, there is literature concerned with appropriate calculation of the standard errors. [27], [99] and [108] discussed modified approaches for normal covariance-structure tests under elliptically distributed errors, whilst [111] proposed the linear regression model with least squares estimated explanatory variables which possess inflated standard errors.

The approach used in my research is a parametric approach which assumes the distribution of the error terms follow other more flexible distributions. The advantage of this approach is that the regression coefficients can be estimated by likelihood methods, and the examination of residuals can act as part of assessing the appropriateness of the model. In the existing literature, the t distribution appears to be the most popular alternative distribution for the error terms. [76] proposed a robust statistical model based on maximum likelihood for a general model with multivariate t errors from a frequentist perspective. They argued that the strategy combines conceptual simplicity with generality. [48] also proposed the t distribution from a Bayesian perspective. In [66], a tractable skew extension of the t distribution was proposed as a further alternative for the normal and t distributions. The density function of this distribution is defined as

$$f(t;a,b) = C_{a,b}^{-1}\left\{1+\frac{t}{(a+b+t^2)^{1/2}}\right\}^{a+1/2}\left\{1-\frac{t}{(a+b+t^2)^{1/2}}\right\}^{b+1/2}, \tag{5.1}$$

where $a>0$, $b>0$, $C_{a,b} = 2^{a+b-1}B(a,b)(A+b)^{1/2}$, and B is the beta function. There are two simple examples presented in [66] to illustrate the apparent usefulness of this distribution.

Apart from the t distribution family, the skew-normal distribution family which is a superset of the normal family has been introduced in [4]. [5], [6] and [7] discussed the distribution in detail and proposed that it can serve as an alternative to normal distribution in statistical applications. The density function of Azzalini's version of the univariate skew-normal distribution is defined as

$$\phi(z;\xi,\omega,\alpha) = 2\phi(z;\xi,\omega)\Phi(\alpha z;\xi,\omega), \tag{5.2}$$

where ϕ and Φ are the standard normal density function and CDF respectively, and α is the shape parameter. When $\alpha = 0$, the distribution reduces to the standard normal distribution $N(\xi,\omega^2)$.

In this chapter, we focus on fitting the hyperbolic distribution to data which is heavier-tailed than normal and skewed. In [107], Richard Trendall investigated fitting a linear regression model to data assuming hyperbolically distributed errors, in the hope of providing a solution for data that violates the assumption of the standard leasts square model. The basic code in `R` was developed by him, however there remained some important matters requiring further research:

- The fitting function has problems finding the maximum values of the log likelihood function. Although some attempts were made to solve the problem, none of those techniques proved successful.
- Appropriate studies regarding fitting the hyperbolic linear model to datasets of different sizes were lacking

- Studies regarding the suitability of the hyperbolic regression model for modeling real data were lacking

In this chapter an improved algorithm is proposed and investigated which provides a much more reliable platform. Hyperbolic linear regression is demonstrated with some real data examples and the results compared with linear regression models assuming errors from other distributions.

5.1 Methodology

5.1.1 Regression Algorithm

The hyperbolic regression model takes the form of ordinary leasts square regression but instead of assuming normally distributed errors, it assumes the errors have a hyperbolic distribution. Then the hyperbolic regression model has the form

$$y = X\gamma + e \tag{5.3}$$

where e is a hyperbolic error term with mean 0 and distribution given by

$$e \sim \mathrm{Hyp}(\mu, \delta, \alpha, \beta), \tag{5.4}$$

with

$$\mathbb{E}[e] = \mu + \beta\sqrt{\frac{\delta^2}{\alpha^2-\beta^2}}\frac{K_2(\delta\sqrt{\alpha^2-\beta^2})}{K_1(\delta\sqrt{\alpha^2-\beta^2})} = 0. \tag{5.5}$$

$\mathbb{E}[Y]$ is set equal to a linear predictor of the form

$$\mathbb{E}[Y] = \gamma_0 + \gamma_1 x_1 + \gamma_2 x_2, \ldots, +\gamma_n x_n. \tag{5.6}$$

Since ψ and ζ are more convenient constraints than α and β, the log likelihood is maximized over the alternative parameterization $\pi = \beta/\sqrt{\alpha^2-\beta^2}$, $\zeta = \delta\sqrt{\alpha^2-\beta^2}$.

The hyperbolic distribution has parameters and a fairly complex density function. Therefore the optimization routine can be fragile without careful consideration. Some innovations were introduced to stabilize the routine:

- reduce the number of parameters in the optimization routine;
- provide good starting values; and
- use multi-stage optimization.

As (5.5) shows, the mean of the hyperbolic distribution includes the location parameter μ. Therefore μ is eliminated during the fitting process by first letting

$$S = \beta\sqrt{\frac{\delta^2}{\alpha^2-\beta^2}}\frac{K_2(\delta\sqrt{\alpha^2-\beta^2})}{K_1(\delta\sqrt{\alpha^2-\beta^2})}. \tag{5.7}$$

Then when $\mu = -S$, $\mathbb{E}(e) = 0$.

The likelihood function of the error distribution has the form

$$\mathrm{L}(\gamma_0, \ldots, \gamma_n, \delta, \alpha, \beta) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\delta\alpha \mathrm{K}_1(\delta\sqrt{\alpha^2 - \beta^2})} \cdot e^{-\alpha \sum \sqrt{\delta^2 + (y_i - \gamma_0 - \ldots - \gamma_n + S)^2} + \beta \sum (y_i - \gamma_0 - \ldots - \gamma_n + S)} \tag{5.8}$$

so the optimization only involves $\alpha, \beta, \delta, \gamma_0, \ldots, \gamma_n$.

By changing the regression intercept to $\gamma_0^* = \gamma_0 - S$, the error distribution for the fitting process is given by

$$e^* \sim \mathrm{Hyp}(0, \delta, \alpha, \beta) \tag{5.9}$$

where $\mathbb{E}[e^*] = S$. After the model has been fitted, the estimate $\hat{\mu}$ of the original error distribution e is given by $-\mathbb{E}[e^*]$. Also the estimate $\hat{\gamma_0} = \hat{\gamma_0^*} + \mathbb{E}[e^*]$.

5.1.2 Fitting the Hyperbolic Linear Model

The fitting process consists of two parts: finding the appropriate start values; and maximizing the log likelihood function. As with most iterative optimizations, the quality of start values, especially for distribution parameters, is crucial for the success of the optimization routine ([107]). We first use the QR decomposition to obtain the values for the regression coefficients $\gamma_0, \ldots, \gamma_n$, then the residuals are fitted using the `hyperbFitStand` function ([96]) to obtain the start values for the hyperbolic distribution. This function standardizes the residuals to mean = 0 and variance = 1. Recall that the hyperbolic distribution is the subclass of the GHyp distribution with $\lambda = 1$. The transformed residuals are then fitted by the standardized GHyp distribution $\mathrm{sh}(x, \zeta, \rho)$ with $\lambda = 1$ as proposed in [96].

Definition 9. *The standardized GHyp distribution is defined as*

$$sh(x, \zeta, \rho) = GHyp(\lambda, \alpha^*, \beta^*, \delta^*, \mu^*) \tag{5.10}$$

where $(\lambda, \alpha^*, \beta^*, \delta^*, \mu^*)$ is a fifth parameterization closely related to the second parameterization (ζ, ρ) where

$$\alpha^* = \left\{ \frac{\zeta^2}{1-\rho^2} K_\lambda(\zeta)(1 + \frac{\rho^2\zeta^2}{1-\rho^2} \Delta K_\lambda(\zeta)) \right\}^{1/2} \tag{5.11}$$

$$\beta^* = \alpha^* \rho \tag{5.12}$$

$$\delta^* = \frac{1}{\alpha^*} \frac{\zeta}{\sqrt{1-\rho^2}} \tag{5.13}$$

$$\mu^* = -\frac{1}{\alpha^*} \frac{\rho\zeta^2}{1-\rho^2} K_\lambda(\zeta). \tag{5.14}$$

The fitting procedure for the start values of the error distribution parameters uses the maximum likelihood technique. The standardization is beneficial because the standardized hyperbolic distribution has fewer parameters than the hyperbolic distribution, i.e. the optimization dimension is reduced.

The fitted parameters (ζ, ρ) are then transformed back to the unstandardized scale. The start value of γ_0^* can thus be found by simple calculation. The start values found by this approach provide a reliable platform for the following optimization.

Once we have obtained the start values, the process proceeds to the second part. As mentioned in Section 5.1.1, we use a two stage alternating optimization method to optimize the likelihood function. This approach limits fitting the hyperbolic distribution to residuals only involving the distribution parameters. This ensures that in cases where the model contains many explanatory variables, the optimization is still stable. In the first stage, the log likelihood function is optimized with respect to the regression coefficients given the values of distribution parameters. In the second stage, the log likelihood function is optimized with respect to the distribution parameters given the regression coefficient values from the first stage. These two stages alternate until both reach convergence or the routine exceeds the maximum number of iterations allowed. The maximum changing ratios $r_{\text{coef}}, r_{\text{param}}$ are used to determine when the routine reaches convergence. Let $\gamma_0^{*(k)}, \ldots, \gamma_n^{(k)}$ be the regression coefficient estimates and $\delta^{(k)}, \pi^{(k)}, \zeta^{(k)}$ be the distribution parameter estimates after the kth iteration. Then

$$r_{\text{coef}} = \max\left(\frac{\left|\gamma_0^{*(k)} - \gamma_0^{*(k+1)}\right|}{\gamma_0^{*(k)}}, \ldots, \frac{\left|\gamma_n^{(k)} - \gamma_n^{(k+1)}\right|}{\gamma_n^{(k)}} \right) \tag{5.15}$$

and

$$r_{\text{param}} = \max\left(\frac{\left|\delta^{(k)} - \delta^{(k+1)}\right|}{\delta^{(k)}}, \frac{\left|\pi^{(k)} - \pi^{(k+1)}\right|}{\pi^{(k)}}, \frac{\left|\zeta^{(k)} - \zeta^{(k+1)}\right|}{\zeta^{(k)}} \right) \tag{5.16}$$

If $r_{\text{coef}} \leq$ tolerance then the first stage converges. If $r_{\text{param}} \leq$ tolerance then the second stage converges.

There are three optimization methods, using either the `optim` or the `nlm` function, implemented in the first stage:

- "Nelder-Mead" An implementation of the Nelder-Mead simplex method which uses the `optim` function.
- "BFGS' A quasi-Newton method, which is also implemented using `optim`.
- "nlm" The `nlm` function which is an implementation of the method proposed by Schnabel, Koontz and Weiss ([90]).

These three approaches were tested and compared with 290 artificial datasets. The datasets were created by setting the coefficients arbitrarily i.e $\gamma_0 = 2000$, $\gamma_1 = -25$ with the error distribution over a designated parameter range. The tail parameter α and the skewness parameter β

were determined by choosing values of ψ and χ which range over the shape triangle described for instance in [85]. The maximum log likelihood values and the percentage error of the coefficient estimates calculated as

$$\varepsilon_i = \frac{\hat{\gamma}_i - \gamma_i}{\gamma_i} \times 100, \quad i = 0, 1, \tag{5.17}$$

were compared for the three different optimization methods.

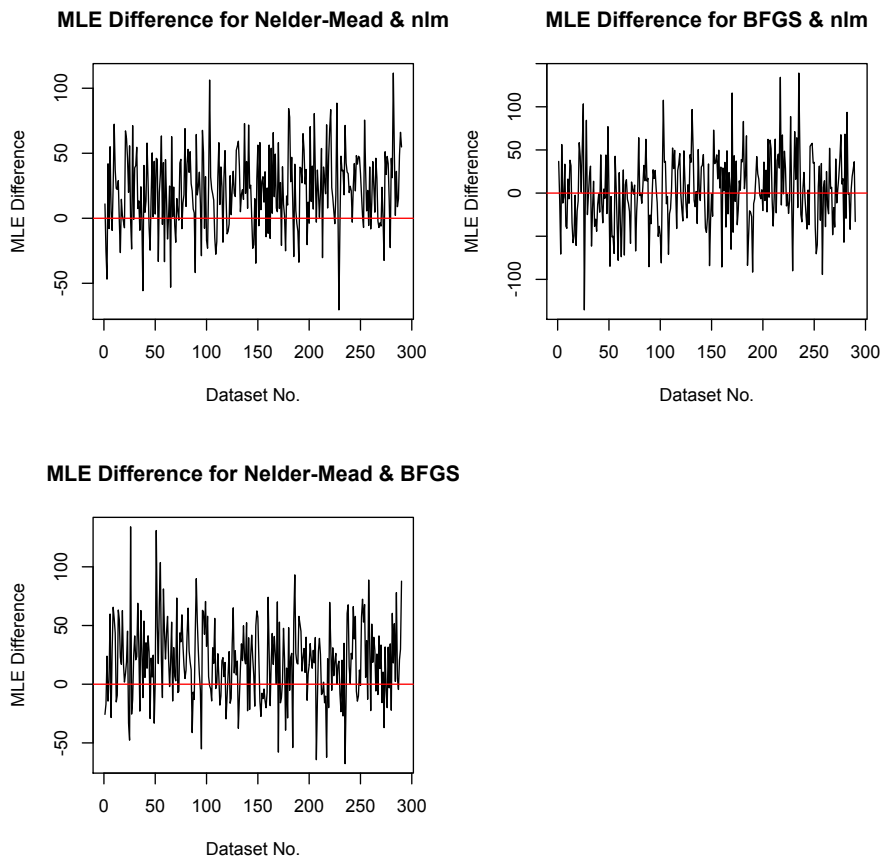

Figure 5.1: Maximum Likelihood Estimate Differences

Figure 5.1 shows the differences of maximum log likelihood values between each pair of optimization methods. The difference between maximum log likelihood values estimated by the Nelder-Mead method and the nlm method are in general greater than zero, likewise the difference between the Nelder-Mead method and the nlm method, whilst the difference between the BFGS method and the nlm method is around zero. Thus these three methods estimate somewhat different maximum log likelihood values for the same dataset.

Table 5.1: Maximum Percentage Error

	γ_0	γ_1
Nelder-Mead	0.57	0.14
BFGS	3.28	0.44
nlm	1.78	0.26

Table 5.1 shows the maximum percentage errors of 290 datasets. The performance of the Nelder-Mead method is superior to the other two methods with maximum percentage errors of only 0.57 and 0.14. Therefore the default method for the `hyperblm` function is the Nelder-Mead method.

The second stage uses `constrOptim` to optimize the log likelihood with constraints $\delta > 0$ and $\zeta > 0$. The `constrOptim` function minimizes the objective function subject to linear inequality constraints using the barrier method with logarithmic barrier function ([75]).

5.1.3 Confidence Intervals of Parameters

Wald's confidence intervals are implemented to construct the confidence intervals for our coefficient estimates $\hat{\gamma}_i$. Wald's confidence interval for θ_i is obtained by

$$\hat{\theta}_i \pm t_{\alpha/2} * \mathrm{se}(\hat{\theta}_i) \tag{5.18}$$

There are two methods implemented to calculate the standard error $\mathrm{se}(\hat{\theta}_i)$ of the regression coefficients and the distribution parameters. The first one is by calculating the Hessian matrix

which is given by ([107])

$$
\begin{aligned}
H_{11} &= n\,\frac{1-\pi^2}{1+\pi^2} + \frac{\zeta}{(1+\pi^2)^{3/2}}\,t_{0,-1} \\
H_{12} = H_{21} &= \zeta\left[\frac{\pi}{\sqrt{1+\pi^2}}\,t_{0,-1} - t_{1,0}\right] \\
H_{13} = H_{31} &= \zeta\left[t_{1,0} - \frac{\pi}{\sqrt{1+\pi^2}}\,t_{2,1}\right] \\
H_{14} = H_{41} &= \frac{\zeta}{\delta}\left[n - \frac{\pi}{\sqrt{1+\pi^2}}\,t_{1,1}\right] \\
H_{22} &= \zeta\left[n\zeta S(\zeta) - 2nR(\zeta) + \sqrt{1+\pi^2}\,t_{0,-1} - \pi t_{1,0}\right] \\
H_{23} = H_{32} &= \zeta\left[\pi t_{1,0} - \sqrt{1+\pi^2}\,t_{2,1}\right] \\
H_{24} = H_{42} &= \frac{\zeta}{\delta}\left[n\pi - \sqrt{1+\pi^2}\,t_{1,1}\right] \\
H_{33} &= \zeta\left[\sqrt{1+\pi^2}\,\left(2t_{2,3} + t_{4,3}\right) - \pi t_{1,0}\right] \\
H_{34} = H_{43} &= \frac{\zeta}{\delta}\left[\sqrt{1+\pi^2}\,\left(2t_{1,3} + t_{3,3}\right) - n\pi\right] \\
H_{44} &= \frac{\zeta}{\delta^2}\sqrt{1+\pi^2}\,t_{0,3}
\end{aligned}
$$

$$
\begin{aligned}
H_{1(4+j)} &= H_{(4+j)1} \\
&= \frac{\zeta}{\delta}\sum_{i=1}^{n}\left\{1 - \frac{\pi}{\sqrt{1+\pi^2}}\,\frac{(y_i - \mathbb{E}[Y])/\delta}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{1/2}}\right\}X_{ji}
\end{aligned}
$$

$$
\begin{aligned}
H_{2(4+j)} &= H_{(4+j)2} \\
&= \frac{\zeta}{\delta}\sum_{i=1}^{n}\left\{\pi - \sqrt{1+\pi^2}\,\frac{(y_i - \mathbb{E}[Y])/\delta}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{1/2}}\right\}X_{ji}
\end{aligned}
$$

$$
\begin{aligned}
H_{3(4+j)} &= H_{(4+j)3} \\
&= \frac{\zeta}{\delta}\sqrt{1+\pi^2}\sum_{i=1}^{n}\left\{2\frac{(y_i - \mathbb{E}[Y])/\delta}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{3/2}} + \frac{(y_i - \mathbb{E}[Y])^3/\delta^3}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta)^2\right]^{3/2}}\right\}X_{ji}
\end{aligned}
$$

$$
\begin{aligned}
H_{4(4+j)} &= H_{(4+j)4} \\
&= \frac{\zeta}{\delta^2}\sqrt{1+\pi^2}\sum_{i=1}^{n}\frac{X_{ji}}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{3/2}}
\end{aligned}
$$

$$
\begin{aligned}
H_{(4+j)(4+m)} &= H_{(4+m)(4+j)} \\
&= \frac{\zeta}{\delta^2}\sqrt{1+\pi^2}\sum_{i=1}^{n}\frac{X_{ji}X_{mi}}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{3/2}}
\end{aligned}
$$

where

$$R_\lambda(\zeta) = \frac{K_{\lambda+1}(\zeta)}{K_\lambda(\zeta)}$$
$$S_\lambda(\zeta) = \frac{K_{\lambda+2}(\zeta)K_\lambda(\zeta) - K^2_{\lambda+1}(\zeta)}{K^2_\lambda(\zeta)}$$
$$t_{r,k} = \sum_{i=1}^{n} \frac{(y_i - \mathbb{E}[Y])^r/\delta^r}{\left[1 + (y_i - \mathbb{E}[Y])^2/\delta^2\right]^{k/2}}.$$

The Hessian matrix was obtained by Richard Trendall in his thesis ([107]) using the information matrix of the hyperbolic distribution given by [13]. Although the Hessian matrix is given here in analytical form, it has to be approximated numerically as it contains the unknown parameters. The function `tsHessian` from the **DistributionUtils** package is used in the numerical approximaton. This estimated Hessian matrix however is not always guaranteed to be positive definite which was pointed out in [13]. Two solutions are provided for this issue. Firstly, we use `make.positive.definite`, an implementation of the EST algorithm proposed by [57] in 1988, from the **corpor** package, to compute the nearest symmetric positive semidefinite matrix. Secondly, we implement the bootstrap approach to estimate se($\hat{\theta}_i$). The bootstrap, first proposed by B. Efron in 1979, is a computer-based method for assigning measures of accuracy to statistical estimates([42]). The parametric bootstrap method is set out as

1. Generate a random sample the same size y from the estimated hyperbolic error distribution $e_B \sim \text{Hyp}(\widehat{\mu}, \widehat{\delta}, \widehat{\alpha}, \widehat{\beta})$
2. Let $y_B = x_i\hat{\gamma}_I + e_B$
3. Recalculate the maximum likelihood values which are considered to be the bootstrap estimates $(\widehat{\gamma}_{1B}, \ldots, \widehat{\gamma}_{nB}, \widehat{\mu}_B, \widehat{\delta}_B, \widehat{\alpha}_B, \widehat{\beta}_B)$
4. Repeat the above three procedures until the bootstrap sample size for the maximum likelihood estimates is reasonable.

The parametric bootstrap approach is inefficient but more reliable than the estimation of the Hessian matrix. For this reason it is set as the default method for the `summary` function.

5.2 Applications

Three real data examples are provided here to demonstrate the regression with hyperbolic errors functions. The first one is the classic stack loss data which is obtained from 21 days of operation of a plant for the oxidation of ammonia (NH3) to nitric acid (HNO3). The nitric oxides produced are absorbed in a countercurrent absorption tower (source [28]). The second one fits the linear model with hyperbolic errors to data on 349 nursing facilities in the state of Wisconsin in the cost report year 2001 (source [46]). The third one derives a capital asset pricing model (CAPM) for the Apple daily closing share price from January 2004 to December 2010.

