Probabilistic Laplace Transform Inversion

by

Zishan Huang

A Major Paper

Submitted to the Faculty of Graduate Studies

through the Department of Mathematics and Statistics

in Partial Fulfillment of the Requirements for

the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

Probabilistic Laplace Transform Inversion

by

Zishan Huang

APPROVED BY:

Dr. Abdulkadir Hussein, Reader
Department of Mathematics and Statistics

Dr. Myron Hlynka, Supervisor
Department of Mathematics and Statistics

January 21, 2016

Author's Declaration of Originality

I hereby certify that I am the sole author of this major paper.

I certify that, to the best of my knowledge, my major paper does not infringe upon anyone's copyright nor proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my major paper, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted materials that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission of the copyright owner(s) to include such materials in my major paper and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my major paper, including any final revisions, as approved by committee and the Graduate Studies Office, and that this major paper has not been submitted for a higher degree to any other University or Institution.

Abstract

In this paper, we develop a method to invert Laplace transforms using the probabilistic interpretation of Laplace transforms. We show how to overcome certain difficulties which arise when trying to apply our method.

Acknowledgments

I would like to express my profound gratitude to my supervisor Professor Hlynka for his guidance and help throughout the creation of my major paper. I also want to show my thanks to him for his academic inspiration and for the patience he always showed to me throughout my study at the University of Windsor.

I would like to thank my family and friends for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Definition and Properties of Laplace transforms

Most of the material in this chapter can be found directly in (2) and (3)

Laplace Transforms have broad applications in Classical Control Theory, Mathematics, Mechanical and Electrical Engineering, and Physics. Mainly, they are considered as an efficient way of solving differential equations. Since many sorts of problems in science and engineering can be reduced to differential equations, this makes Laplace Transforms an extremely applicable tool.

1.2 Definition of Laplace Transform

Most of the material in this chapter can be found in (2) and (3).

Definition 1.2.1. *If X is a continuous, non-negative random variable, the Laplace*

Transform of the probability density $f(x)$ with non-negative support , is denoted by $L(s)$ and is given by

$$fL(s)=\int_0^\infty e^{-sx} f(x)\ \mathrm{d}x.$$

$$\textit{where } x > 0.$$

Definition 1.2.2. *The Laplace-Stieltjes Transform of a function $f(x)$ is denoted by $L(s)$ and is given by*

$$L(s) = \int_0^\infty e^{-sx}\ \mathrm{d}F(x).$$

Since we will be focusing on Laplace transforms of p.d.f.'s with non-negative support, we will discuss the convergence of the Laplace transform in this case.

Property 1.2.3. *If f(x) is a p.d.f. with non-negative support, then the improper integral*

$$\int_0^\infty e^{-sx} f(x)\, dx$$

converges uniformly for all $s \geq 0$.

Proof Since $f(x) \geq 0$ for $x \geq 0$ and $1 \geq e^{-sx} \geq 0$ for all $x \geq 0$ and $s \geq 0$, the integral $e^{-sx} f(x) \geq 0$ for $x \geq 0$ and $s \geq 0$. Also, $\int_0^t f(x)\, dx \geq \int_0^t e^{-sx} f(x)\, dx$. It follows that $\int_0^t e^{-sx} f(x)\, dx$ is increasing in t and bounded by 1. Therefore, the integral converges uniformly for all $s \geq 0$. ∎

This result shows that the Laplace transform of all p.d.f.'s with non-negative support, exists for all $s \geq 0$. Further. it must be the case that

$$1 \geq f^*(s) \geq 0 \text{ for all } s \geq 0.$$

We also interpret the Laplace transform of probability density functions as the probability that the corresponding random variable wins a "race" against an exponentially distributed catastrophe.

Our interest is in a process which generates events where the time until the next event has p.d.f. $f(x)$. To calculate the Laplace transform of f(x), consider an independent process that generates "catastrophes" (a catastrophe is simply another type of random variable) . If the time between catastrophes is distributed as an random variable with rate s, we find that the Laplace transform of the distribution of the time until the next event, $L(s)$ is simply the long-term proportion of time that the event occurs before the catastrophe. This result is summarized in the following theorem.