5.2.1 Stack-Loss Data

The stack loss data has 4 attributes as shown in Table 5.2.

Table 5.2: Stack-Loss Data

Variable Name	Description
`stack.loss`	Stack loss
`airflow`	Flow of cooling air
`temperature`	Cooling Water Inlet Temperature
`acid`	Concentration of acid (per 1000, minus 500)

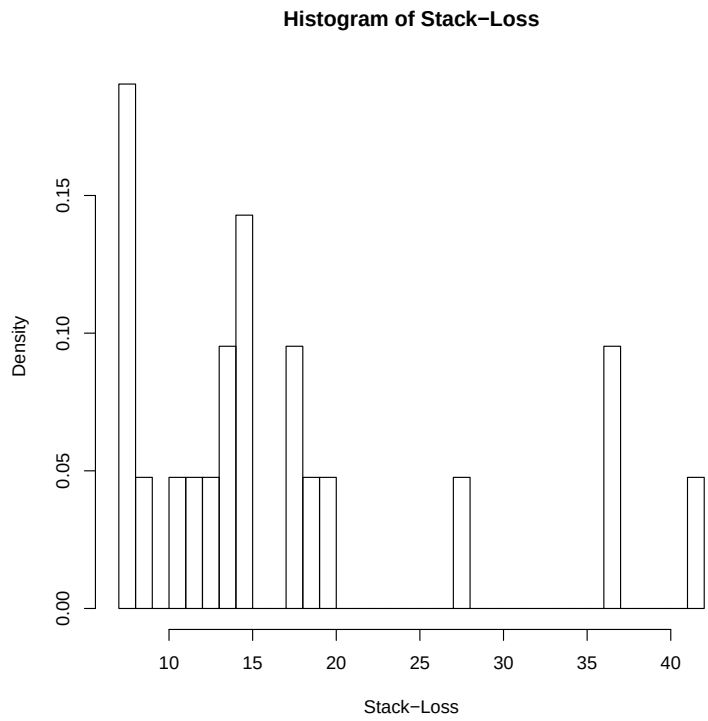

Figure 5.2: Histogram of Stack-Loss

As Figure 5.2 indicates, the response variable `stack.loss` is fairly right skewed with some notably large values. The sample size is small ($n = 21$), and the normal assumption is not appropriate. The result of fitting the hyperbolic model is as below:

```
HModel1 <- hyperblm(stack.loss ~ airflow + temperature + acid)
summary.hyperblm(HModel1)

Call:
hyperblm(formula = stack.loss ~ airflow + temperature + acid)
```

```
Data:      (Intercept) airflow temperature acid
Parameter estimates:
              Estimate Std. Error t value  Pr(>|t|)
(Intercept) -36.979982  15.218735 -2.4299  0.029149 *
airflow       0.812946   0.083668  9.7163 1.333e-07 ***
temperature   0.764824   0.241937  3.1613  0.006935 **
acid         -0.124650   0.154007 -0.8094  0.431838
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error distribution parameter estimates:
      Estimate Std. Error
mu    -1.07157     1.0802
delta  0.19049     0.0631
alpha  0.55297     0.1861
beta   0.17378     0.2182

Likelihood:          -51.06739
Method:              Nelder-Mead
Convergence code:    0
Iterations:          6
```

The hyperbolic model indicates that `airflow` and `temperature` are significant explanatory variables. [76] showed that fitting the t model with estimated parameter $\hat{\nu} = 1.1$ to the data significantly improves the fit (likelihood ratio chi-squared statistic 5.44 on 1 df) compared to the normal model, even though the asymptotic theory can not be trusted due to the small sample size.

We first examine the hyperbolic regression fit in Figure 5.3 and compare it with two other commonly used robust linear regression models briefly discussed at the beginning of this chapter. The result of fitting the skew-normal model is as below:

```
SNModel1 <- selm(stack ~ airflow + temperature + acid,
                 family = "SN", data = stackloss)
summary(SNModel1)
Call: selm(formula = stack ~ airflow + temperature + acid,
    family = "SN", data = stackloss)
Number of observations: 21
Family: SN
Estimation method: MLE
Log-likelihood: -52.1665
Parameter type: CP

CP residuals:
```

```
     Min        1Q    Median        3Q       Max
-7.76985  -1.73646  -0.09243   2.52069   5.77425

Regression coefficients
                 estimate  std.err  z-ratio Pr{>|z|}
(Intercept.CP)  -43.2057  14.0553  -3.0740    0.002
airflow           0.7550   0.1316   5.7368    0.000
temperature       1.2446   0.3382   3.6797    0.000
acid             -0.1293   0.1555  -0.8318    0.406

Parameters of the SEC random component
        estimate std.err
s.d.      2.9292   0.480
gamma1   -0.3192   0.712
```

The result of fitting the skew-t model is as below:

```
STModel1 <- selm(stack ~ airflow + temperature + acid,
                 family = "ST", data = stackloss)
summary(STModel1, param = "DP")
Call: selm(formula = stack ~ airflow + temperature + acid,
    family = "ST", data = stackloss)
Number of observations: 21
Family: ST
Estimation method: MLE
Log-likelihood: -49.48134
Parameter type: DP

DP residuals:
    Min       1Q   Median       3Q      Max
-9.3351  -0.7363   0.3578   0.9613   8.3121

Regression coefficients
                 estimate   std.err   z-ratio Pr{>|z|}
(Intercept.DP)  -38.05630   3.79218 -10.03546    0.000
airflow           0.85841   0.05779  14.85283    0.000
temperature       0.47576   0.14895   3.19399    0.001
acid             -0.07926   0.05765  -1.37484    0.169

Parameters of the SEC random component
       estimate std.err
omega    0.9824   0.446
alpha    0.2823   0.702
nu       1.1367   0.531
```

Figure 5.4 illustrates the result of fitting the skew-normal model while Figure 5.5 illustrates the result of fitting the skew-t model.

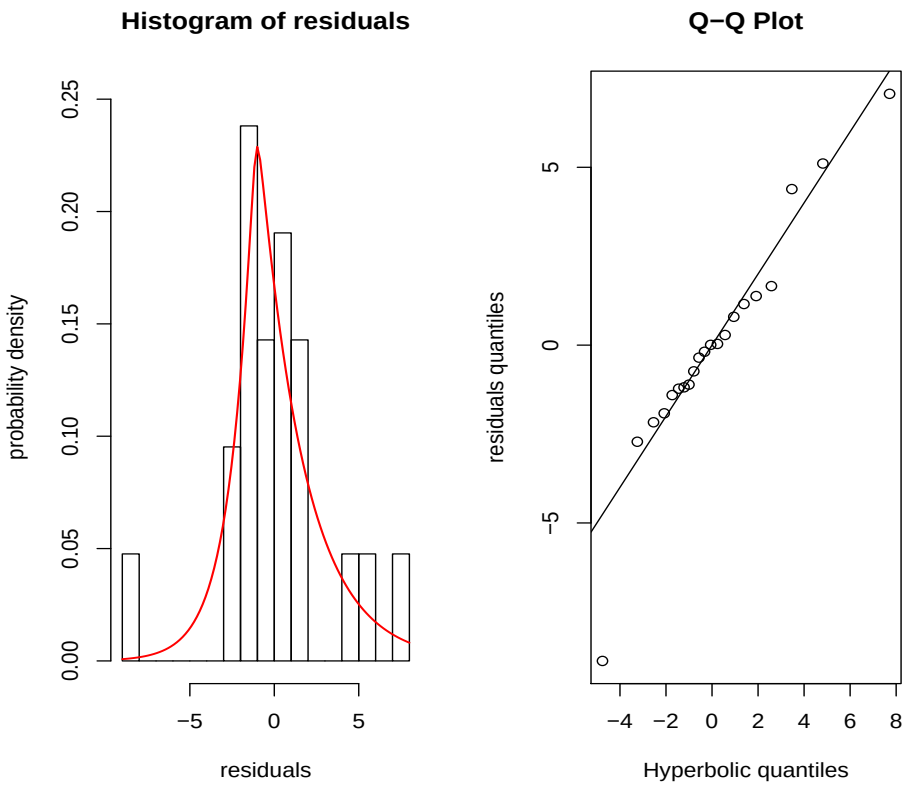

Figure 5.3: Stack-Loss Residual Plot and Q-Q Plot of Hyperbolic Model

The residual plot of Figure 5.4 shows that the skew-normal model is not an appropriate model for stack loss data as the skew-normal distribution density curve does not capture the spike in the residuals. The residual plots of Figure 5.3 and Figure 5.5 reveal that the hyperbolic and skew-t models provide much more satisfactory fitting results

The Q-Q plots of Figure 5.3 and 5.5 illustrate that the fitting results are fairly comparable between the hyperbolic model and skew-t model, although the hyperbolic model appears to have a slightly better fit.

We next compare the log-likelihood of the hyperbolic model with the t models in [76], i.e. the t model with estimated $\hat{\nu} = 1.1$, as well as the skew-normal and skew-t model.

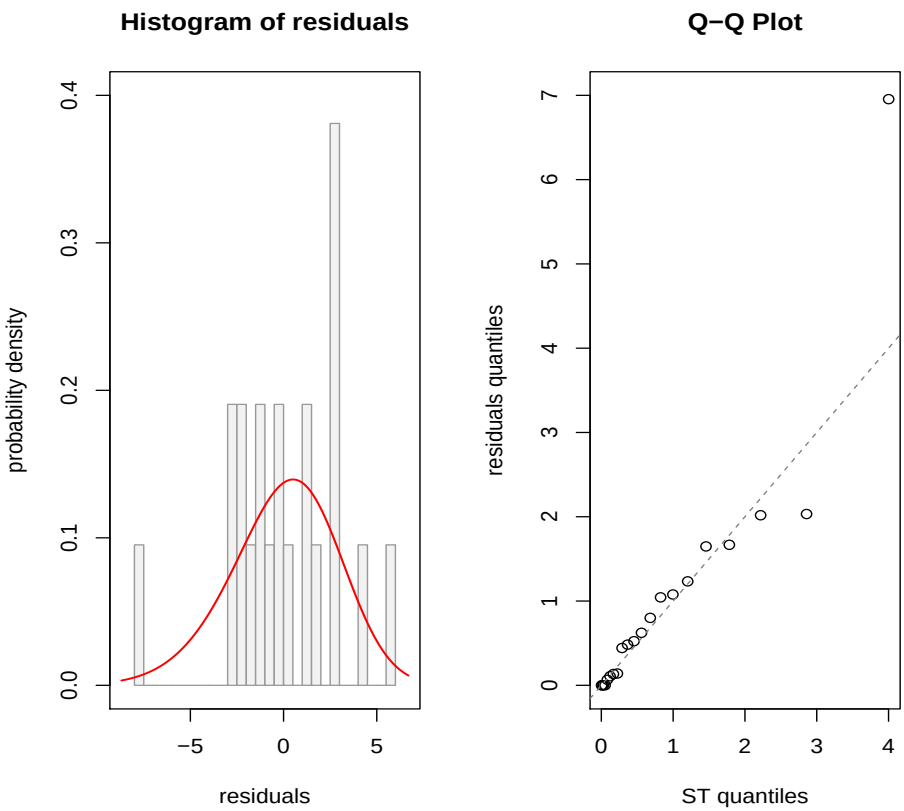

Figure 5.4: Stack-Loss Residual Plot and Q-Q Plot of Skew-Normal Model

Table 5.3: Summary Log-Likelihood of Hyperbolic Model, Skew-Normal Model, Skew-t Model and t Model

	Hyperbolic	Skew-Normal	Skew-t	t
Log-likelihood	−51.1	−52.2	−49.5	−30.3

From the result in Table 5.3, we can conclude that the performances of the hyperbolic model, the skew-normal model and the skew-t model are comparable, however the t model proposed in [76] appears to obtain a significantly better result. It is possible that the t distribution fits better here because it can have heavier tails than the hyperbolic.

5.2.2 Wisconsin Nursing Homes

The Wisconsin nursing facilities finance data has 10 attributes, shown in Table 5.4.

As Figure 5.6 indicates, the response variable `TPY` is fairly right skewed and exhibits a long tail, therefore the normal distribution assumption is not appropriate. There are several approaches for fitting a heavy tail regression model to this dataset that are discussed and

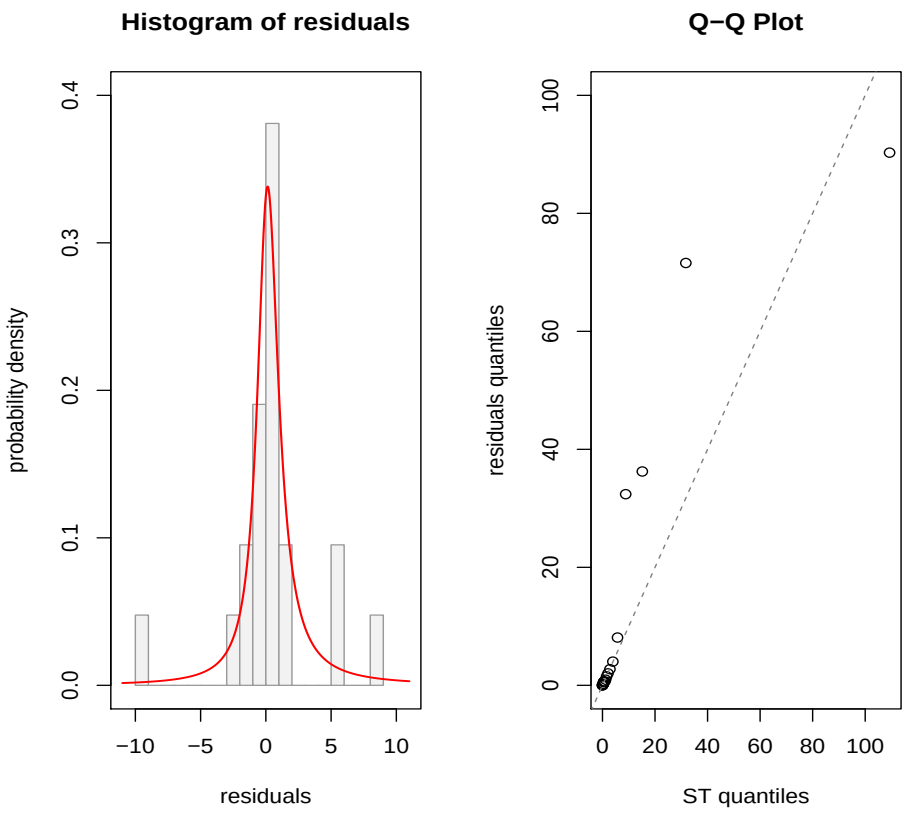

Figure 5.5: Stack-Loss Residual Plot and Q-Q Plot of Skew-t Model

Table 5.4: Wisconsin Nursing House Finance Data

Variable Name	Description
`TPY`	Total patient years
`SqrFoot`	Square footage of the nursing home
`Urban`	1 if urban, 0 if rural
`Pro`	1 if for profit, 0 for non-profit
`TaxExempt`	1 if tax-exempt
`SelfIns`	1 if self-funded for insurance
`MCert`	1 if Medicare certified
`NumBed`	Number of beds

compared in [46]. Those approaches include fitting generalized linear models directly to the data using the gamma distribution and the inverse Gaussian distribution. Both models use the logarithmic link function. From the result of these fitted generalized linear models, the variable ln(NumBed) is identified as an offset variable. [46] points out that the response variable TPY can be rescaled by the offset variable as an alternative modeling strategy. In addition, the modified accelerated failure time (AFT) model is proposed which is a log location-scale model. The response variable y is scaled as $\ln(y) = \mu + \theta \ln(y_0)$ where y_0 has a standard distribution,

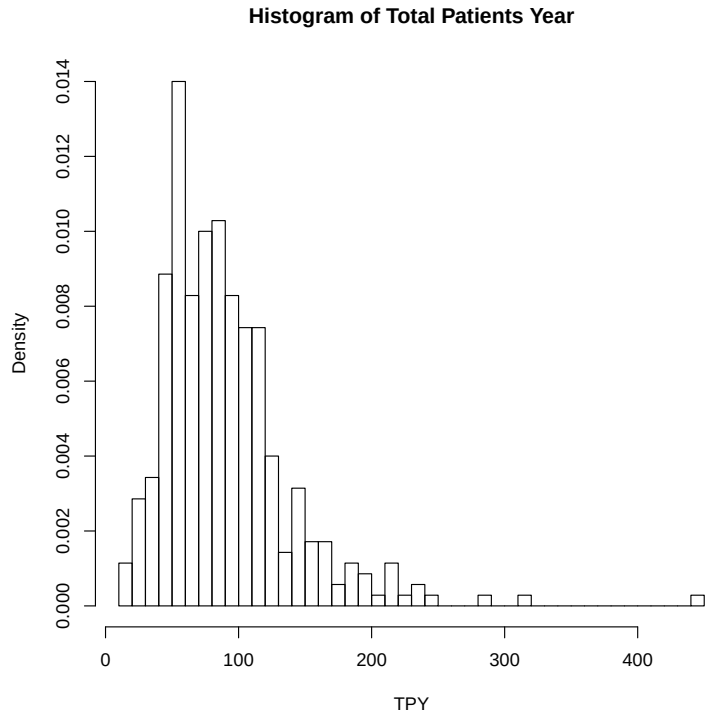

Figure 5.6: Histogram of total patient years

the Weibull, lognormal, and log-logistic distributions being commonly used. In [46], the gamma distribution and the ratio of two gamma distributions have been used in attempting to fit y_0. This is equivalent to fitting y with the generalized gamma distribution and the generalized beta of the second kind of distribution (GB2) respectively. This AFT model is applied to the transformed response variable defined as

$$\text{Occupancy Rate} = \frac{\text{TPY}}{\text{NumBed}} \times 100 \tag{5.19}$$

The annual occupancy rate, shown in Figure 5.7, is fairly left skewed and still exhibits a fairly long tail.

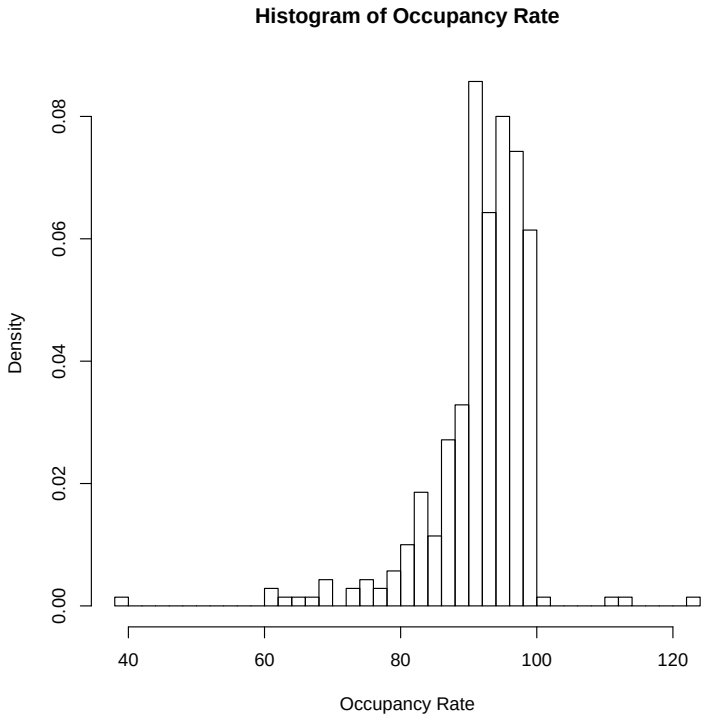

Figure 5.7: Histogram of Annual Occupancy Rate

Table 5.5: Goodness-of-fit Statistics of Gamma Model, Inverse Gaussian Model, Generalized Gamma Model and GB2 Model

	Gamma	Inverse Gaussian	Generalized Gamma	GB2
Log-likelihood	−1131.24	−1218.15	−1148.135	−1098.723
AIC	2280.47	2454.31	2316.270	2219.446
BIC	2315.17	2489.00	2319.296	2223.822

GB2 is recommended to be the best fitting model from the goodness-of-fit test. However none of the explanatory variables is significant and the Q-Q plot shows some discrepancies for smaller values of nursing homes total patient years ([46]). These shortcomings motivate us to fit the hyperbolic regression model to the data. We will only consider TPY as a response variable. This is because we do not use the logarithm link function in our model as the expected value of TPY is no longer proportional to NumBed. We then compare the result with not only the models in [46] but also the other two commonly used robust linear regression models briefly discussed at the beginning of this chapter. The first one assumes the residuals have a skew-t distribution and the second one assumes the residuals have a skew-normal distribution. The results of the hyperbolic regression fit are below.

```
Hmodel2 <- hyperblm(TPY ~ NumBed + SqrFoot +
                    Pro + TaxExempt + SelfIns + MCert + Urban)
summary.hyperblm(HModel2)

Call:
hyperblm(formula = TPY ~ NumBed + SqrFoot + Pro +
TaxExempt + SelfIns + MCert + Urban)

Data:      (Intercept) NumBed SqrFoot Pro TaxExempt SelfIns
              MCert Urban
Parameter estimates:
              Estimate Std. Error  t value  Pr(>|t|)
(Intercept) -4.1614496  1.3284844  -3.1325 0.0018844 **
NumBed       0.9505878  0.0053565 177.4631 < 2.2e-16 ***
SqrFoot      0.0283823  0.0079530   3.5687 0.0004104 ***
Pro          0.8147273  0.5506125   1.4797 0.1398888
TaxExempt    2.0240248  0.5867299   3.4497 0.0006321 ***
SelfIns     -0.2384328  0.5730197  -0.4161 0.6776010
MCert       -1.0940438  0.9683832  -1.1298 0.2593747
Urban       -0.2276561  0.5385584  -0.4227 0.6727720
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error distribution parameter estimates:
      Estimate Std. Error
mu     3.62913     0.6054
delta  0.43089     0.2013
alpha  0.25649     0.0208
beta  -0.10922     0.0238

Likelihood:          -1085.666
Method:              Nelder-Mead
Convergence code:    0
Iterations:          5
```

The explanatory variables `NumBed`, `SqrFoot` and `TaxExempt` are statistically significant. In [46], the gamma model has variables `NumBed` and `SqrFoot` as statistically significant whereas the inverse Gaussian model only has variable `NumBed` as significant. When the occupancy rate is considered as a response variable, the generalized gamma model has two significant variables and the GB2 has none. This is explained by Edward Frees as the occupancy rate being rescaled by `NumBed`, a very important explanatory variable ([46]). Consequently, the hyperbolic regression model appears to be more able to identify explanatory variables which are statistically significant.

To further assess the fit of the model, we explore the residual plots. Figure 5.8 shows the residuals from the hyperbolic regression model. The left-hand panel shows the histogram of residuals with the fitted density curve overlaid. The right-panel shows the Q-Q plot of residuals. The density curve captures the peaked-ness and skewness of the residual histogram which indicates that our assumption the error has a hyperbolic distribution is fairly reasonable. The Q-Q plot shows that the hyperbolic regression model fits reasonably well without any obvious discrepancies for the smaller values of nursing home occupancy, as occurred in GB2 ([46]).

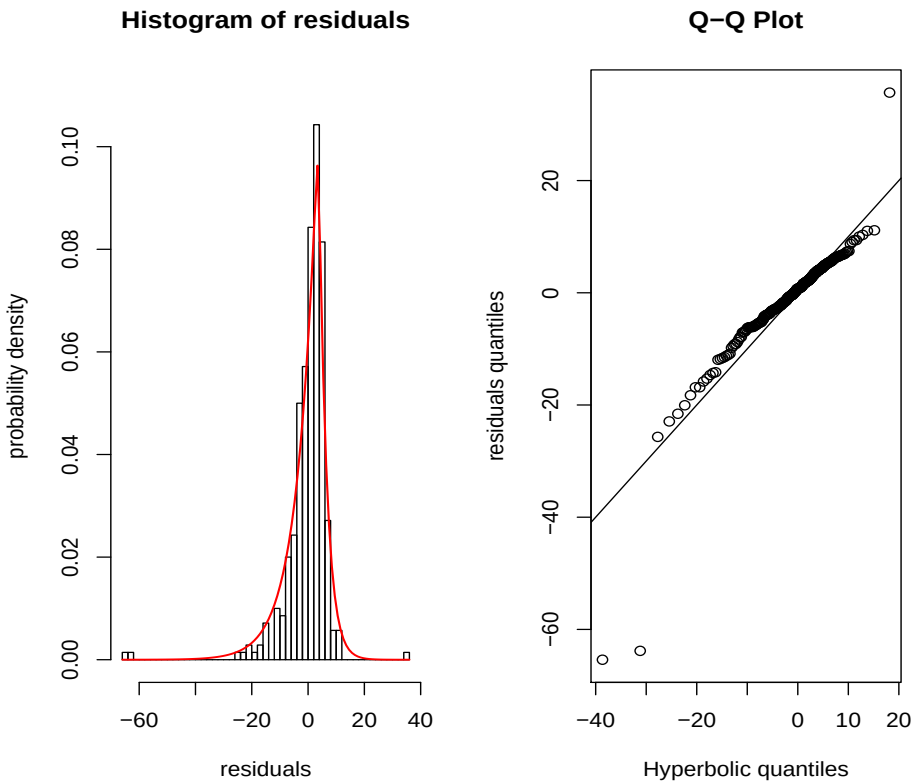

Figure 5.8: Total Patient Years Residual Plot and Q-Q Plot of Hyperbolic Model

As mentioned in the beginning of this Chapter, the alternative possible approaches are to fit the data with a linear model using a skew-normal distribution or a skew-t distribution. We apply the two approaches that are implemented in the **sn** package. Since neither of them use the logarithm link function, we will only consider TPY as a response variable again.

The result of fitting the skew-normal model is as below.

```
SNModel2 <- selm(tpy ~ numbed + sqrfoot + pro +
                 taxexempt + self + mcert + urban,
                 family = "SN")
summary(SNModel2)
```

```
Call: selm(formula = tpy ~ numbed + sqrfoot + pro +
    taxexempt + self + mcert + urban, family = "SN")
Number of observations: 350
Family: SN
Estimation method: MLE
Log-likelihood: -1155.291
Parameter type: CP

CP residuals:
    Min      1Q  Median      3Q     Max
-63.108  -1.714   1.897   4.715  31.418

Regression coefficients
               estimate  std.err  z-ratio Pr{>|z|}
(Intercept.CP) -1.91901  1.65947 -1.15640    0.248
numbed          0.91403  0.01214 75.26918    0.000
sqrfoot         0.07801  0.01882  4.14513    0.000
pro            -0.99799  1.34738 -0.74069    0.459
taxexempt      -0.41819  1.33994 -0.31209    0.755
self            0.41757  0.69263  0.60288    0.547
mcert          -0.92883  1.18338 -0.78489    0.433
urban          -1.09292  0.69412 -1.57455    0.115

Parameters of the SEC random component
       estimate std.err
s.d.     6.8356   0.273
gamma1  -0.6078   0.062
```

The result of fitting the skew-t model is as below.

```
STModel2 <- selm(tpy ~ numbed + sqrfoot + pro +
                 taxexempt + self + mcert + urban,
                 family = "ST")
summary(STModel2, param = "DP")
Call: selm(formula = tpy ~ numbed + sqrfoot + pro +
    taxexempt + self + mcert + urban, family = "ST")
Number of observations: 350
Family: ST
Estimation method: MLE
Log-likelihood: -1066.539
Parameter type: DP

DP residuals:
    Min      1Q  Median      3Q     Max
```

```
-71.573   -7.575   -3.697   -1.308   30.733

Regression coefficients
                 estimate   std.err   z-ratio Pr{>|z|}
(Intercept.DP)   0.632695  1.062947  0.595227    0.552
numbed           0.955995  0.009754 98.012556    0.000
sqrfoot          0.020540  0.013998  1.467410    0.142
pro              0.761488  0.928861  0.819809    0.412
taxexempt        2.152746  0.884676  2.433373    0.015
self            -0.179110  0.418299 -0.428188    0.669
mcert           -0.767969  0.689802 -1.113318    0.266
urban           -0.221257  0.419326 -0.527648    0.598

Parameters of the SEC random component
      estimate std.err
omega    4.991   0.524
alpha   -2.431   0.647
nu       2.583   0.413
```

Figure 5.9 indicates that the skew-normal distribution assumption is not appropriate as the fitted skew-normal distribution does not capture the spike in the histogram at all and there are some discrepancies for larger values of nursing homes total patient years. The skew-t distribution, Figure 5.10, on the other hand, offers a reasonable solution to TPY as a response variable.

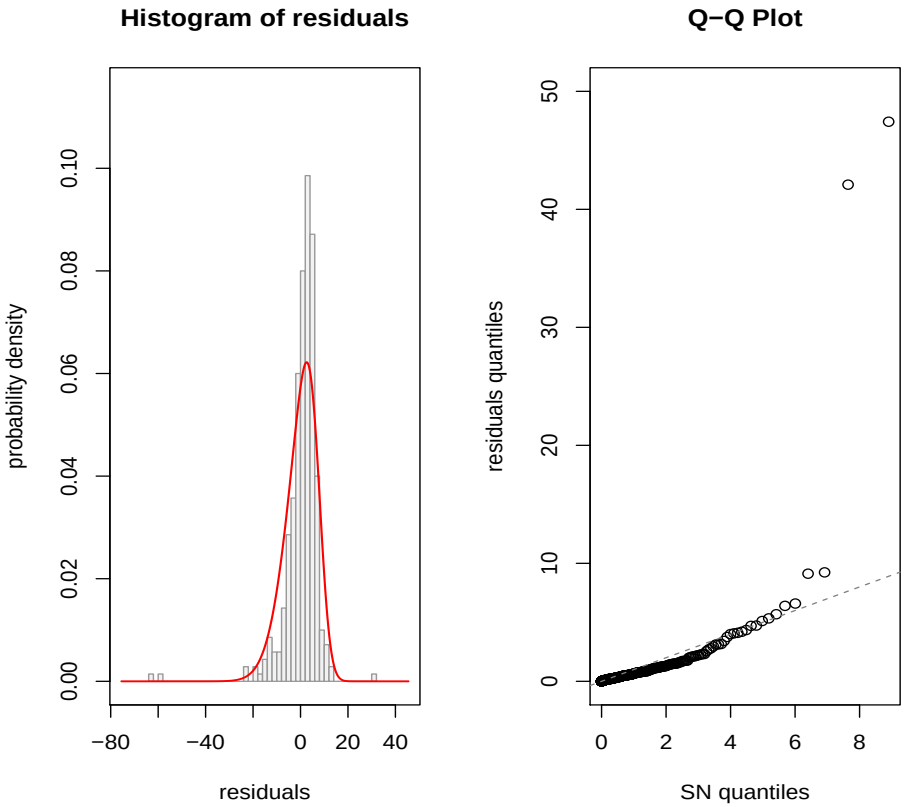

Figure 5.9: Total Patient Years Residual Plot and Q-Q Plot of Skew-Normal Model

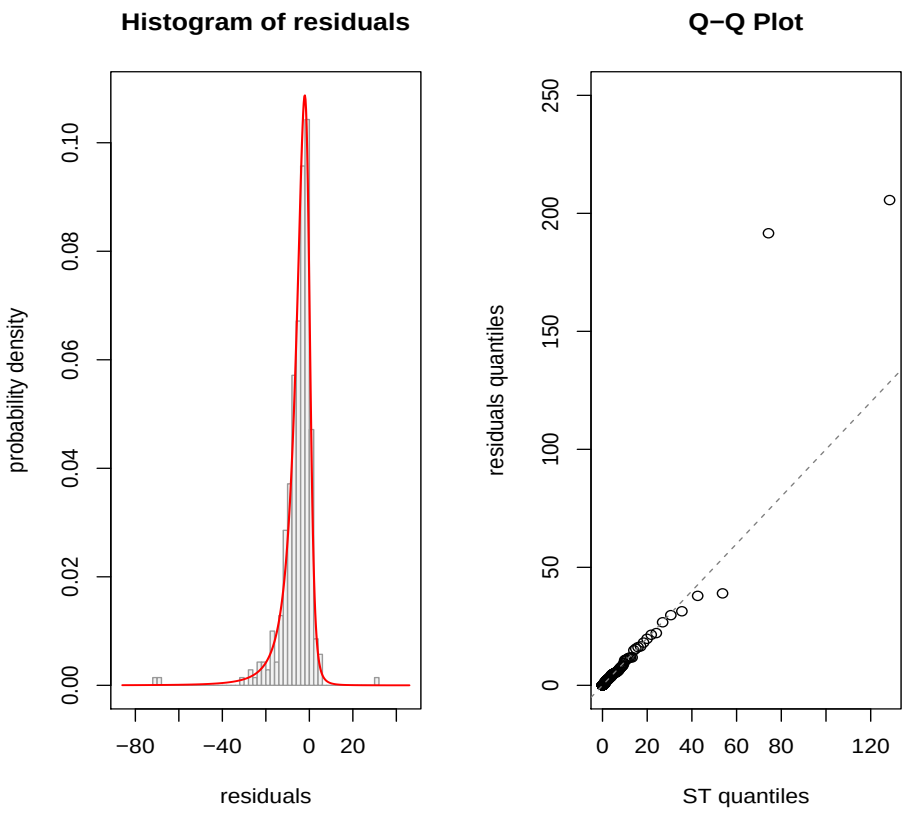

Figure 5.10: Total Patient Years Residual Plot and Q-Q Plot of Skew-t Model

To further compare the fit of the two models, Table 5.6 summarizes the goodness-of-fit statistics, residual sum of square (RSS) and $R^2 = 1 - \text{RSS/TSS}$ for each model.

Table 5.6: Summary Goodness-of-Fit Statistics of Hyperbolic Model, Skew-Normal Model and Skew-t Model

	Log-likelihood	AIC	BIC	RSS	R^2
Hyperbolic Distribution	−1085.666	2185.332	2212.318	20326.87	0.9759
Skew-Normal Distribution	−1155.308	2324.616	2351.602	36763.77	0.9563
Skew-t Distribution	−1066.539	2147.078	2174.064	30319.25	0.964

From Table 5.6, the skew-t distribution fits slightly better but is still comparable with the hyperbolic regression from goodness-of-fit statistic perspective. However the RSS of the hyperbolic regression model is significantly smaller than the RSS of the skew-t distribution. As a result, the hyperbolic regression model explains slightly more variation in the data (97.6% > 96.4%). For these reasons, we consider the hyperbolic regression as having comparable fit to skew-t regression.

5.2.3 Capital Asset Pricing Model

The CAPM, still widely used in applications such as estimating the cost of capital for firms and evaluating the performance of managed portfolio, ([45]) is defined as

$$E(R_i) = R_f + \beta_i[E(R_m) - R_f] \tag{5.20}$$

where $E(R_i)$ is the expected return on the capital asset and R_f is the risk-free rate of interest. The coefficient β_i, the so called market beta, is the sensitivity of the expected excess asset returns to the expected market returns and can be found using the regression. In the example, we will apply the CAPM to the Apple daily log returns, to estimate the market beta for Apple share prices. We take the S&P 500 as the market price and use three-month T-bills rates as risk free returns. The log returns are found by differencing the log closing prices, and the excess returns are the log-returns minus the T-bill rates.

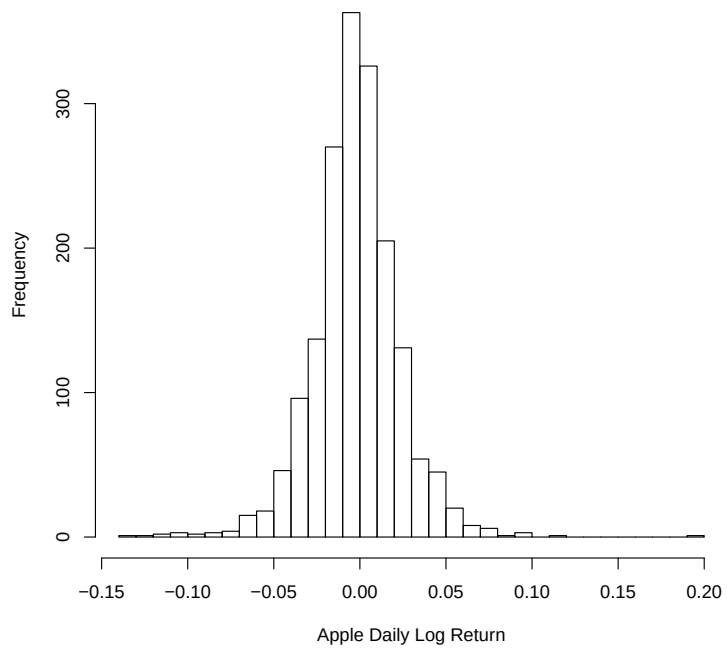

Figure 5.11: Histogram of Apple Daily Log Return

As Figure 5.11 shows, the Apple daily log return is roughly symmetric with a fairly large spike. This peaked-ness indicates our hyperbolic regression may be appropriate for this data. We again will compare the fitting result with the skew-normal and skew-t distribution models. The result of fitting the hyperbolic regression model is as below.

```
APPLEdf <- log(APPLE[2:length(APPLE)]) -
log(APPLE[1:(length(APPLE)-1)]) - Tbilldaily[-1]
SPdf <- log(SP[2:length(SP)]) - log(SP[1:(length(SP)-1)]) -
Tbilldaily[-1]
HModel3 <- hyperblm(APPLEdf ~ SPdf)
summary.hyperblm(HModel3)

Call:
hyperblm(formula = APPLEdf ~ SPdf)

Data:      (Intercept) SPdf
Parameter estimates:
               Estimate  Std. Error t value  Pr(>|t|)
(Intercept) -0.00184758  0.00051229 -3.6065  0.000319 ***
SPdf         1.10749379  0.02739605 40.4253 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error distribution parameter estimates:
        Estimate Std. Error
mu     0.0017864     0.0015
delta  0.0014672     0.0002
alpha 68.1913991     1.9698
beta  -2.1834686     5.5566

Likelihood:          4471.55
Method:              Nelder-Mead
Convergence code:    0
Iterations:          2
```

The intercept γ_0 is significant which contrasts with CAPM's assumption that $\gamma_0 = 0$. This indicates the Apple asset is mis-priced according to CAPM. The `SPdf` is significant as well, which means the market beta β_j is significant: it is less than the CAPM expectation. We next explore residual plots to assess the model further. Although the Q-Q plot in Figure 5.12 shows some discrepancies for small values of daily log returns, the fitted density curve closely follows the residual histogram. The fitting result seems satisfactory.

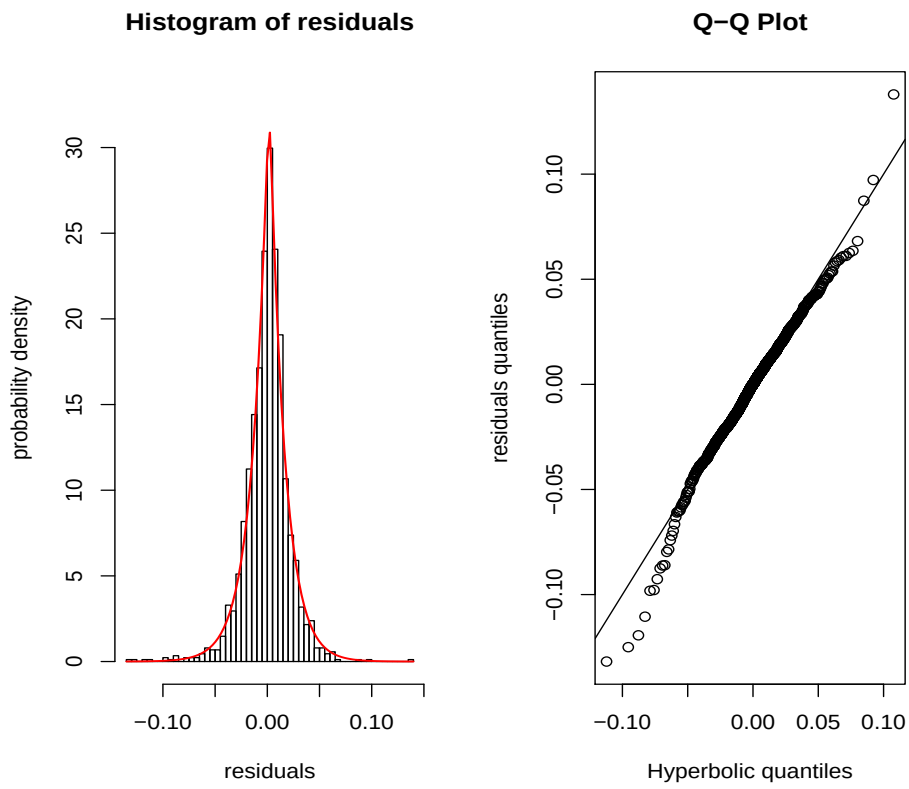

Figure 5.12: Apple Residual Plot and Q-Q Plot of Hyperbolic Model

After applying the skew-normal and skew-t distribution models to the data, we first present the fitting result of both models. Firstly the skew-normal model:

```
SNModel3 <- selm(APPLEdf ~ SPdf, family = "SN")
summary(SNModel3)
Call: selm(formula = APPLEdf ~ SPdf, family = "SN")
Number of observations: 1762
Family: SN
Estimation method: MLE
Log-likelihood: 4337.489
Parameter type: CP

CP residuals:
      Min        1Q    Median        3Q       Max
-0.131376 -0.010082  0.001306  0.010893  0.137501

Regression coefficients
                 estimate    std.err    z-ratio Pr{>|z|}
(Intercept.CP) -0.0020537  0.0004955 -4.1446933        0
```

```
SPdf            1.0781871   0.0359158 30.0198271        0

Parameters of the SEC random component
        estimate std.err
s.d.     0.02074   0.000
gamma1 -0.21968   0.037
```

Next, the result of fitting the skew-t model is as below.

```
STModel3 <- selm(APPLEdf ~ SPdf, family = "SN")
summary(STModel3, param = "DP")
Call: selm(formula = APPLEdf ~ SPdf, family = "ST")
Number of observations: 1762
Family: ST
Estimation method: MLE
Log-likelihood: 4472.004
Parameter type: DP

DP residuals:
      Min        1Q    Median        3Q       Max
-0.136869 -0.015241 -0.004012  0.005600  0.132646

Regression coefficients
                estimate   std.err   z-ratio Pr{>|z|}
(Intercept.DP)  0.003246  0.001424  2.279974    0.023
SPdf            1.099107  0.033392 32.914823    0.000

Parameters of the SEC random component
       estimate std.err
omega   0.01448   0.001
alpha  -0.35597   0.121
nu      3.40117   0.319
```

We then plotted the residual plots for those two models. The residual plots of Figure 5.13 and 5.14 show that neither of two fitted density curves successfully captures the peaked-ness of the data. The Q-Q plots of Figure 5.13 and 5.14 in addition show some obvious discrepancies for larger values of Apple daily log return. For these reasons, the hyperbolic model seems to perform better than the other two models.

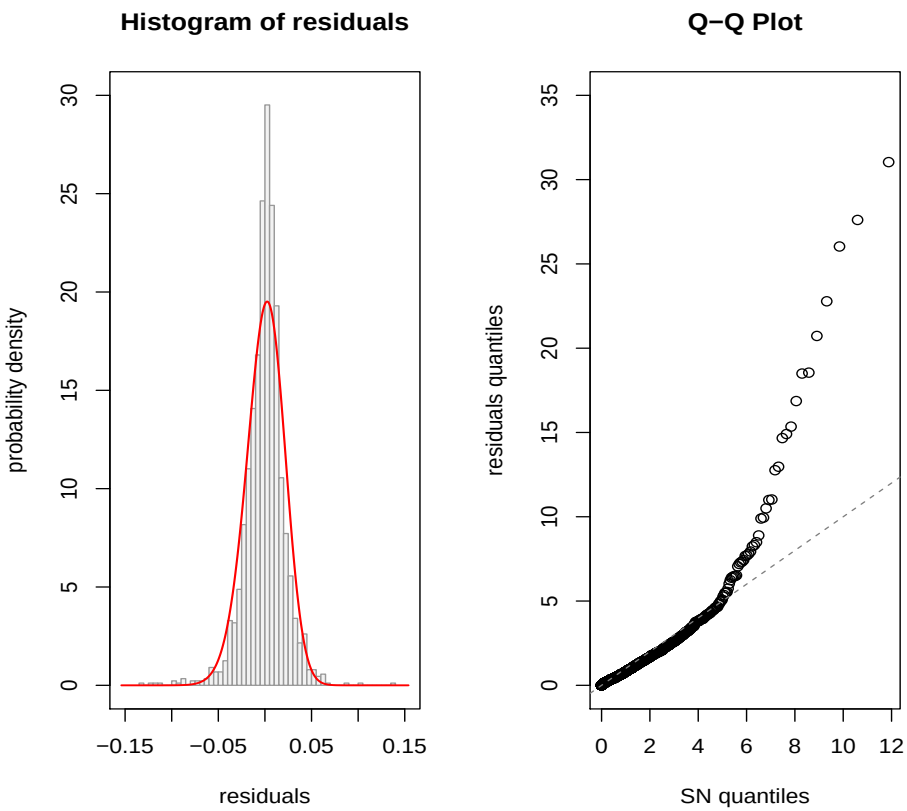

Figure 5.13: Apple Residual Plot and Q-Q Plot of Skew-Normal Model

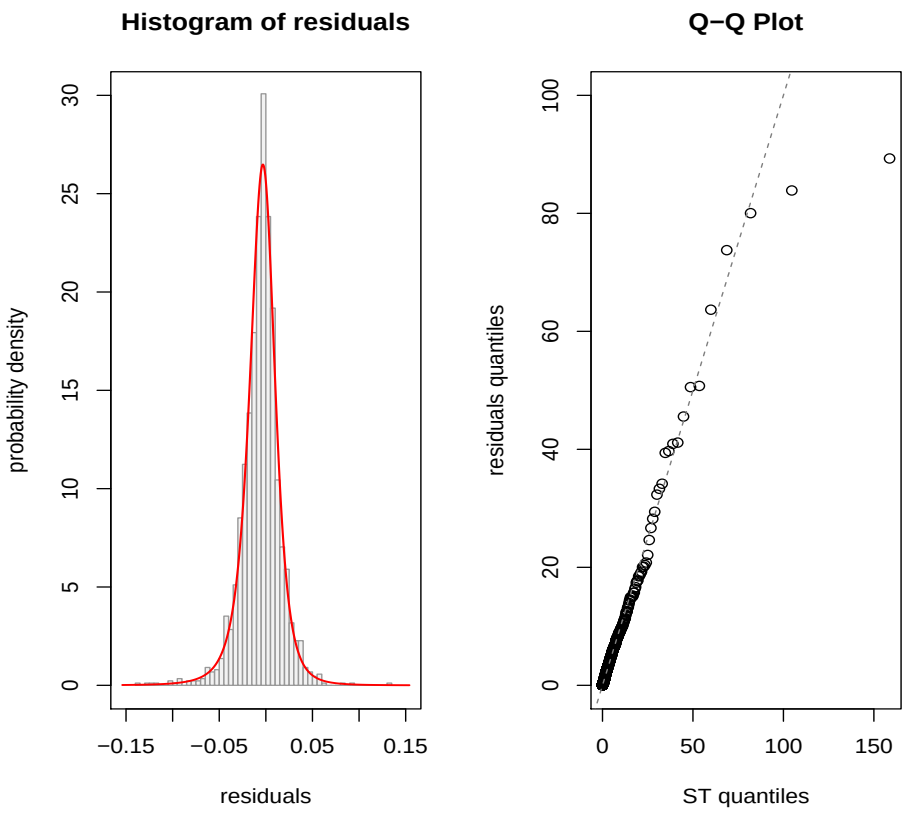

Figure 5.14: Apple Residuals Plot of and Q-Q Plot of Skew-t Distribution

To compare the fitting results further, goodness-of-fit statistics are summarized in the Table 5.7. The skew-t distribution again obtains similar AIC and BIC values to the hyperbolic regression. However the residual sum of square of the hyperbolic regression is smaller, therefore the model explains more of the variation in the data. In summarizing all the findings from the plots, we believe it is evidently seen that the hyperbolic regression model has the better fitting result than skew-normal regression model and has the slightly better fitting result as skew-t regression model.

Table 5.7: Summary Goodness-of-Fit Statistics of Hyperbolic Model, Skew-Normal Model and Skew-t Model

	Log-likelihood	AIC	BIC	RSS	R^2
Hyperbolic	4471.55	−8941.1	−8935.626	0.7706	0.3182
Skew-Normal	4337.487	−8672.974	−8667.5	1.2456	−0.1021
Skew-t	4472.004	−8942.008	−8936.534	0.8160	0.278

5.3 Conclusion

In this chapter, the hyperbolic regression model as an alternative approach for parametric robust regression. The examples in Section 5.2 illustrate the hyperbolic model provides a better fitting result to heavy tailed and skewed data than the skew-normal model and performs comparably with the skew-t model.

Apart from the usefulness of the model itself, the approach of implementing hyperbolic model can also be adapted for the NIG distribution as well. Recall that as discussed in Chapter 1, the NIG distribution is also a subclass of the GHyp distribution with $\lambda = -1/2$ and it appears more tractable than the hyperbolic distribution. However the approach may not suitable for the skew hyperbolic Student-t and GHyp distribution. The former unsuitability is because the skew hyperbolic Student-t distribution does not always have moments therefore the standardization procedure will not work while the latter distribution has five parameters which may cause the optimization procedure to be unstable.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The main purpose of my research was to investigate potential solutions for computational problems arising in the use of the GHyp distribution, the hyperbolic distribution and the skew hyperbolic Student-t distribution in statistical applications, in particular using R. The investigation includes a combination of reviewing/comparing the existing potential solutions and proposing and investigating new approaches summarized as follows:

Chapter 2

In this chapter, I investigated algorithms for general routine tasks for any univariate unimodal continuous distribution by means of numerical calculation, and modifications to an existing incomplete Bessel function approximation method to deal with overflow/underflow concerns.

Chapter 3

New algorithms for approximating the CDF and quantile function of the GHyp distribution and the CDF of the skew hyperbolic Student-t distribution were proposed and investigated in this chapter.

Chapter 4

Chapter 4 was mainly devoted to reviewing and comparing existing proposed random number generation approaches for the GIG distribution and the hyperbolic distribution.

Chapter 5

Although the basic idea of linear modeling with hyperbolic distribution errors is from [107], I proposed and investigated a new approach in Chapter 5 for performing hyperbolic linear regression that is superior to the approach used in [107]. In addition, hyperbolic linear regression using the proposed approach was shown to perform satisfactorily for real data.

In the thesis, the problems that have been addressed are:

- The current methods of approximating the GHyp distribution CDF and quantile function are inefficient, lack flexibility and are not particular accurate. To remedy these, I
  - Improved the CDF approximation approach which breaks the distribution curve into 2 regions instead of 8 to perform the required numerical integration. The CDF is no longer set to 1 or 0 for any density $< 10^{-5}$ in the upper or lower tail. Instead, the new and improved CDF approximation approach calculates the exact value.
  - Proposed a quantile function approximation algorithm which consists of two approaches, numerical integration and spline interpolation, to increase accuracy, flexibility and efficiency. Using the spline interpolation approach, the time taken to approximate quantiles is significantly shortened compared to the numerical integration approach. However using the numerical integration approach, the approximation result is more accurate.

- The existing approach to approximate the skew hyperbolic Student-t distribution CDF fails badly when the quantiles are in the distribution's polynomial tail. I proposed a new approach which transforms the density function using the split-t transformation prior to approximating it numerically. This new approach was shown to perform reasonably well for most regions of the distribution including the polynomial tail, even if the required distribution is extremely skewed, i.e. $\nu = 0.5$.

- The performance of the ratio of uniforms with shifted mode approach to generate random variates under the GIG distribution declines substantially when $\beta \to 0$ and $\lambda \to 0$. By reviewing and implementing 6 other approaches, including 3 for all parameter spaces and 3 for parameters with restricted domain, I concluded the best approach to generating observations from the GIG distribution is to use the ratio of uniforms with shifted mode approach when $\lambda \geq 1$ or $\beta > \min[1/2, 2/(3\sqrt{1-\lambda})]$ and Hörmann's rejection sampling method when $0 < \lambda < 1$ and $\beta \leq \min[1/2, 2/(3\sqrt{1-\lambda})]$.

  I then concluded mixing the GIG distributed random variate generated by this approach and normally distributed random variates performs best for the GHyp distribution.

  For the hyperbolic distribution, I compared mixing the GIG distributed random variates generated by the ratio of uniforms with shifted mode approach and normally distributed random variates and 2 other approaches, namely the TDR approach with or without squeeze function. From the comparison, I concluded that mixing the GIG distributed random variates generated by the ratio of uniforms with shifted mode approach and normally distributed random variates to generate the hyperbolic distribution is superior to the other methods.

- Linear modeling with hyperbolic distribution errors provides an alternative solution to robust regression for skewed, heavy tailed data. However the published fitting function `hyperbFit` is not reliable when fitting models with more than two explanatory variables. I proposed the use of a two stage alternating optimization method to optimize the likelihood function. This approach stabilized the fitting process and the improved function `hyperblm` can be applied to model real data as well as simulated data. By comparing hyperbolic linear regression with other robust modeling approaches, I concluded that hyperbolic linear regression is comparable to linear regression with skew-t distribution errors, and superior to other approaches I examined.

Apart from the valuable results from my research, there are some areas that require further investigation.

- The split-t transformation proposed to approximate the skew hyperbolic Student-t distribution CDF can compute a probability as small as 10^{-4}. There still remains the problem of tail probability calculation in the extreme tails of this distribution that requires further research.

- Confidence intervals for the coefficients in linear hyperbolic regression are calculated by either the bootstrap method or computing the nearest symmetric positive semidefinite Hessian matrix as the Hessian matrix is not always positive definite. Further research, such as profile log likelihood, is required to remedy this as the bootstrap method is not efficient.

- As mentioned in Chapter 5, the approach used for implementing hyperbolic linear regression can also be adapted for the NIG distribution.

APPENDIX A

APPENDIX: SOURCE FUNCTIONS

A.1 Utility Functions

A.1.1 moranTest Function

```
moranTest <- function(x, densFn, alpha, param = NULL, ...){

  ## Purpose: The function implements a goodness fit test
  ##          using Moran's log spacings statistic.
  ##
  ## ======================================================
  ## Arguments: x, Vector of random generated numbers under
  ##               tested distribution
  ##            densFn, The root name of the distribution
  ##                    to be tested
  ##            alpha, Quantile of the Chi-square distribution
  ##            param, A vector giving the parameter values for the
  ##                   distribution specified by \texttt{densFn}.
  ##                   If no \texttt{param} values
  ##                   are specified, then the default
  ##                   parameter values of each
  ##                   distribution are used instead.
  ##            ..., Additional arguments to allow
  ##                 specification of the
  ##                 parameters of the distribution
  ##                 other than specified by param.


    if (missing(densFn) | !(is.function(densFn) | is.character(densFn)))
    stop("'densFn' must be supplied as a function or name")
```

```
CALL <- match.call()

pfun <- match.fun(paste("p", densFn, sep = ""))

if(is.null(param)){
    y <- sort(pfun(x, ...))
    l <- list(...)
    k <- length(l)
}
else{
    y <- sort(pfun(x, param = param, ...))
    k <- length(param)
}


## Calculate M
M <- sum(log(diff(unique(y))), na.rm=TRUE)
if (y[length(y)] == 1) {
    M <- -(M + log(y[1]))
}
if (y[length(y)] != 1) {
    M <- -(M + log(y[1]) + log(1 - y[length(y)]))
}
if (M == Inf) {
    M <- 0
}


## Calculate T
n <- length(x)
m <- n + 1


ym <- m*(log(m) - digamma(1)) - 1/2 - 1/(12*m)
sm <- m*(pi^2 / 6 - 1) - 1/2 - 1/(6*m)

C1 <- ym - sm^0.5*(0.5*n)^0.5
C2 <- sm^0.5*(2*n)^-0.5

if (M == 0) {
    T <- 0
}
else {
    T <- (M + k/2 - C1)/C2
}
```

```
    ## Goodness-of-fit Test
    test <- T > qchisq(alpha, df=length(x))
    return(test)

}
```

A.1.2 distStepSize Function

```
distStepSize <- function(densFn, dist,
                         param = NULL, side = c("right","left"), ...){


  ## Purpose: Determine step size for calculating the range
  ##          of a unimodal distribution
  ## ========================================================
  ## Arguments: densFn, The name of the density function for which
  ##                    the step size needs to be calculated.
  ##            dist, Current distance value, for skew hyperbolic
  ##                  distribution only.
  ##            param, parameter vector of the distribution
  ##            side, "right" for a step to the right,
  ##                  "left" for a step to the right.
  ##            ..., Passes arguments in particular the
  ##                 parameters of the distribution to
  ##                 random sample generation function.


    set.seed(123)
    rfun <- match.fun(paste("r", densFn, sep = ""))
    side <- match.arg(side)
  # Generate a random sample
    n <- 50
    if(is.null(param)){
        sample <-  rfun(n, ...)
    } else {
        sample <- rfun(n, param = param)
    }
  # Approximated the empirical median
    mid <- median(sample)

  # Special arrangement of skew hyperbolic
  # Student's t distribution
    if (densFn == "skewhyp"){
        l <- list(...)
        delta <- ifelse(is.null(param), l$delta, param[2])
        nu <- ifelse(is.null(param), l$nu, param[4])
```

```
        beta <- ifelse(is.null(param), l$beta, param[3])
        if (beta > 0){
        step <- ifelse(side == "left", delta,
                       delta*abs(beta)*(nu*dist)^(-2/nu))
    }
    if (beta < 0){
        step <- ifelse(side == "right", delta,
                       delta*abs(beta)*(nu*dist)^(-2/nu))
    }
    if (isTRUE(all.equal(beta, 0))){
        step <- exp(dist/nu)
    }
        step <- c(step, mid)
    } else{
        quans <- as.vector(quantile(sample, probs = c(0.25, 0.75)))
        step <- ifelse(side == "left", mid - quans[1],
                       quans[2] - mid)
        step <- c(step, mid)
    }
    return(step)
}
```

A.1.3 distMode Function

```
distMode <- function(densFn, param = NULL, ...){


  ## Purpose: Approximate the mode of a unimodal distribution
  ## ========================================================
  ## Arguments: densFn, The name of the density function
  ##                     for which the mode needs to be calculated.
  ##            param, parameter vector of the distribution
  ##            ..., Passes arguments to optimize function.


    dfun <- match.fun(paste("d", densFn, sep = ""))
    if(densFn == "skewhyp"){
        l <- list(...)
        delta <- ifelse(is.null(param), l$delta, param[2])
    } else delta <- 0
    median <- distStepSize(densFn, param = param,
                           dist = delta, side = "left", ...)[2]
    if(is.null(param)){
        modefun <- function(x){
            log(dfun(x, ...))
        }
```

```
} else {
    modefun <- function(x){
        log(dfun(x, param = param))
    }
}
stepHigh <- distStepSize(densFn, param = param,
                         dist = delta, side = "right", ...)[1]
xHigh <- median + stepHigh
if (densFn == "skewhyp"){
    if(is.null(param)){
        while(dfun(xHigh, ...) > dfun(median, ...)){
            xHigh <- xHigh +
                distStepSize(densFn, param = param,
                             dist = delta, side = "right", ...)[1]
        }
    } else {
        while(dfun(xHigh, param = param) > dfun(median, param = param)){
            xHigh <- xHigh +
                distStepSize(densFn, param = param,
                             dist = delta, side = "right", ...)[1]
        }
    }
} else {
    if(is.null(param)){
        while(dfun(xHigh, ...) > dfun(median, ...)){
            xHigh <- xHigh + stepHigh
        }
    } else {
        while(dfun(xHigh, param = param) > dfun(median, param = param)){
            xHigh <- xHigh + stepHigh
        }
    }
}
stepLow <-  distStepSize(densFn, param = param,
                         dist = delta, side = "left", ...)[1]
xLow <- median - stepLow
if (densFn == "skewhyp"){
    if(is.null(param)){
        while(dfun(xLow, ...) > dfun(median, ...)){
            xLow <- xLow -
                distStepSize(densFn, param = param,
                             dist = delta, side = "left", ...)[1]
        }
    } else {
        while(dfun(xLow, param = param) > dfun(median, param = param)){
            xLow <- xLow -
                distStepSize(densFn, param = param,
```

```
                                dist = delta, side = "left", ...)[1]
            }
        }
    } else {
        if(is.null(param)){
            while(dfun(xLow, ...) > dfun(median, ...)){
                xLow <- xLow - stepLow
            }
        } else {
            while(dfun(xLow, param = param) > dfun(median, param = param)){
                xLow <- xLow - stepLow
            }
        }
    }
    range <- c(xLow, xHigh)
    optResult <- optimize(f = modefun, interval = range,
                          maximum = TRUE)
    mode <- optResult$maximum
    mode
}
```

A.1.4 pDist Function

```
pDist <- function(densFn = "norm", q, param = NULL,
                  subdivisions = 100, lower.tail = TRUE,
                  log.p = FALSE,
                  intTol = .Machine$double.eps^0.25,
                  valueOnly = TRUE, ...){

  ## Purpose: Implement general distribution function evaluating
  ##          approach
  ## ========================================================
  ## Arguments: densFn, The root name of the distribution
  ##                    to be tested
  ##            q, Vector of quantiles
  ##            param, parameter vector of the distribution
  ##            subdivision, The maximum number of subintervals
  ##                         of Integrate function
  ##            lower.tail, If lower.tail = TRUE, the cumulative
  ##                        density is taken from the lower tail
  ##            intTol, integrate function accuracy requested
  ##            valueOnly, If valueOnly = TRUE calls to pDist
  ##                       only return the value obtained for the
  ##                       integral. If valueOnly = FALSE an
  ##                       estimate of the accuracy of the
  ##                       numerical integration is also
```

```
##                        returned
##              ..., Passes additional arguments to integrate,
##                   distMode or distCalcRange functions.
##                   In particular, the parameters of
##                   the distribution


  CALL <- match.call()
  dfun <- match.fun(paste("d", densFn, sep = ""))
  mode <- distMode(densFn, param = param, ...)
  ## match the density function for different distribution
  qLess <- which((q <= mode)&(is.finite(q)))
  ## when q is less than mode
  qGreater <- which((q > mode)&(is.finite(q)))
  ## when q is greater than mode
  prob <- rep(NA, length(q))
  err <- rep(NA, length(q))
  prob[q == -Inf] <- 0
  prob[q == Inf] <- 0
  err[q %in% c(-Inf, Inf)] <- 0

  ## Integrate density from Inf / -Inf to the mode
  for (i in qLess){
      if(is.null(param)){
          intRes <- integrate(dfun, -Inf, q[i],
                              subdivisions = subdivisions,
                              rel.tol = intTol, ...)
      } else {
          intRes <- integrate(dfun, -Inf, q[i], param = param,
                              subdivisions = subdivisions,
                              rel.tol = intTol, ...)
      }
      prob[i] <- intRes$value
      err[i] <- intRes$abs.error
  }

  for (i in qGreater){
      if(is.null(param)){
          intRes <- integrate(dfun, q[i], Inf,
                              subdivisions = subdivisions,
                              rel.tol = intTol, ...)
      } else {
          intRes <- integrate(dfun, q[i], Inf, param = param,
                              subdivisions = subdivisions,
                              rel.tol = intTol, ...)
      }
      prob[i] <- intRes$value
```

```
        err[i] <- intRes$abs.error
    }

    if (lower.tail == TRUE)
   {
       prob[q > mode] <- 1 - prob[q > mode]
   }
   else
   {
       prob[q <= mode] <- 1 - prob[q <= mode]
   }

    if(log.p == TRUE)
      {
        prob = log(prob)
      }

    # Return Value:
    ifelse(valueOnly, return(prob),
           return(list(value = prob, error = err)))

}
```

A.1.5 qDist Function

```
qDist <- function(densFn = "norm", p, param = NULL,
                  lower.tail = TRUE, log.p = FALSE,
                  method = "spline",  nInterpol = 501,
                  uniTol = .Machine$double.eps^0.25, subdivisions = 100,
                  intTol = uniTol, ...){


  ## Purpose: Implement general quantile function evaluating
  ##          approach
  ## ========================================================
  ## Arguments: densFn, The root name of the distribution
  ##                    to be tested
  ##            p, Vector of probablities
  ##            param, parameter vector of the distribution
  ##            lower.tail, If lower.tail = TRUE, the cumulative
  ##                        density is taken from the lower tail
  ##            method, If "spline" quantiles are found from a
  ##                    spline approximation to the distribution
  ##                    function. If "integrate", the
  ##                    distribution function used is always
  ##                    obtained by integration
```

```
##            nInterpol, Number of points used in qDist for
##                       cubic spline interpolation of the
##                       distribution function
##            uniTol, uniroot function accuracy requested
##            valueOnly, If valueOnly = TRUE calls to pDist
##                       only return the value obtained for the
##                       integral. If valueOnly = FALSE an
##                       estimate of the accuracy of the
##                       numerical integration is also
##                       returned
##            subdivision, The maximum number of subintervals
##                         of Integrate function
##            intTol, integrate function accuracy requested
##            ..., Passes additional arguments to integrate,
##                 distMode or distCalcRange Functions.
##                 In particular, the parameters of
##                 the distribution


  CALL <- match.call()
  mode <- distMode(densFn, param = param, ...)
  pMode <- pDist(densFn, q = mode, param = param, intTol = intTol, ...)
  if(lower.tail == FALSE){
    p = 1 - p
  }
  if(log.p == TRUE){
    p = exp(p)
  }
  quant <- rep(NA, length(p))
  invalid <- which((p < 0) | (p > 1))
  pFinite <- which((p > 0) & (p < 1))
  if(densFn == "skewhyp"){
      l <- list(...)
      delta <- ifelse(is.null(param), l$delta, param[2])
  } else delta <- 0

  xRange <- distCalcRange(densFn, param = param, tol = 10^(-5), ...)

  if (method == "integrate"){
      less <- which((p <= pMode) & (p >.Machine$double.eps^7.5))
      quant <- ifelse(p <= .Machine$double.eps^5, -Inf, quant)
      if (length(less) > 0){
          pLow <- min(p[less])
          step <- distStepSize(densFn, param = param,
                               dist = delta, side = "left", ...)[1]
          xLow <- mode - step
          if (densFn == "skewhyp"){
```

```
            while(pDist(densFn, xLow,
                        param = param, intTol = intTol, ...) >= pLow)
            {
                xLow <- xLow -
                    distStepSize(densFn, param = param,
                                 dist = delta, side = "left", ...)[1]
            }
        } else {
            while(pDist(densFn, xLow,
                        param = param, intTol = intTol, ...) >= pLow)
            {
                xLow <- xLow - step
            }
        }
        xRange <- c(xLow, mode)
        zeroFn <- function(x, p){
            return(pDist(densFn, x, param = param,
                         intTol = intTol, ...) - p)
        }
        for (i in less)
        {
            quant[i] <- uniroot(zeroFn, p = p[i],
                                interval = xRange, tol = uniTol)$root
        }
    }

    greater <- which ((p > pMode) & (p < (1 - .Machine$double.eps^7.5)))
    p[greater] <- 1 - p[greater]
    quant <- ifelse(p >= (1 - .Machine$double.eps^5), Inf, quant)
    if (length(greater) > 0){
        pHigh <- min(p[greater])
        step <-  distStepSize(densFn, param = param,
                              dist = delta, side = "right", ...)[1]
        xHigh <- mode + step
        if (densFn == "skewhyp"){
            while (pDist(densFn, xHigh, param = param, intTol = intTol,
                         lower.tail = FALSE, ...) >= pHigh)
            {
                xHigh <- xHigh +
                    distStepSize(densFn, param = param,
                                 dist = delta, side = "left", ...)[1]
            }
        } else {
            while (pDist(densFn, xHigh, param = param, intTol = intTol,
                         lower.tail = FALSE, ...) >= pHigh)
            {
                xHigh <- xHigh + step
```

```
            }
        }
        xRange <- c(mode, xHigh)
        zeroFn <- function(x, p){
            return(pDist(densFn, x, param = param, intTol = intTol,
                         lower.tail = FALSE, ...) - p)
        }
        for (i in greater)
        {
            quant[i] <- uniroot(zeroFn, p = p[i],
                                interval = xRange, tol = uniTol)$root
        }
    }
} else if (method == "spline"){
    inRange <- which((p > pDist(densFn, xRange[1],
                                param = param, intTol = intTol, ...)) &
                     (p < pDist(densFn, xRange[2],
                                param = param, intTol = intTol, ...)))
    small <- which((p <= pDist(densFn, xRange[1],
                    param = param, intTol = intTol, ...)) & (p >= 0))
    large <- which((p >= pDist(densFn, xRange[2],
                    param = param, intTol = intTol, ...)) & (p <= 1))
    extreme <- c(small, large)
    xVals <- seq(xRange[1], xRange[2], length.out = nInterpol)
    yVals <- pDist(densFn, xVals, param = param,
                   subdivisions = subdivisions, intTol = intTol, ...)
    splineFit <- splinefun(xVals, yVals)
    zeroFn <- function(x, p){
        return(splineFit(x) - p)
    }

    for (i in inRange){
        quant[i] <- uniroot(zeroFn, p = p[i],
                            interval = xRange, tol = uniTol)$root
    }

    if (length(extreme) > 0){
        quant[extreme] <- qDist(densFn, p[extreme], param = param,
                                log.p = log.p,
                                lower.tail = lower.tail,
                                method = "integrate",
                                nInterpol = nInterpol, uniTol = uniTol,
                                subdivisions = subdivisions,
                                intTol = intTol, ...)
    }
}
```

```
    return(quant)
}
```

A.1.6 momIntegrated Function

```
momIntegrated <- function(densFn = "ghyp", param = NULL,
                          order,  about = 0, absolute = FALSE, ...){


  ## Purpose: Implement general distribution function evaluating
  ##          approach
  ## ========================================================
  ## Arguments: densFn, The root name of the distribution
  ##                     to be tested
  ##            param, parameter vector for skew hyperbolic
  ##                   distribution
  ##            order, The order of the moment or
  ##                   absolute moment to be calculated.
  ##            about, The point about which the moment is
  ##                   to be calculated
  ##            absolute, Whether absolute moments or ordinary
  ##                      moments are to be calculated
  ##            ..., Passes additional arguments to integrate.
  ##                 In particular, the parameters of
  ##                 the distribution


  if (missing(densFn) | !(is.function(densFn) | is.character(densFn)))
    stop("'densFn' must be supplied as a function or name")

  ## Set default integration limits
  low <- -Inf
  high <- Inf

  if (is.character(densFn)) {

    if (is.null(densFn))
      stop("unsupported distribution")
    if (densFn == "ghyp" | densFn == "hyperb" |
        densFn == "gig" | densFn == "vg")
    {
        if (!exists(paste("d",densFn,sep = ""), mode = "function"))
            stop("Relevant package must be loaded")
    }
```

```
if (densFn == "invgamma" | densFn == "inverse gamma"){
    l <- list(...)
    shape <- l$shape
    if(shape <= order)
        stop("Order must be less than shape parameter for inverse gamma")
    low <- 0
    dinvgamma <- function(x, shape, rate = 1, scale = 1/rate) {
    dens <- ifelse(x <= 0, 0,
                   (scale / x)^shape * exp(-scale / x) / (x * gamma(shape)))
    return(dens)
  }

  if (!absolute) {
    ddist <- function(x, order, about, ...) {
      (x - about)^order * dinvgamma(x, ...)
    }
  } else {
    ddist <- function(x, order, about, ...) {
      abs(x - about)^order * dinvgamma(x, ...)
    }
  }
} else {
    dfun <- match.fun(paste("d", densFn, sep = ""))
    if (densFn == "gamma"){
        l <- list(...)
        shape <- l$shape
        if(order <= -(shape))
            stop("Order must be greater than shape parameter for gamma")
        low <- 0
    }
    if (!absolute) {
        if (is.null(param)){
            ddist <- function(x, order, about, ...) {
                (x - about)^order * dfun(x, ...)
            }
        } else {
            ddist <- function(x, order, about, param) {
                (x - about)^order * dfun(x, param = param)
            }
        }
    } else {
            if (is.null(param)){
                ddist <- function(x, order, about, ...) {
                    abs(x - about)^order * dfun(x, ...)
                }
```

```
                } else {
                    ddist <- function(x, order, about, param) {
                        abs(x - about)^order * dfun(x, param = param)
                    }
                }
            }
        }
    }
    if (is.null(param)){
        mom <- integrate(ddist, low, high,
                         order = order, about = about,
                         subdivisions = 1000,
                         rel.tol = .Machine$double.eps^0.5, ...)[[1]]
    } else {
        mom <- integrate(ddist, low, high, param = param,
                         order = order, about = about,
                         subdivisions = 1000,
                         rel.tol = .Machine$double.eps^0.5)[[1]]
    }

  ## Return Value:
  return(mom)
}
```

A.1.7 `incompleteBesselKV` Function

```
incompleteBesselKV <- function(x, y, nu,
                               tol = (.Machine$double.eps)^(0.85),
                               nmax = 90){

  ## Purpose: Calculates the incomplete Bessel K function using
  ##          the modified algorithm which originally
  ##          proposed by Slavinsky and Safouhi (2009)
  ## ========================================================
  ## Arguments: x, Value of the first argument of the
  ##               incomplete Bessel K function
  ##            y, Value of the first argument of the
  ##               incomplete Bessel K function
  ##            nu, The order of the incomplete Bessel K
  ##                function
  ##            tol, The tolerance for the difference between
  ##                 successive approximations of the
  ##                 incomplete Bessel K function
  ##            nmax, The maximum order allowed for the
  ##                  approximation of the incomplete
  ##                  Bessel K function
```

```
  KNu <- besselK(2*sqrt(x*y), nu)
  m <- length(x)
  IBFOut <- .Fortran("incompleteBesselVK",
                     as.double(x),
                     as.double(y),
                     as.double(nu),
                     as.double(tol),
                     as.integer(m),
                     as.integer(nmax),
                     as.double(KNu),
                     IBF = double(m),
                     result = integer(m)
                     )

  Gp <- IBFOut$result
  IBF <- IBFOut$IBF
  if(length(Gp) == m)
    warning("Maximum G transformation order reached for all x and y")
  if(length(Gp) < m && length(Gp) > 0)
    warning("Maximum G transformation order reached for some x and y")
  Unreliable = cbind(Gp, x[Gp], y[Gp])
  colnames(Unreliable) = c("Index", "x", "y")
  return(list(Value = IBF, Unreliable = Unreliable))
}
```

```
subroutine incompleteBesselVK(x,y,m,nu,eps,nmax,KNu,ize,
     $                                IBF,result)
      integer nmax
      integer m,k,n,ize
      double precision nu,eps
      double precision x(1:m),y(1:m),KNu(1:m),IBF(1:m)
      integer result(1:m)
      double precision G(1:nmax,1:m)
      double precision GM(1:nmax),GN(0:nmax)
      double precision Am(0:nmax,0:nmax)
      double precision An(0:nmax,0:nmax)
      double precision Cnp(0:(nmax+1)*(nmax+2)/2)

      call combinatorial(nmax, Cnp)
      call SSFcoef(nmax,nu-1D0,Am)
      call SSFcoef(nmax,-nu-1D0,An)

      do k=1,m
      if(x(k).ge.y(k)) then
        call GDENOM(0,x(k),y(k),nu,An,nmax,Cnp,ize,GN)
```

```
      call GDENOM(1,x(k),y(k),nu,An,nmax,Cnp,ize,GN)
      call GNUM(1,x(k),y(k),nu,Am,nmax,Cnp,ize,GN,GM)
     G(1,k) = x(k)**nu*GM(1)/(GN(1)*(-x(1)*y(1)))
     do n=2,nmax
        call GDENOM(n,x(k),y(k),nu,An,nmax,Cnp,ize,GN)
        call GNUM(n,x(k),y(k),nu,Am,nmax,Cnp,ize,GN,GM)
        G(n,k) = x(k)**nu*GM(n)/(GN(n)*(-x(k)*y(k))**n)
        if(dabs((G(n,k)-G(n-1,k))/G(n-1,k)).lt.eps) then
          exit
          end if
       end do
     else if(y(k).gt.x(k)) then
       call GDENOM(0,y(k),x(k),-nu,Am,nmax,Cnp,ize,GN)
       call GDENOM(1,y(k),x(k),-nu,Am,nmax,Cnp,ize,GN)
       call GNUM(1,y(k),x(k),-nu,An,nmax,Cnp,ize,GN,GM)
       G(1,k) = y(k)**(-nu)*GM(1)/(GN(1)*(-x(1)*y(1)))
       do n=2,nmax
          call GDENOM(n,y(k),x(k),-nu,Am,nmax,Cnp,ize,GN)
          call GNUM(n,y(k),x(k),-nu,An,nmax,Cnp,ize,GN,GM)
          G(n,k) = y(k)**(-nu)*GM(n)/(GN(n)*(-x(k)*y(k))**n)
          if(dabs((G(n,k)-G(n-1,k))/G(n-1,k)).lt.eps) then
             G(n,k) = 2D0*(x(k)/y(k))**(nu/2D0)*KNu(k)-G(n,k)
             exit
          end if
       end do
     end if
   result(k) = n
   IBF(k) = G(n,k)
   end do
   return
   end


   subroutine SSFcoef(nmax,nu,A)
   implicit double precision(a-h,o-z)
   implicit integer (i-n)
   integer l,i,nmax
   double precision nu,A(0:nmax,0:nmax)
   A(0,0) = 1D0
   do l=1,nmax
     do i=1,l-1
     A(l,i) = (-nu+i+l-1D0)*A(l-1,i)+A(l-1,i-1)
     end do
     A(l,0) = (-nu+l-1D0)*A(l-1,0)
     A(l,l) = 1D0
   end do
   return
```

```
end

subroutine combinatorial(nu, Cnp)
implicit double precision (a-h, o-z)
implicit integer (i-n)

dimension Cnp(0:*)

do n=0, nu
   Cnp(n*(n+1)/2 + 0) = 1.0d0
   Cnp(n*(n+1)/2 + n) = 1.0d0
   do np=1, n-1
      Cnp(n*(n+1)/2+np) = Cnp(n*(n-1)/2+np-1)+Cnp(n*(n-1)/2+np)
   end do
end do
return
end

subroutine GNUM(n,x,y,nu,Am,nmax,Cnp,ize,GN,GM)
implicit double precision(a-h,o-z)
implicit integer (i-n)
integer n,nmax
double precision x,y,nu
double precision Am(0:nmax,0:nmax)
double precision Cnp(0:*),GM(1:nmax),GN(0:nmax)
GM(n) = 0D0
do ir=1,n
  terme=0D0
  do is=0,ir-1
   termepr = 0D0
     do i=0,is
      termepr = termepr+Am(is,i)*(-x)**i
     end do
terme = terme + termepr*Cnp(ir*(ir-1)/2+is)*(1D0/y)**is
  end do
  GM(n) = GM(n) + Cnp(n*(n+1)/2+ir)*(-1D0)**ir
$         * GN(n-ir)*terme
end do
if (ize .EQ. 1) then
GM(n) = GM(n)*(-x*y)**n*dexp(-x-y)/x**nu/y
else if (ize .EQ. 2) then
GM(n) = GM(n)*(-x*y)**n*dexp(-x-y+2*sqrt(x*y))/x**nu/y
 end if
return
end

subroutine GDENOM(n,x,y,nu,An,nmax,Cnp,ize,GN)
```

```
      implicit double precision(a-h,o-z)
      implicit integer (i-n)
      integer n,nmax
      double precision x,y,nu
      double precision An(0:nmax,0:nmax)
      double precision Cnp(0:*),GN(0:nmax)
      GN(n) = 0D0
      do ir=0,n
        terme=0D0
        do i=0,ir
        terme = terme+An(ir,i)*x**i
        end do
        GN(n) = GN(n) + Cnp(n*(n+1)/2+ir)*(-1D0/y)**ir*terme
      end do
      if (ize .EQ. 1) then
      GN(n) = GN(n)*x**(nu+1)*dexp(x+y)
      else if (ize .EQ. 2) then
      GN(n) = GN(n)*x**(nu+1)*dexp(x+y-2*sqrt(x*y))
       end if
      return
end
```

A.2 Probability Function and Quantile Function Estimation

A.2.1 pghyp Function

```
pghyp <- function (q, mu = 0, delta = 1, alpha = 1, beta = 0,
                   lambda = 1,param = c(mu, delta, alpha,
                                        beta, lambda),
                   lower.tail = TRUE, subdivisions = 100,
                   intTol = .Machine$double.eps^0.25,
                   valueOnly = TRUE, ...){


  ## Purpose: Implement GHyp distribution function evaluating
  ##          approach
  ## ========================================================
  ## Arguments: q, Vector of quantiles
  ##            mu, Location parameter of the GHyp distribution
  ##            delta, Scale parameter of the GHyp distribution
  ##            alpha, Tail parameter of the GHyp distribution
  ##            beta, Skewness parameter of the GHyp distribution
  ##            lambda, Shape parameter of the GHyp distribution
  ##            param, parameter vector of the distribution
  ##            lower.tail, If lower.tail = TRUE, the cumulative
  ##                        density is taken from the lower tail
```

```
##            subdivision, The maximum number of subintervals
##                         of Integrate function
##            intTol, integrate function accuracy requested
##            valueOnly, If valueOnly = TRUE calls to pDist
##                       only return the value obtained for the
##                       integral. If valueOnly = FALSE an
##                       estimate of the accuracy of the
##                       numerical integration is also
##                       returned
##            ..., Passes additional arguments to integrate
##                 function


parResult <- ghypCheckPars(param)
case <- parResult$case
errorMessage <- parResult$errMessage
if (case == "error")
    stop(errMessage)
mu <- param[1]
delta <- param[2]
alpha <- param[3]
beta <- param[4]
lambda <- param[5]

modeDist <- ghypMode(param = param)
qLess <- which((q <= modeDist)&(is.finite(q)))
qGreater <- which((q > modeDist)&(is.finite(q)))
prob <- rep(NA, length(q))
err <- rep(NA, length(q))

prob[q == -Inf] <- 0
prob[q == Inf] <- 0
err[q %in% c(-Inf, Inf)] <- 0

dghypInt <- function(q)
{
    dghyp(q, param = param)
}

for (i in qLess)
{
    intRes <- integrate(dghypInt, -Inf, q[i],
                        subdivisions = subdivisions,
                        rel.tol = intTol, ...)
    prob[i] <- intRes$value
    err[i] <- intRes$abs.error
}
```

```
  for (i in qGreater)
  {
      intRes <- integrate(dghypInt, q[i], Inf,
                          subdivisions = subdivisions,
                          rel.tol = intTol, ...)
      prob[i] <- intRes$value
      err[i] <- intRes$abs.error
  }

  if (lower.tail == TRUE)
  {
      prob[q > modeDist] <- 1 - prob[q > modeDist]
  }
  else
  {
      prob[q <= modeDist] <- 1 - prob[q <= modeDist]
  }

  ifelse(valueOnly, return(prob),
         return(list(value = prob, error = err)))
}
```

A.2.2 qghyp Function

```
qghyp <- function (p, mu = 0, delta = 1, alpha = 1, beta = 0,
                   lambda = 1, param = c(mu, delta, alpha,
                                         beta, lambda),
                   lower.tail = TRUE,
                   method = c("integrate", "spline"),
                   nInterpol = 501,
                   uniTol = .Machine$double.eps^0.25,
                   subdivisions = 100, intTol = uniTol, ...){


  ## Purpose: Implement GHyp quantile function evaluating
  ##          approach
  ## ========================================================
  ## Arguments: p, Vector of probabilities
  ##            mu, Location parameter of GHyp distribution
  ##            delta, Scale parameter of GHyp distribution
  ##            alpha, Tail parameter of GHyp distribution
  ##            beta, Skewness parameter of GHyp distribution
  ##            lambda, Shape parameter of GHyp distribution
  ##            param, parameter vector of the distribution
  ##            lower.tail, If lower.tail = TRUE, the cumulative
```

```
##                        density is taken from the lower tail
##            method, If "spline" quantiles are found from a
##                    spline approximation to the distribution
##                    function. If "integrate", the
##                    distribution function used is always
##                    obtained by integration
##            nInterpol, Number of points used in qghyp for
##                       cubic spline interpolation of
##                       the distribution function
##            uniTol, uniroot function accuracy requested
##            subdivision, The maximum number of subintervals
##                         of Integrate function
##            intTol, integrate function accuracy requested
##            ..., Passes additional arguments to integrate
##                 function

parResult <- ghypCheckPars(param)
case <- parResult$case
errMessage <- parResult$errMessage
if (case == "error")
  stop(errMessage)
if (!lower.tail) {
  p <- 1 - p
  lower.tail == TRUE
}
method <- match.arg(method)
mu <- param[1]
delta <- param[2]
alpha <- param[3]
beta <- param[4]
lambda <- param[5]
modeDist <- ghypMode(param = param)
pModeDist <- pghyp(modeDist, param = param, intTol = intTol)
xRange <- ghypCalcRange(param = param, tol = 10^(-5))
quant <- rep(NA, length(p))
invalid <- which((p < 0) | (p > 1))
pFinite <- which((p > 0) & (p < 1))
if (method == "integrate") {
  less <- which((p <= pModeDist) & (p > .Machine$double.eps^8))
  quant <- ifelse(p <= .Machine$double.eps^8, -Inf, quant)
  if (length(less) > 0) {
    pLow <- min(p[less])
    xLow <- modeDist - sqrt(ghypVar(param = param))
    while (pghyp(xLow, param = param, intTol = intTol) >=
             pLow) {
      xLow <- xLow - sqrt(ghypVar(param = param))
    }
```

```
      xRange <- c(xLow, modeDist)
      zeroFn <- function(x, param, p) {
        return(pghyp(x, param = param,
                     subdivisions = subdivisions,
                     intTol = intTol) - p)
      }
      for (i in less) {
        quant[i] <- uniroot(zeroFn, param = param,
                            p = p[i],
                            interval = xRange,
                            tol = uniTol)$root
      }
    }
    greater <- which((p > pModeDist) &
                       (p < (1 - .Machine$double.eps^8)))
    p[greater] <- 1 - p[greater]
    quant <- ifelse(p >= (1 - .Machine$double.eps^8), Inf,
                    quant)
    if (length(greater) > 0) {
      pHigh <- min(p[greater])
      xHigh <- modeDist + sqrt(ghypVar(param = param))
      while (pghyp(xHigh, param = param, intTol = intTol,
                   lower.tail = FALSE) >= pHigh) {
        xHigh <- xHigh + sqrt(ghypVar(param = param))
      }
      xRange <- c(modeDist, xHigh)
      zeroFn <- function(x, param, p) {
        return(pghyp(x, param = param,
                     lower.tail = FALSE,
                     subdivisions = subdivisions,
                     intTol = intTol) - p)
      }
      for (i in greater) {
        quant[i] <- uniroot(zeroFn, param = param,
                            p = p[i],
                            interval = xRange,
                            tol = uniTol)$root
      }
    }
  }
  else if (method == "spline") {
    inRange <- which((p > pghyp(xRange[1], param = param,
                                intTol = intTol)) &
                       (p < pghyp(xRange[2], param = param,
                                  intTol = intTol)))
    small <- which((p <= pghyp(xRange[1], param = param,
                               intTol = intTol)) & (p >= 0))
```

```
    large <- which((p >= pghyp(xRange[2], param = param,
                               intTol = intTol)) & (p <= 1))
    extreme <- c(small, large)
    xVals <- seq(xRange[1], xRange[2], length.out = nInterpol)
    yVals <- pghyp(xVals, param = param,
                   subdivisions = subdivisions,
                   intTol = intTol)
    splineFit <- splinefun(xVals, yVals)
    zeroFn <- function(x, p) {
      return(splineFit(x) - p)
    }
    for (i in inRange) {
      quant[i] <- uniroot(zeroFn, p = p[i],
                          interval = xRange,
                          tol = uniTol)$root
    }
    if (length(extreme) > 0) {
      quant[extreme] <- qghyp(p[extreme], param = param,
                              lower.tail = lower.tail,
                              method = "integrate",
                              nInterpol = nInterpol,
                              uniTol = uniTol,
                              subdivisions = subdivisions,
                              intTol = intTol, ...)
    }
  }
  return(quant)
}
```

A.2.3 `pskewhyp` Function

```
findDelta <- function(beta, nu, ...)
{
    ## Purpose: Find the value of delta
    ## ====================================================
    ## Arguments: beta, skewness parameter of skew hyperbolic
    ##            nu, shape parameter of the skew hyperbolic
    ##            ..., Additional arguments may passed to optim
    ##

    alpha <- sqrt(2.5)
    optFn <- function(delta) {
        abs(dskewhyp(alpha*delta,
                     param = c(0, 1, beta, nu), log = TRUE) -
                dskewhyp(0, param = c(0, 1, beta, nu),
                         log = TRUE) + 1.25)
```

```
    }

    opt <- optim(1, optFn, ...)
    opt
}


findNu <- function(delta, beta, nu, nuMax = NULL, ...)
{
    ## Purpose: Find the value of nu
    ## ====================================================
    ## Arguments: delta, value of delta found using findDelta
    ##            beta, skewness parameter of skew hyperbolic
    ##            nu, shape parameter of the skew hyperbolic
    ##            nuMax, maximum range over which to search for nu
    ##            ..., Additional arguments may passed to optim


    if (is.null(nuMax)) nuMax <- nu + 100
    optFn <- function(nu0) {
        abs(dskewhyp(x = delta,
                     param = c(0, 1, beta, nu), log = TRUE) -
                dskewhyp(0, param = c(0, 1, beta, nu),
                         log = TRUE) -
            (nu0 + 1)*log(1 + 1/nu0)/2)
    }
    opt <- optim(nu, optFn, method = "Brent", lower = 0, upper = nuMax, ...)
    opt
}


pskewhypGK <- function(q, param,
                       method = c("Gaussian","integrate"),
                       subdivisions = 100,
                       intTol = .Machine$double.eps^0.25){

    ## Purpose: Implement split-t transformation for evaluating
    ##          pskewhyp
    ## ========================================================
    ## Arguments: q, quantile
    ##            param, parameter vector for skew hyperbolic
    ##                   distribution
    ##            method, Integration method
    ##            subdivision, The maximum number of subintervals
    ##                         of Integrate function
    ##            intTol, integrate function accuracy requested
```

```
  mu <- param[1]
  delta <- param[2]
  beta <- param[3]
  nu <- param[4]
  p <- rep(NA, length(q))
  z0 <- rep(NA, length(q))
  p[q == -Inf] <- 0
  p[q == Inf] <- 1

  qLess <- which((q <= mu)&(is.finite(q)))
  qGreater <- which((q > mu)&(is.finite(q)))
  q[qLess] <- 2*mu - q[qLess]

  ## Adjust q and along with it mu and delta
  q <- (q - mu)/delta
  beta <- delta*beta
  mu <- 0

  if (length(qLess) != 0){
  ## Find  G K delta
  deltaLess <- findDelta(-beta, nu,
                         method = "Brent",
                         lower = 0, upper = 100)$par
  ## Find G K nu
  nu0Less <- findNu(deltaLess, -beta, nu, nuMax = 8)$par
  ## Upper limit
  z0[qLess] <- pt(q[qLess]/deltaLess,
                  nu0Less, lower.tail = FALSE)
  }

  if (length(qGreater) != 0){
    ## Find  G K delta
  deltaGreater <- findDelta(beta, nu,
                         method = "Brent",
                         lower = 0, upper = 100)$par
  ## Find G K nu
  nu0Greater <- findNu(deltaGreater, beta, nu, nuMax = 8)$par
  ## Upper limit
  z0[qGreater] <- pt(q[qGreater]/deltaGreater,
                     nu0Greater, lower.tail = FALSE)
}

  for (i in qLess){
    f <- function(z){
    fz <- deltaLess*dskewhyp(deltaLess*qt(z, nu0Less,
                                          lower.tail = FALSE),
                             param = c(0, 1, -beta, nu))/
```

```
                dt(qt(z, nu0Less,
                      lower.tail = FALSE), nu0Less)
    fz
  }
    if (method == "Gaussian"){
      tryOpt <- try(quadgr(f, 0, z0[i]), silent = TRUE)
      if (class(tryOpt) == "try-error"){
        int <- NA
      } else {
        int <- tryOpt$value
    }
    }
    else if (method == "integrate"){
      tryOpt <- try(integrate(f, 0, z0[i],
                              subdivisions = subdivisions,
                              rel.tol = intTol), silent = TRUE)
      if (class(tryOpt) == "try-error"){
        int <- NA
      } else {
        int <- tryOpt$value
    }
    }
    p[i] = int
  }

  for (i in qGreater){
    f <- function(z){
        fz <- deltaGreater*
          dskewhyp(deltaGreater*qt(z, nu0Greater,
                                   lower.tail = FALSE),
                           param = c(0, 1, beta, nu))/
                dt(qt(z, nu0Greater,
                      lower.tail = FALSE), nu0Greater)
    fz
  }
    if (method == "Gaussian"){
      tryOpt <- try(quadgr(f, 0, z0[i]), silent = TRUE)
      if (class(tryOpt) == "try-error"){
        int <- NA
      } else {
        int <- tryOpt$value
    }
    }
    else if (method == "integrate"){
      tryOpt <- try(integrate(f, 0, z0[i],
                              subdivisions = subdivisions,
                              rel.tol = intTol), silent = TRUE)
```

```
        if (class(tryOpt) == "try-error"){
          int <- NA
        } else {
          int <- tryOpt$value
      }
      }
      p[i] = 1 - int
    }

    return(p)
}
```

A.3 Random Number Generation of Generalized Inverse Gaussian and Hyperbolic Distribution

A.3.1 `rgigGamma` Function

```
rgigGamma <- function(n, param){

  ## Purpose: Implement Dagpunar rejection sampling
  ##          approach
  ## ========================================================
  ## Arguments: n, Number of random variates required
  ##            param, parameter vector of the GIG distribution


  chi <- param[1]
  psi <- param[2]
  lambda <- param[3]
  alpha <- sqrt(psi/chi)
  beta <- sqrt(psi*chi)
  if (lambda <= 0) stop("lambda must be positive")
  if (beta <= 0) stop("beta must be positive")

  #The parameter of the Gamma distributed hat
  gam = 2*lambda^2*(sqrt(1 + (beta/lambda)^2) - 1)/beta
  a1 = 0.5*(beta - gam)
  a2 = 0.5*beta
  a3 = sqrt(beta*(beta - gam))
  a4 = gam/2

  # Rejection Sampling
  output <- numeric(n)
  for(i in 1:n){
    need.value <- TRUE
```

```
    while(need.value == TRUE){
      # Generate R and X
      r <- runif(1)
      x <- rgamma(1, lambda, a4)
      if (-log(r) > x*a1 + a2/x - a3){
        need.value = FALSE
      }
    }
    output[i] <- x
  }

  return(output)
}
```

A.3.2 rgigHoer Function

```
rgigHoer <- function(n, param){

  ## Purpose: Implement Hoermann rejection sampling
  ##          approach
  ## ========================================================
  ## Arguments: n, Number of random variates required
  ##            param, parameter vector of the GIG distribution


  chi <- param[1]
  psi <- param[2]
  lambda <- param[3]
  alpha <- sqrt(psi/chi)
  beta <- sqrt(psi*chi)
  x0 <- beta/(1 - lambda)
  a <- max(x0, 2/beta)


  #Evaluate the mode of the distribution
  m <- beta/((1 - lambda) + sqrt((1 - lambda)^2 + beta^2))



  #The function e in f(x)=ce(x)
  h <- function(x, lambda, chi, psi){
    x^(lambda - 1)*exp(-(1/2) * beta * (x^(-1) + x))
  }
  k1 = h(m, lambda, chi, psi)
  k2 = exp(-beta)
  k3 = a^(lambda - 1)
```

```
#The Envelope Function
ev2 <- function(x, lambda, beta){
    x^(lambda - 1) * k2
  }

ev3 <- function(x, a, lambda, beta){
  k3 * exp(-(beta/2 * x))
}

#Computing area
A1 <- k1*x0
A2 <- ifelse(lambda == 0, k2*log(2/beta^2),
             k2*(a^lambda - x0^lambda)/lambda)
A3 <- 2*k3*exp(-beta*a/2)/beta
A = A1 + A2 + A3

output <- numeric(n)

# Rejection Sampling

for(i in 1:n){
  need.value <- TRUE
  while(need.value == TRUE){
      # Generate U1 and U2
        U1 <- runif(1, min = 0, max = A)
        U2 <- runif(1)
        if (U1 <= A1){
          x <- x0*U1/A1
          if(U2*k1<=h(x, lambda, chi, psi)){
            need.value <- FALSE
          }
        }
        else if (U1<=(A1+A2)){
          U1 = U1 - A1
          x <- ifelse(lambda == 0,
                      beta*exp(V*exp(beta)),
                      (x0^lambda +
                         U1*lambda/k2)^(1/lambda))
          if (U2*ev2(x, lambda, beta)<=h(x, lambda, chi, psi)){
            need.value <- FALSE
          }
        } else{
          U1 = U1 - A1 - A2
          x <- -2/beta*log(exp(-a*beta/2) - U1*beta/(2*k3))
          if(U2*ev3(x, a, lambda, beta)<=h(x, lambda, chi, psi)){
            need.value <- FALSE
```

```
            }
          }
        }
        output[i] <- x
      }
  return(output)
}
```

A.3.3 rhypTDR Function

```
rhypTDR <- function(n, param){

  ## Purpose: Implement transformed density rejection sampling
  ##          approach proposed by Hoermann
  ## ========================================================
  ## Arguments: n, Number of random variates required.
  ##            param, parameter vector of
  ##                   the hyperbolic distribution.


  mu <- param[1]
  delta <- param[2]
  alpha <- param[3]
  beta <- param[4]
  zeta <- delta * sqrt(alpha^2 - beta^2)
  hyperbPi <- beta/sqrt(alpha^2 - beta^2)

  if(abs(alpha) <= beta) stop("beta must be smaller than
                              the absolute value of alpha")
  if(delta <= 0) stop("delta must be positive")

  # restrict to delta = 1, mu = 0
  alpha = alpha * delta
  beta = beta * delta
  lowerupper <- hyperbCalcRange(param = c(0, 1, alpha, beta))
  lower <- lowerupper[1]
  upper <- lowerupper[2]

  # Quasi Density of Hyperbolic Distribution
  quasiDens <- function(x, alpha, beta){
    quasiDens <- exp(-alpha * sqrt(1 + x^2) + beta * x)
  }

  # T transformed quasi density function
  h <- function(x, alpha, beta){
    h <- -alpha * sqrt(1 + x^2) + beta * x
```

```
}

# The derivative of h(x)
dh <- function(x, alpha, beta){
  h <- -alpha * x/sqrt(1 + x^2) + beta
}

# Touch points of l(x) and h(x)
xl <- uniroot(function(x){
  -alpha * sqrt(1 + x^2) + beta * x + zeta + 1},
  lower = lower, upper = hyperbPi)$root
xr <- uniroot(function(x){
  -alpha * sqrt(1 + x^2) + beta * x + zeta + 1},
  lower = hyperbPi, upper = upper)$root


hxl <- h(xl, alpha, beta)
hxr <- h(xr, alpha, beta)
dhxl <- dh(xl, alpha, beta)
dhxr <- dh(xr, alpha, beta)

#Two intersection points of the three parts of l(x)
bl <- xl + (-zeta - hxl)/dhxl
br <- xr + (-zeta - hxr)/dhxr

#F(h(m))
Fhm <- exp(-zeta)
fm <- Fhm

#The areas between x-axis and T^(-1)(l(x)) for
#the three intervals
vl <- Fhm/dhxl
vc <- fm * (br - bl)
vr <- -Fhm/dhxr

# Rejection Sampling
output <- numeric(n)
for(i in 1:n){
  need.value <- TRUE
  while(need.value == TRUE){
    U <- (vl + vc + vr)*runif(1)
    if (U <= vl){
      X <- (log(-U * dhxl + Fhm) - hxl)/dhxl + xl
      lx <- exp(dhxl * (X - xl) + hxl)
    }else{
      if(U <= (vl + vc)){
        X <- ((U - vl)/vc) * (br - bl) + bl
```

```
          lx <- fm
        }else{
          X <- (log(((U - (vl + vc))) *
                      dhxr + Fhm) - hxr)/dhxr + xr
          lx <- exp(dhxr*(X-xr)+hxr)
        }
      }
      V <- lx * runif(1)
      fX <- quasiDens(X, alpha, beta)
      if(V <= fX){
          need.value <- FALSE
      }
    }
    output[i] <- X
  }
  return(delta * output + mu)
}
```

A.3.4 `rhypTDRsq` Function

```
rhypTDRsq <- function(n,param){

  ## Purpose: Implement transformed density rejection with
  ##          squeeze function sampling approach proposed by
  ##          Hoermann
  ## ========================================================
  ## Arguments: n, Number of random variates required.
  ##            param, parameter vector of
  ##                   the hyperbolic distribution.


  mu <- param[1]
  delta <- param[2]
  alpha <- param[3]
  beta <- param[4]
  zeta <- delta * sqrt(alpha^2 - beta^2)
  hyperbPi <- beta/sqrt(alpha^2 - beta^2)
  count = numeric(n)

  if(abs(alpha) <= beta) stop("beta must be smaller than
the absolute value of alpha")
  if(delta <= 0) stop("delta must be positive")

  # restrict to delta = 1, mu = 0
  alpha = alpha*delta
  beta = beta*delta
```

```
lowerupper <- hyperbCalcRange(param = c(0, 1, alpha, beta))
lower <- lowerupper[1]
upper <- lowerupper[2]

# Quasi Density of Hyperbolic Distribution
quasiDens <- function(x, alpha, beta){
  quasiDens <- exp(-alpha * sqrt(1 + x^2) + beta * x)
}

# T transformed quasi density function
h <- function(x, alpha, beta){
  h <- -alpha * sqrt(1 + x^2) + beta * x
}

# The derivative of h(x)
dh <- function(x , alpha , beta){
  h <- -alpha * x/sqrt(1 + x^2) + beta
}

# Touch points of l(x) and h(x)
xl <- uniroot(function(x){
  -alpha * sqrt(1 + x^2) + beta * x + zeta + 1},
  lower = lower, upper = hyperbPi)$root
xr <- uniroot(function(x){
  -alpha * sqrt(1 + x^2) + beta * x + zeta + 1},
  lower = hyperbPi, upper = upper)$root


hxl <- h(xl, alpha, beta)
hxr <- h(xr, alpha, beta)
dhxl <- dh(xl, alpha, beta)
dhxr <- dh(xr, alpha, beta)

#Two intersection points of the three parts of l(x)
bl <- xl + (-zeta - hxl)/dhxl
br <- xr + (-zeta - hxr)/dhxr

#F(h(m))
Fhm <- exp(-zeta)
fm <- Fhm

#The areas between x-axis and T^(-1)(l(x)) for
#the three intervals
vl <- Fhm/dhxl
vc <- fm * (br - bl)
vr <- -Fhm/dhxr
```

```
#Define squeeze function
sl <- (-zeta - hxl)/(hyperbPi - xl)
sr <- (-zeta - hxr)/(hyperbPi - xr)


# Rejection Sampling
output <- numeric(n)
for(i in 1:n){
  need.value <- TRUE
  while(need.value == TRUE){
    U <- (vl + vc + vr) * runif(1)
    if(U <= vl){
      X <- (log(-U * dhxl + Fhm) - hxl)/dhxl + xl
      lx <- exp(dhxl * (X - xl) + hxl)
    }else{
      if(U <= (vl + vc)){
        X <- ((U - vl)/vc) * (br - bl) + bl
        lx <- fm
      }else{
        X <- (log(((U - (vl + vc))) * dhxr +
                    Fhm) - hxr)/dhxr + xr
        lx <- exp(dhxr * (X - xr) + hxr)
      }
    }
    V <- lx * runif(1)
    fX <- quasiDens(X, alpha, beta)
    if(X < hyperbPi){
      if((X > xl) &
           (V <= exp(-zeta - (hyperbPi - X) * sl))){
        need.value <- FALSE
      }else{
        if(V <= fX){
          need.value <- FALSE
        }
      }
    }else{
      if((X < xr) &
           (V <= exp(-zeta - (hyperbPi - X) * sr))){
        need.value <- FALSE
      }else{
        if(V <= fX){
          need.value <- FALSE
        }
      }
    }
  }
  output[i] <- X
```

```
  }
  return(delta * output + mu)
}
```

A.4 Robust Linear Modeling using the Hyperbolic Distribution

A.4.1 `hyperblm` Function

```
hyperblm <- function(formula, data, subset, weights, na.action,
                     x = FALSE, y = FALSE, contrasts = NULL,
                     offset, method="Nelder-Mead",
                     startMethod="Nelder-Mead", startStarts="BN",
                     paramStart=NULL,
                     maxiter = 100, tolerance = 0.0001,
                     controlBFGS=list(maxit=1000),
                     controlNM=list(maxit=10000),
                     maxitNLM=10000,
                     controlCO = list(), silent = TRUE, ...){


  ## Purpose: Fits linear models with hyperbolic errors
  ##
  ## ========================================================
  ## Arguments: formula, A symbolic description of the model
  ##                     to be fitted.
  ##            data, An optional data frame, list or environment
  ##                  containing the variables in the model.
  ##            subset, An optional vector specifying a subset
  ##                    of observations to be used in the
  ##                    fitting process.
  ##            weights, An optional vector of weights to be used
  ##                     in the fitting process.
  ##            na.action, A function which indicates what should
  ##                       happen when the data contain NAs.
  ##            x, y, If TRUE, the corresponding components of
  ##                  the fit (the explanatory matrix and the
  ##                  response vector) are returned.
  ##            contrasts, A list, whose entries are values
  ##                       (numeric matrices or character strings
  ##                       naming functions) to be used as
  ##                       replacement values for the
  ##                       contrasts replacement function
  ##                       and whose names are the names of
  ##                       columns of data containing factors.
  ##            offset, A term to be added to a linear predictor,
  ##                    such as in a generalised linear model,
```

```
##                    with known coefficient 1 rather than
##                    an estimated coefficient.
##            method, Possible values are "BFGS", "Nelder-Mead"
##                    and "nlm".
##            startMethod, Possible values are "BFGS"
##                         and "Nelder-Mead".
##            startStarts, Possible values are "BN", "FN",
##                         "SL", "US" and "MoM".
##            paramStart, A vector of parameter start values
##                        for the optimization routine.
##            maxiter, The maximum number of two-stage
##                     optimization alternating iterations.
##            tolerance, The two-stage optimization convergence
##                       ratio.
##            controlBFGS, A List of control parameters for optim
##                         when using BFGS optimisation method in
##                         first stage.
##            controlNM, A list of control parameters for optim
##                       when using NM optimisation method in
##                       first stage.
##            maxitNLM, The maximum number of iterations for
##                      the NLM optimizer.
##            controlCO, A list of control parameters for
##                       constrOptim in second stage.
##            silent, If TRUE, the error messgae of
##                    optimizer will not be displayed.


ret.x <- x
ret.y <- y
cl <- match.call()
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data", "subset", "weights", "na.action",
             "offset"), names(mf), 0)
mf <- mf[c(1,m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
y <- model.response(mf, "numeric")
w <- model.weights(mf)
offset <- model.offset(mf)
if(!is.null(offset) && length(offset) != NROW(y))
  stop("Number of offsets is ", length(offset),
       ", should equal ", nrow(y), " (number of observations)")
if(is.empty.model(mt)){
  x <- NULL
```

```
    regressionResult <- list(coefficients = numeric(0),
                             distributionParams = numeric(0),
                             residuals=y, fitted.values=0 * y,
                             weights=w, rank=0,
                             df.residual=length(y))
    if(!is.null(offset))
      regressionResult$fitted.values <- offset
  }
  else{
    x <- model.matrix(mt, mf, contrasts)
    regressionResult <- hyperblmfit(x, y, offset = offset,
                                    method = method,
                                    startMethod = startMethod,
                                    startStarts = startStarts,
                                    paramStart = paramStart,
                                    maxiter = maxiter,
                                    tolerance = tolerance,
                                    controlBFGS = controlBFGS,
                                    controlNM = controlNM,
                                    maxitNLM = maxitNLM,
                                    controlCO = controlCO,
                                    silent = silent, ...)
  }
  class(regressionResult) <- "hyperblm"
  regressionResult$na.action <- attr(mf, "na.action")
  regressionResult$offset <- offset
  regressionResult$contrasts <- attr(x, "contrasts")
  regressionResult$xlevels <- .getXlevels(mt, mf)
  regressionResult$call <- cl
  regressionResult$terms <- mt
  if(ret.x)
    regressionResult$x <- x
  if(ret.y)
    regressionResult$y <- y
  regressionResult
}
```

A.4.2 `hyperblmfit` Function

```
hyperblmfit <- function(x, y, paramStart = NULL, offset = NULL,
                        method = c("Nelder-Mead","BFGS","nlm"),
                        startMethod = c("Nelder-Mead","BFGS"),
                        startStarts = c("BN","US","FN","SL","MoM"),
                        maxiter = 100, tolerance = 0.001,
                        controlBFGS = list(maxit = 1000),
                        controlNM = list(maxit = 1000),
```

```
                        maxitNLM = 10000,
                        controlCO = list(), silent = TRUE,
                        breaks = NULL, ...){


## Purpose: Fitting function for linear models with hyperbolic
##          errors
##
## ========================================================
## Arguments: x, Explanatory variable(s)
##            y, Response variable(s)
##            paramStart, A vector of parameter start values
##                        for the optimization routine.
##            offset, A term to be added to a linear predictor,
##                    such as in a generalised linear model,
##                    with known coefficient 1 rather than
##                    an estimated coefficient.
##            method, Possible values are "BFGS", "Nelder-Mead"
##                    and "nlm".
##            startMethod, Possible values are "BFGS"
##                         and "Nelder-Mead".
##            startStarts, Possible values are "BN", "FN",
##                         "SL", "US" and "MoM".
##            maxiter, The maximum number of two-stage
##                     optimization alternating iterations.
##            tolerance, The two-stage optimization convergence
##                       ratio.
##            controlBFGS, A List of control parameters for optim
##                         when using BFGS optimisation method in
##                         first stage.
##            controlNM, A list of control parameters for optim
##                       when using NM optimisation method in
##                       first stage.
##            maxitNLM, The maximum number of iterations for
##                      the NLM optimizer.
##            controlCO, A list of control parameters for
##                       constrOptim in second stage.
##            silent, If TRUE, the error messgae of
##                    optimizer will not be displayed.
##            breaks, Numbers of intervals the density curve is
##                    divided into


if(is.null(n <- nrow(x)))
  stop("'x' must be a matrix")
if(n == 0)
  stop("0 (non-NA) cases")
```

```
p <- ncol(x)
if(p == 0){
  return(list(coefficients = numeric(0), residuals = y,
              fitted.values = 0*y, rank = 0,
              df.residual = length(y)))
}else{
  xNames <- colnames(x)
}
ny <- NCOL(y)
if(is.matrix(y) && ny == 1)
  y <- drop(y)
if(!is.null(offset))
  y <- y - offset
if(NROW(y) != n)
  stop("incompatible dimensions")
storage.mode(x) <- "double"
storage.mode(y) <- "double"
## set default error message
errMessage <- ""


if(is.null(paramStart)){
    qrx <- qr(x)
    resids <- qr.resid(qrx, y)
    Beta <- as.numeric(qr.coef(qrx, y))
    startInfo <- hyperbFitStand(resids,
                                startMethod = startMethod,
                                method = "constrOptim",
                                startValues = startStarts,
                                silent = TRUE, ...)
    residsParamStart <- as.numeric(startInfo$param)

    ## change residsParamStart to param set number 1
    ## (mu,delta,pi,zeta)
    distparam <-
      as.numeric(hyperbChangePars(2, 1,
                                  param =
                                    residsParamStart))[-1]
    coef <- c(residsParamStart[1] + Beta[1], Beta[-1])
    breaks <- startInfo$breaks
}else{
  if(length(paramStart)!= (3 + p))
    stop(paste("Parameters start value should be of dimension",
               3 + p, sep=" "))
  if(paramStart[2] <= 0)
    stop("zeta in paramStart must be greater than zero")
  if(paramStart[1] <= 0)
```

```
    stop("delta in paramStart must be greater than zero")
  distparam <- paramStart[1:3]
  coef <- paramStart[-(1:3)]
}

## Set some parameters to help with optimization
eps <- 1e-16
ratioCoef <- 1
rationParam <- 1
iter <- 0

## A artificial iterations counter that equals to
## the number of iterations
## when the optimization does not converge and
## is greater than maximum
## iterations number when the optimization converges.
## This helps to break the loop.
hiter <- 0

sOnellfunc <- function(coef){
  KNu <- besselK(distparam[3], nu = 1)
  resids <- y -
    as.vector(x * as.matrix(coef))
  hyperbDens <- (2 * distparam[1] *
                   sqrt(1 + distparam[2]^2) * KNu)^(-1) *
                 exp(-distparam[3] * (sqrt(1 + distparam[2]^2) *
                 sqrt(1 + ((resids)/distparam[1])^2) -
                 distparam[2] * ((resids)/distparam[1])))
  return(-sum(log(hyperbDens)))
}


sTwollfunc <- function(distparam) {
  ## Protect against attempts to make parameters < 0
  if (distparam[3] <= eps) return(1e99)
  if (distparam[1] <= eps) return(1e99)
  KNu <- besselK(distparam[3], nu = 1)
  resids <-
    y - as.vector(x * as.matrix(coef))
  hyperbDens <- (2 * distparam[1] *
                   sqrt(1 + distparam[2]^2) * KNu)^(-1) *
                 exp(-distparam[3] * (sqrt(1 + distparam[2]^2) *
                 sqrt(1 + ((resids)/distparam[1])^2) -
                 distparam[2] * ((resids)/distparam[1])))
  return(-sum(log(hyperbDens)))
}
```

```
while(!is.null(coef) &&
        !is.null(distparam) && hiter < maxiter){
  output <- numeric(7)
  ind <- 1:6

  ##Stage one find the coefficient
  ##optimize method
  if(!is.null(distparam)){
    coefOld <- coef
  if(method == "BFGS"){
      tryOpt <- try(optim(coef, sOnellfunc,
                          NULL, method = "BFGS",
                          control = controlBFGS, ...),
                    silent = silent)
      if (class(tryOpt) == "try-error"){
          errMessage <- unclass(tryOpt)
      } else {
          optOutCoef <- tryOpt
      }
  }
  if(method=="Nelder-Mead"){
      tryOpt <- try(optim(coef, sOnellfunc,
                          NULL, method = "Nelder-Mead",
                          control = controlNM, ...),
                    silent = silent)
      if (class(tryOpt) == "try-error"){
          errMessage <- unclass(tryOpt)
    } else {
      optOutCoef <- tryOpt
    }
}

  if(method=="nlm"){
      ind <- c(2, 1, 5, 4)
      tryOpt <- try(nlm(sOnellfunc, coef,
                        iterlim = maxitNLM, ...),
                    silent = silent)
    if (class(tryOpt) == "try-error"){
        errMessage <- unclass(tryOpt)
    } else {
      optOutCoef <- tryOpt
    }
}
  if (errMessage == ""){
      coef <- as.numeric(optOutCoef[[ind[1]]])
      ratioCoef <- max(abs((coefOld - coef)/coefOld))
```

```
    }
    else coef <- NULL
    }

    ## Stage Two Optmization
    if (!is.null(coef)){
    ind <- 1:6
    distparamOld <- distparam
    tryOpt <- try(optOut<- constrOptim(theta = distparam,
                                       sTwollfunc, NULL,
                                       ui = diag(c(1, 0, 1)),
                                       ci = c(0, -1e+99, 0),
                                       control = controlCO, ...),
                  silent = silent)

    if (class(tryOpt) == "try-error"){
        errMessage <- unclass(tryOpt)
      } else {
        optOut <- tryOpt
      }
    if (errMessage == ""){
      distparam <- as.numeric(optOut[[ind[1]]])
      ratioParam <- max(abs((distparamOld - distparam)/distparamOld))
    } else {
      distparam <- NULL
    }
    }
    iter <- iter + 1
    if(ratioCoef <= tolerance && ratioParam <= tolerance )
      hiter <- maxiter + 1
    else hiter <- iter
}


  if (!is.null(distparam)){
    alpha <- distparam[3] * sqrt(1 + distparam[2]^2) / distparam[1]
    beta <- distparam[3] * distparam[2] / distparam[1]
      ordistparam <- c(distparam[1], alpha, beta)

    KNu <- besselK(distparam[3], nu = 1)
    resids <- y - as.vector(x * as.matrix(coef))
    hyperbDens <- (2 * distparam[1] *
                     sqrt(1 + distparam[2]^2) * KNu)^(-1) *
      exp(-distparam[3] * (sqrt(1 + distparam[2]^2) *
                           sqrt(1 + (resids/distparam[1])^2) -
                           distparam[2] * (resids/distparam[1])))
```

```
      ## Calculate the MLE
      maxLik <- sum(log(hyperbDens))
      if(ratioCoef <= tolerance && ratioParam <= tolerance)
        conv <- 0
      else if(ratioCoef > tolerance && ratioParam <= tolerance)
      conv <- 1
      else if(ratioCoef <= tolerance && ratioParam > tolerance)
      conv <- 2
      else conv <- 3
    iter <- iter
  } else {
    ordistparam <- NULL
    maxLik <- NULL
    conv <- 3
    iter <- NULL
  }

  # Fitted value
  fits <- x * as.matrix(coef)

  if(!is.null(offset)){
    fits <- fits + offset
  }

  resids <- y - fits
  m.r <- mean(resids)

  fits <- fits + m.r
  resids <- resids - m.r

  distributionParams <- c(-m.r, ordistparam)
  coef[1] <- coef[1] + m.r

  regressionResult <- list(coefficients = coef,
                           distributionParams = distributionParams,
                           MLE = maxLik, method = method,
                           convergence = conv, iterations = iter,
                           fitted.values=fits,
                           paramStart = paramStart,
                           breaks = breaks,
                           residsParamStart=residsParamStart,
                           xNames=xNames,
                           residuals=resids,
                           xMatrix = x, yVec = y)

  return(regressionResult)
}
```

A.4.3 S3 Methods for `hyperblm`

A.4.3.1 `print.hyperblm` Function

```
print.hyperblm <- function(object,
                           digits = max(3, getOption("digits")-3), ...){


  ## Purpose: Print function for hyperbolic modelling
  ##          result
  ##
  ## ========================================================
  ## Arguments: object, The hyperblm fitted model
  ##            digits, Desired number of digits when
  ##                    the object is printed.
  ##            ..., Additional arguments may pass in the
  ##                 function.


  cat("\nCall:\n", deparse(object$call), "\n", sep = "")
  coefficients <- object$coefficients
  distributionParams <- object$distributionParams
  cat("\nData:     ", object$obsName, "\n")
  if(length(coefficients) == 0){
    cat("\nNo coefficient\n")
  } else{
    cat("Regression coefficient estimates:\n")
    print.default(format(coefficients,digits=digits),
                  print.gap=2,quote=FALSE)
  }
  if(length(distributionParams) == 0){
    cat("\nNo parameter for hyperbolic error distribution\n")
  } else{
    names(distributionParams) <- c("mu", "delta", "alpha", "beta")
    cat("Distribution Parameter estimates:\n")
    print.default(format(distributionParams,digits=digits),
                  print.gap=2,quote=FALSE)
  }
  cat("Method:            ", object$method, "\n")
  cat("Likelihood:        ", object$MLE, "\n")
  cat("Convergence code:  ", object$conv, "\n")
  cat("Iteration:         ", object$iter, "\n")
  invisible(object)
}
```

A.4.3.2 `coef.hyperblm` Function

```
coef.hyperblm <- function(object, ...){

  ## Purpose: Obtain the regression coefficients
  ##          and error distribution parameters
  ##          of the fitted model.
  ##
  ## ========================================================
  ## Arguments: object, The hyperblm fitted model
  ##            ..., Additional arguments may pass in the
  ##                 function.


  list(coefficients = object$coefficient,
       distributionParams = object$distributionParams)
}
```

A.4.3.3 `plot.hyperblm` Function

```
plot.hyperblm <- function(object, breaks = "FD",
                          plotTitles = c("Residuals vs Fitted Values",
                                         "Histogram of residuals",
                                         "Log-Histogram of residuals",
                                         "Q-Q Plot"), ...){


  ## Purpose: Obtain a residual vs fitted value plot,
  ##          a histgram of residuals with error distribution density
  ##          curve superimposed,
  ##          a histgram of log residuals with error distribution
  ##          error density curve superimposed
  ##          and a QQ plot.
  ##
  ## ========================================================
  ## Arguments: object, The hyperblm fitted model
  ##            breaks, Histogram breaks
  ##            plotTitles, A vector of plot titles.
  ##            ..., Additional arguments pass to in particular
  ##                 plotting functions.



  if (! "hyperblm" %in% class(object))
    stop("Object must belong to class hyperblm")


  if(is.null(object$coef))
```

```
  stop("Object has Null output")

par(mar = c(6, 4, 4, 2) + 0.1)

x <- object$xMatrix
y <- as.numeric(object$yVec)
fitted.values <- object$fitted.values
residuals <- object$residuals
xName <- object$xName
yName <- names(object$model)[1]
distributionParams <- object$distributionParams
coefficients <- object$coefficients


hypDens <- function(residuals)
  dhyperb(residuals, param = distributionParams)
logHypDens <- function(residuals)
  log(dhyperb(residuals, param = distributionParams))

par(mfrow = c(2, 2), ...)

plot(fitted.values, residuals, xlab="Fitted Values",
    ylab="Residuals", main=plotTitles[1])
abline(h=0, lty=3)


histData <- hist(residuals, breaks=breaks, right=FALSE,
                 plot=FALSE)
breaks = histData$breaks
empDens <- ifelse(!is.finite(log(histData$density)),
                  NA, histData$density)
ymax <- 1.06 * max(hypDens(seq(min(breaks), max(breaks), 0.001)),
                   empDens, na.rm=TRUE)

hist(residuals, right = FALSE, freq=FALSE, ylim = c(0, ymax),
     main = plotTitles[2], breaks = breaks,
     ylab = "probability density", xlab = "residuals", ...)
curve(dhyperb(x, param = distributionParams), add = TRUE,
      ylab = NULL, col = 2, lwd = 1.5)


logHist(residuals, breaks = breaks, include.lowest = TRUE,
        right = FALSE, main = plotTitles[3], ...)
curve(log(dhyperb(x, param = distributionParams)),
     add = TRUE, ylab = NULL, xlab = NULL, col = 2)
```

```
  alphas <- (1:length(residuals) - 0.5)/length(residuals)
  quantiles <- qhyperb(alphas, param = distributionParams)
  qqplot(quantiles, residuals,
         main = plotTitles[4],
         xlab = "Hyperbolic quantiles",
         ylab = "residuals quantiles", ...)
  abline(0, 1)


}
```

A.4.3.4 `summary.hyperblm` Function

```
summary.hyperblm <- function(object,
                             hessian = FALSE,
                             nboots = 1000, ...){

  ## Purpose: Summary function for hyperbolic modelling
  ##          result
  ##
  ## ========================================================
  ## Arguments: object, The hyperblm fitted model
  ##            hessian, Whether to use Hessian matrix
  ##                     to compute standard error.
  ##            nboots, Number of iterations to approximate
  ##                    standard error
  ##            ..., Additional arguments may pass in the
  ##                 function.


  if (! "hyperblm" %in% class(object))
    stop("Object must belongs to class hyperblm")
  x <- object$xMatrix
  y <- as.numeric(object$yVec)
  distparam <- as.numeric(object$distributionParams)
  coef <- as.numeric(object$coef)
  param <- c(distparam, coef)
  hs <- NULL

  if(hessian == FALSE){
    n = ncol(x)
    bParam = matrix(ncol = 4, nrow = nboots)
    bCoef = matrix(ncol = n, nrow = nboots)

    for(i in 1:nboots){
      w = apply(x, 2, sample, replace = TRUE)
      z = as.vector(w*as.matrix(coef)) +
```

```
        rhyperb(length(y), param = distparam)
      tryOpt <- try(fittedresult <-
                    hyperblmfit(x = w,
                                y = z,
                                method = object$method,
                                startMethod = object$startMethod,
                                startStarts = object$startStarts,
                                paramStart = object$paramStart),
                    silent = TRUE)
      if (class(tryOpt) == "try-error"){
        bParam[i, ] <- rep(NA, 4)
        bCoef[i, ] <- rep(NA, n)
      } else {
        bParam[i, ] <- as.numeric(fittedresult$distributionParams)
        bCoef[i, ] <- as.numeric(fittedresult$coef)
        }
    }
    ses = c(sqrt(apply(bParam, 2, var, na.rm = TRUE)),
      sqrt(apply(bCoef, 2, var, na.rm = TRUE)))
  } else {
  param <- c(distparam, coef)
  llfunc <- function(param){
    resids <- y - as.vector(x * as.matrix(param[-(1:4)]))
    hyperbDens <- dhyperb(resids, param = param[1:4])
    return(sum(log(hyperbDens)))
  }
  hs <- tsHessian(param, llfunc)
  varcov <- make.positive.definite(solve(hs))
  variance <- diag(varcov)
  ses <- sqrt(variance)
}

  object$tval <- coef/ses[-(1:4)]
  object$rdf <- nrow(object$xMatrix) - ncol(object$xMatrix)-3
  object$pval <- 2*pt(abs(object$tval),
                      object$rdf, lower.tail=FALSE)
  object$hessian <- hs
  object$Coefsds <- ses[-(1:4)]
  object$Paramsds <- ses[1:4]

  class(object) <- "summary.hyperblm"
  return(object)
}
```

A.4.3.5 `print.summary.hyperblm` Function

```
print.summary.hyperblm <- function(object,
                                   digits = max(3, getOption("digits") - 3),
                                   ...){

  ## Purpose: Print function for hyperbolic modelling
  ##          summary result
  ##
  ## ==========================================================
  ## Arguments: object, The hyperblm fitted model
  ##            digits, Desired number of digits when
  ##                    the object is printed.
  ##            ..., Additional arguments may pass in the
  ##                 function.

  if (class(object) != "summary.hyperblm")
    stop("Object must belong to class summary.hyperblm")
  cat("\nCall:\n", deparse(object$call), "\n", sep = "")

  cat("\nData:      ", object$xNames, "\n")


  if (!is.null(object$hessian)) {
    cat ("Hessian: (mu, delta, pi, zeta) parameter\n")
    print.default(object$hessian)
  }
  cat("Parameter estimates:\n")
  if (is.null(object$Coefsds)) {
    coefficients <- object$coef
    names(coefficients) <- object$xNames
    print.default(format(coefficients, digits=digits),
                  print.gap=2, quote=FALSE, right=TRUE)
  } else {
    coefficients <- cbind(object$coef, object$Coefsds,
                          object$tval, object$pval)
    dimnames(coefficients) <- list(object$xNames,
                                   c("Estimate", "Std. Error",
                                     "t value", "Pr(>|t|)"))
    printCoefmat(coefficients)
  }

  cat("\nError distribution parameter estimates:\n")
  if (is.null(object$Paramsds)) {
    distparam <- object$distributionParams
    names(distparam) <- c("mu", "delta", "alpha", "beta")
    print.default(format(distparam, digits=digits),
                  print.gap=2, quote=FALSE, right=TRUE)
  } else {
```

```
    distparam <- cbind(object$distributionParams, object$Paramsds)
    dimnames(distparam) <- list(c("mu", "delta", "alpha", "beta"),
                                c("Estimate", "Std. Error"))
    printCoefmat(distparam)
  }

  cat("\nLikelihood:         ", object$MLE, "\n")
  cat("Method:              ", object$method, "\n")
  cat("Convergence code:    ", object$conv, "\n")
  cat("Iterations:          ", object$iter, "\n")
  invisible(object)
}
```

Bibliography

[1] K. Aas and I. H. Haff.
The generalized hyperbolic skew student's t-distribution.
Journal of Financial Econometrics, 4(2):275–309, 2006.

[2] J. H. Ahren and U. Dieter.
Computer methods for sampling from the exponential and normal distributions.
Commun. ACM, 15:873–882, 1972.

[3] A. C. Atkinson.
The simulation of generalized inverse Gaussian and hyperbolic random variables.
SIAM Journal on Scientific and Statistical Computing, 3(4), 1982.

[4] A. Azzalini.
A class of distributions which includes the normal ones.
Scandinavian Journal of Statistics, 12:171–178, 1985.

[5] A. Azzalini.
The skew-normal distribution and related multivariate families.
Scandinavian Journal of Statistics, 32(2):159–188, 2005.

[6] A. Azzalini and A. Capitanio.
Statistical applications of the multivariate skew-normal distribution.
Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61:579–602, 1999.

[7] A. Azzalini and A. Capitanio.
The Skew-Normal and Related Families.
Cambridge University Press, 2014.

[8] O. Bardorff-Nielsen.
Hyperbolic distributions and distributions on hyperbolae.
Scandinavian Journal of Statistics, pages 151–157, 1978.

[9] O. Barndorff-Nielsen.
Exponentially decreasing distributions for the logarithm of particle size.

Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 353:401–419, 03 1977.

[10] O. Barndorff-Nielsen.
Normal Inverse Gaussian Processes and the Modelling of Stock Returns.
University of Aarhus, Department of Theoretical Statistics, 1995.

[11] O. Barndorff-Nielsen.
Normal inverse Gaussian distributions and stochastic volatility modelling.
Scandinavian Journal of Statistics, 24(1):1–13, 1997.

[12] O. Barndorff-Nielsen and P. Blaesild.
Hyperbolic distributions and ramifications: Contributions to theory and application.
In C. Taillie, G. Patil, and B. Baldessari, editors, *Statistical Distributions in Scientific Work*, volume 79 of *NATO Advanced Study Institutes Series*, pages 19–44. Springer Netherlands, 1981.

[13] O. Barndorff-Nielsen and P. Blaesild.
Hyperbolic distributions and ramifications: Contributions to theory and application.
In C. Taillie, G. Patil, and B. Baldessari, editors, *Statistical Distributions in Scientific Work*, volume 79 of *NATO Advanced Study Institutes Series*, pages 19–44. Springer Netherlands, 1981.

[14] O. Barndorff-Nielsen and P. Blaesild.
Hyperbolic distributions.
In *Encyclopedia of Statistical Sciences*, volume 3. New York: Wiley, 1983.

[15] O. Barndorff-Nielsen and C. Halgreen.
Infinite divisibility of the hyperbolic and generalized inverse Gaussian distributions.
Zeitschrift fur Wahrscheinlichkeitstheorie und Verwandte Gebiete, 38(4):309–311, 1977.

[16] O. Barndorff-Nielsen and N. Shephard.
Normal modified stable processes.
Theory Probab. Math. Statist., 65:1–20, 2002.

[17] O. Barndorff-Nielsen, P. Blaesild, and C. Halgreen.
First hitting time models for the generalized inverse Gaussian distribution.
Stochastic Processes and their Applications, 7(1), 1978.

[18] O. Barndorff-Nielsen, P. Blæsild, J. Jensen, and M. Sorenson.
The fascination of sand.
In A. C. Atkinson and S. E. Fienberg, editors, *A Celebration of Statistics*, pages 57–87. New York: Springer-Verlag, 1985.

[19] O. E. Barndorff-Nielsen.
Processes of normal inverse Gaussian type.
Finance and Stochastics, 2(1):41–68, 1997.

[20] F. E. Benth and J. Saltyte-Benth.
The normal inverse Gaussian distribution and spot price modelling in energy markets.
International Journal of Theoretical and Applied Finance, 7(2):177–192, 2004.

[21] B. M. Bibby and M. Sørensen.
A hyperbolic diffusion model for stock prices.
Finance and Stochastics, 1:25–41, 1996.

[22] B. M. Bibby and M. Sørensen.
Hyperbolic processes in finance.
Handbook of heavy tailed distributions in finance, pages 211–248, 2003.

[23] P. Blaesild.
The Shape of the Generalized Inverse Gaussian and Hyperbolic Distributions.
Department of Theoretical Statistics, Inst., Univ., 1978.

[24] E. Bolviken and F. E. Benth.
Quantification of risk in Norwegian stocks via the normal inverse Gaussian distribution.
In *Proceedings of the AFIR 2000 Colloquium, Tromso, Norway*, volume 8798, 2000.

[25] G. E. P. Box and D. R. Cox.
An analysis of transformations.
Journal of the Royal Statistical Society. Series B (Methodological), 26:211–252, 1964.

[26] G. E. P. Box and M. E. Muller.
A note on the generation of random normal deviates.
The Annals of Mathematical Statistics, 29:610–611, 1958.

[27] M. W. Browne.
Asymptotically distribution-free methods for the analysis of covariance structures.
British Journal of Mathematical and Statistical Psychology, pages 62–83, 1984.

[28] K. A. Brownlee.
Statistical Theory and Methodology in Science and Engineering.
Wiley New York, 1965.

[29] J. B. Burbidge, L. Magee, and A. L. Robb.
Alternative transformations to handle extreme values of the dependent variable.
Journal of the American Statistical Association, 83:123–127, 1988.

[30] R. Byrd, P. Lu, J. Nocedal, and C. Zhu.
A limited memory algorithm for bound constrained optimization.
SIAM Journal on Scientific Computing, 16(5):1190–1208, 1995.

[31] F. Camphausen, K. M., R. P., and S. T.
Object Oriented Implementation of Distributions, 8 2014.
R package version 2.5.3.

[32] R. C. Cheng and M. A. Stephens.
A goodness-of-fit test using Moran statistic with estimated parameters.
Biometrika, 76(2):385–392, 1989.

[33] J. S. Dagpunar.
An easily implemented generalized inverse Gaussian generator.
Commun. Statist. -Simula., 18:703–710, 1989.

[34] J. S. Dagpunar.
Generation of Variates from Standard Distributions, chapter 4, pages 59–77.
John Wiley & Sons, Ltd, 2007.

[35] P. Damien, J. Wakefield, and S. Walker.
Gibbs sampling for Bayesian non,Äêconjugate and hierarchical models by using auxiliary variables.
Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61(2):331–344, 1999.

[36] S. Demarta and A. McNeil.
The t copula and related copulas.
International Statistical Review / Revue Internationale de Statistique, 73(1):111–129, 4 2005.

[37] S. Demarta and A. J. McNeil.
The *t* copula and related copulas.
International Statistical Review, 73(1):111–129, 2005.

[38] G. Derflinger, W. Hörmann, and J. Leydold.
Random variate generation by numerical inversion when only the density is known.
ACM Transactions on Modeling and Computer Simulation, 20(18), 10 2010.

[39] E. Eberlein.
Application of generalized hyperbolic Lévy motions to finance.
In O. E. Barndorff-Nielsen, S. I. Resnick, and T. Mikosch, editors, *Levy Processes: Theory and Applications*, pages 319–336. Birkhäuser Boston, 2001.

[40] E. Eberlein and U. Keller.
Hyperbolic distributions in finance.
Bernoulli, pages 281–299, 1995.

[41] E. Eberlein, U. Keller, and K. Prause.
New insights into smile, mispricing, and value at risk: The hyperbolic model.
The Journal of Business, 71(3):371–405, 1998.

[42] B. Efron.
Bootstrap methods: Another look at the jackknife.
The Annals of Statistics, 7(1):1–26, 1979.

[43] P. Embrechts.
A property of the generalized inverse Gaussian distribution with some applications.
Journal of Applied Probability, 20(3):537–544, 1983.

[44] A. Eriksson, E. Ghysels, and F. Wang.
The normal inverse Gaussian distribution and the pricing of derivatives.
The Journal of Derivatives, 16(3):23–37, 2009.

[45] E. F. Fama and K. R. French.
The capital asset pricing model: Theory and evidence.
The Journal of Economic Perspectives, pages 25–46, 2004.

[46] E. W. Frees.
Regression Modelling with Actuarial and Financial Applications.
Cambridge University Press, New York, 2010.

[47] D. M. Gay.
Usage summary for selected optimization routines.
Computing science technical report, 153:1–21, 1990.

[48] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin.
Bayesian Data Analysis.
London: Chapman and Hall, 1995.

[49] A. Genz and R. E. Kass.
Subregion-adaptive integration of functions having a dominant peak.
Journal of Computational and Graphical Statistics, 6:92–111, 1997.

[50] W. R. Gilks and P. Wild.
Adaptive rejection sampling for Gibbs sampling.
Journal of the Royal Statistical Society. Series C (Applied Statistics), 41(2):337–348, 1992.

[51] I. J. Good.
The population frequencies of species and the estimation of population parameters.
Biometrika, 40:237–264, 1953.

[52] J. Goodman.
Bootstrap goodness of fit tests.
Honours Project Report, Department of Statistics, The University of Auckland, 2014.

[53] C. Halgreen.
Self-decomposability of the generalized inverse Gaussian and hyperbolic distributions.
Zeitschrift fur Wahrscheinlichkeitstheorie und Verwandte Gebiete, 47(1):13–17, 1979.

[54] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel.
Robust Statistics: The Approach Based on Influence Functions.
New York: John Weiley, 1986.

[55] B. E. Hansen.
Autoregressive conditional density estimation.
International Economic Review, 35(3):705–703, 1994.

[56] A. Hanssen and T. A. Oigard.
The normal inverse Gaussian distribution: A versatile model for heavy-tailed stochastic processes.
In *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, volume 6, pages 3985–3988, 2001.

[57] N. J. Higham.
Computing a nearest symmetric positive semidefinite matrix.
Linear Algebra and its Applications, 103:103–118, 5 1988.

[58] H. J. Ho and T.-I. Lin.
Robust linear mixed models using the skew t distribution with application to schizophrenia data.
Biometrical Journal, 52(4):449–469, 2010.

[59] W. Hörmann.
A rejection technique for sampling from T-concave distributions.
ACM Transactions on Mathematical Software, 21(2), 6 1995.

[60] W. Hörmann.
Generating generalized inverse Gaussian distributed random variates.
8th World Congress of the Bernoulli Society, Istanbul, Turkey, 2012.

[61] P. J. Huber.
Robust estimation of a location parameter.
The Annals of Mathematical Statistics, 35(1):73–101, 1964.

[62] P. J. Huber.
The 1972 wald lecture robust statistics: A review.
The Annals of Mathematical Statistics, 43:1041–1067, 1972.

[63] P. J. Huber.
Robust Statistics.
New York: John Weiley, 1981.

[64] S. Iyengar and Q. Liao.
Modeling neural activity using the generalized inverse Gaussian distribution.
Biological Cybernetics, 77(4):289–295, 1997.

[65] J. A. John and N. R. Draper.
An alternative family of transformation.
Journal of the Royal Statistical Society. Series C (Applied Statistics), 29:190–197, 1980.

[66] M. C. Jones and M. J. Faddy.
A skew extension of the t-distribution, with applications.
Journal of the Royal Statistical Society. Series B (Statistical Methodology), 65(1):159–174, 2003.

[67] B. Jørgensen.
Statistical Properties of the Generalized Inverse Gaussian Distribution, volume 9.
Springer New York, 1982.

[68] A. Kalemanova, B. Schmid, and R. Werner.
The normal inverse Gaussian distribution for synthetic CDO pricing.
The journal of derivatives, 14(3):80–94, 2007.

[69] D. Karlis.
An EM type algorithm for maximum likelihood estimation of the normal inverse Gaussian distribution.
Statistics & Probability Letters, 57(1):43–52, 2002.

[70] A. J. Kinderman and J. F. Monahan.
Computer generation of random variables using the ratio of uniform deviates.
ACM Transactions on Mathematical Software, 3(3), 9 1977.

[71] A. J. Kinderman and J. G. Ramage.

Computer generation of normal random variables.
Journal of the American Statistical Association, 71:893–896, 1976.

[72] S. Kotz, N. L. Johnson, and N. Balakrishnan.
Continuous Univariate Distributions, volume 1.
John Wiley & Sons, 2 edition, 1994.

[73] S. Kotz, N. L. Johnson, and N. Balakrishnan.
Continuous Univariate Distributions, volume 2.
John Wiley & Sons, 2 edition, 1995.

[74] U. Küchler, N. K., M. Sørensen, and A. Streller.
Stock returns and hyperbolic distributions.
Mathematical and Computer Modelling, 29:1–15, 1999.

[75] K. Lange.
Numerical Analysis for Statisticians, chapter 11.
New York: Springer, 2010.

[76] K. L. Lange, R. J. A. Little, and J. M. G. Taylor.
Robust statistical modelling using the t distribution.
Journal of the American Statistical Association, 84, 1989.

[77] J. Leydold and W. Hörmann.
UNU.RAN User Manual, 10 2010.
Version 1.8.0.

[78] J. Leydold and W. Hörmann.
Generating generalized inverse Gaussian random variates by fast inversion.
Computational Statistics & Data Analysis, 55(1):213–217, 2011.

[79] J. Leydold and W. Hörmann.
Generating generalized inverse Gaussian random variates.
Statistics and Computing, pages 1–11, 2013.

[80] J. Leydold and W. Hörmann.
R Interface to the UNU.RAN Random Variate Generators, 8 2014.
R package version 0.21.0.

[81] T. Lin, J. Lee, and W. Hsieh.
Robust mixture modeling using the skew t distribution.
Statistics and Computing, 17(2):81–92, 2007.

[82] J. Mencia and E. Sentana.
Estimation and testing of dynamic models with generalized hyperbolic innovations.
CEPR Discussion Paper, 2005.

[83] P. A. P. Moran.
The random division of an interval–part ii.
Journal of the Royal Statistical Society. Series B (Methodological), 13(1):147–150, 1951.

[84] T. B. of Maximum Likelihood Estimates Under Nonstandard Conditions.
Huber, peter j.
In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 221–233, 1967.

[85] M. S. Paolella.
Intermediate Probability: A Computational Approach.
Wiley, Chichester, 2007.
ISBN 978-0-470-02637-3.

[86] K. Prause.
The Generalized Hyperbolic Model: Estimation, Financial Derivatives, and Risk Measures.
PhD thesis, Universität Freiburg, 1999.

[87] W. J. Reed.
The normal-laplace distribution and its relatives.
In *Advances in Distribution Theory, Order Statistics, and Inference*. Birkhäuser Boston, 2006.

[88] T. H. Rydberg.
The normal inverse Gaussian levy process: Simulation and approximation.
Communications in Statistics. Stochastic Models, 13(4):887–910, 1997.

[89] T. H. Rydberg.
Generalized hyperbolic diffusion processes with applications in finance.
Mathematical Finance, 9(2):183–201, 1999.

[90] R. B. Schnabel, J. E. Koontz, and B. E. Weiss.
A modular system of algorithms for unconstrained minimization.
ACM Transactions on Mathematical Software, 11(4):419–440, 12 1985.

[91] D. J. Scott.
The Generalized Hyperbolic Distribution, 2010.
R package version 0.2-0.

[92] D. J. Scott and C. Y. Dong.
The Variance Gamma Distribution, 2010.
R package version 0.3-0.

[93] D. J. Scott and J. S. Fu.
The Normal Laplace Distribution, 2010.
R package version 0.1-1.

[94] D. J. Scott and F. Grimson.
The Skew Hyperbolic Student t*-Distribution*, 2013.
R package version 0.2-0.

[95] D. J. Scott, D. Würtz, and C. Y. Dong.
Software for Distributions in R.
UseR: The R User Conference 2009, July 2009.

[96] D. J. Scott, D. Würtz, and Y. Chalabi.
Fitting the hyperbolic distribution with r: A case study of optimization techniques.
In preparation, 2011.

[97] D. J. Scott, D. Würtz, C. Dong, and T. T. Tran.
Moments of the generalized hyperbolic distribution.
Computational Statistics, 26:459–476, 2011.

[98] D. J. Scott, M. Slevinsky, H. Safouhi, and X. Li.
Tail probabilities using the G transformation, 11 2012.

[99] A. Shapiro and M. W. Browne.
Analysis of covariance structures under elliptical distributions.
Journal of the American Statistical Association, 82:1092–1097, 1987.

[100] H. S. Sichel.
On a distribution representing sentence-length in written prose.
Journal of the Royal Statistical Society. Series A (General), 137:25–44, 1974.

[101] R. M. Slevinsky and H. Safouhi.
New formulae for higher order derivatives and applications.
J. Comp. Appl. Math., 233:405–419, 2009.

[102] R. M. Slevinsky and H. Safouhi.
A recursive algorithm for the G transformation and accurate computation of incomplete Bessel functions.
Appl. Numer. Math., 2010.
In press.

[103] R. M. Slevinsky and H. Safouhi.
Computation of tail probabilities of the generalized inverse Gaussian distribution, 2010.

[104] E. Stadlober.
Sampling from Poisson, Binomial and Hypergeometric Distributions: Ratio of Uniforms as a Simple and Fast Alternative.
Mathematisch-Statistische Sektion, Forschungsges. Joanneum, 1989.

[105] L. Stentoft.
American option pricing using GARCH models and the normal inverse Gaussian distribution.
Journal of Financial Econometrics, 6(4):540–582, 2008.

[106] R. Trendall.
Random number generators for the hyperbolic distribution.
Honours Project Report, Department of Statistics, The University of Auckland, 2004.

[107] R. Trendall.
hypreg: A function for fitting a linear regression model in R with hyperbolic errors.
Master's thesis, The University of Auckland, 2005.

[108] D. E. Tyler.
Robustness and efficiency properties of scatter matrices.
Biometrika, 70:411–420, 1983.

[109] P. Wu and S. Shieh.
Value-at-Risk analysis for long-term interest rate futures: Fat-tail and long memory in return innovations.
Journal of Empirical Finance, 14(2):248–259, 2007.

[110] I. K. Yeo and R. A. Johnson.
A new family of power transformations to improve normality or symmetry.
Biometrika, 87:954–959, 2000.

[111] A. Zellner.
Bayesian and non-Bayesian analysis of the regression model with multivariate Student-t error terms.
Journal of the American Statistical Association, 71(354):400–405, 1976.

[112] D. Zhu and J. W. Galbraith.
A generalized asymmetric student t distribution with application to financial econometrics.
Journal of Econometrics, 157(2):297–305, 2010.