Theorem 1.2.4. *Let X and Y be independent random variables. Further, suppose that $Y \sim exp(s)$ and the p. d.f. of X is $f(x)$. Then,*

$$L(s)=P(Y > X)$$

Proof

$$\begin{aligned} L(s) &= \int_0^\infty f(x)e^{-sx}dx \\ &= \int_0^\infty \int_x^\infty f(x)se^{-sy}dydx \\ &= \int_0^\infty \int_x^\infty f(x)g(y)dydx \\ &= P(X < Y) \quad \blacksquare \end{aligned}$$

It is worth noting that since the time until the next catastrophe is exponential, the catastrophe process is Poisson and memoryless. Note that $L(s)$ is the expected value of e^{-sX}. The moment generating function $M_X(t)$ (m.g.f.) of a random variable X can be obtained from the Laplace transform by replacing s by $-t$.

Property 1.2.5. $f_X^*(s) = M_X(-t)$

Proof

$$L(s) = E_X(e^{-sX}) = M_X(-t) \quad \blacksquare$$

1.3 Laplace transforms of exponential distributions

Definition 1.3.1. *Let $X \sim exp(\lambda), \lambda > 0$, where λ is the rate. Its p.d.f. is given by*

$$f(x) = \lambda e^{-\lambda x}, x \geq 0, \lambda > 0,$$

The c.d.f. is given by

$$F(x) = 1 - e^{-\lambda x}, x \geq 0, \lambda > 0.$$

Property 1.3.2 (Memorylessness)**.** *If $X \sim exp(\lambda)$, then for $\forall s, t \geq 0$,*

$P(X > s + t \mid X > s) = P(X > t)$.

Proof

$$\begin{aligned} P(X > s+t \mid X > s) &= \frac{P(X > s+t)}{P(X > s)} \\ &= \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} \\ &= e^{-\lambda t} \\ &= P(X > t) \end{aligned}$$

Property 1.3.3. *If $X \sim ex(\lambda_1), Y \sim ex(\lambda_2)$ where X and Y are independent, then $P(Y > X) = \frac{\lambda_1}{\lambda_1+\lambda_2}$*

Proof

$$\begin{aligned} P(Y > X) &= \int_0^\infty \int_x^\infty \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 y} dy dx \\ &= \int_0^\infty -\lambda_1 e^{-(\lambda_1+\lambda_2)x} dx \\ &= \frac{\lambda_1}{\lambda_1+\lambda_2} \end{aligned}$$

Property 1.3.4. *If $X \sim ex(\lambda_1)$ and $Y \sim ex(\lambda_2)$ where X and Y are independent, then $min(X, Y) \sim ex(\lambda_1 + \lambda_2)$*

Proof Let $X \sim ex(\lambda_1)$ and $Y \sim ex(\lambda_2)$ and $Z = min(X, Y)$. The cumulative

distribution function of Z is

$$\begin{aligned} F_z(z) &= P(Z \leq z) \\ &= P(min(X,Y) \leq z) \\ &= 1 - P(min(X,Y) > z) \\ &= 1 - P(X > z, Y > z) \\ &= 1 - P(X > z)P(Y > z) \\ &= 1 - e^{-\lambda_1 z} e^{-\lambda_2 z} \\ &= 1 - e^{-(\lambda_1 + \lambda_2)z} \end{aligned}$$

which we recognize as the cumulative distribution function of an exponential random variable with rate $\lambda_1 + \lambda_2$. ∎

With this randomness in mind, we have an intuitive explanation for $P(Y > X)$ where X and Y are exponential random variables. If there are type 1 event s occurring with exponential inter-event times at rate λ_1 per unit time , then all together random events occur at rate $\lambda_1 + \lambda_2$. Now, since truly random events will fall uniformly on any interval, given that a known number of events have occurred in that interval, the probability that the first event is of type 1 is simply the proportion of events that are of type 1, namely $\frac{\lambda_1}{\lambda_1 + \lambda_2}$.

Chapter 2

Brief Introduction of Numerical Methods to Inverting Laplace Transform

The Laplace transform occurs frequently in investigations of queueing theory and telephone traffic models in which it usually represents a probability distribution function. Although the mean and variance of the distribution can be readily obtained from the transform, there are many situations in which the distribution itself is needed. In particular, we may need good analytic and numerical approximations for the complementary distribution when the argument is large. This is the case, for example, when studying waiting times of queues, time delays of work through a computer system, and delays of message progress through data networks.

Taken from [Computational Probability] "Numerical transform inversion has an

odd place in computational probability. Historically, transforms were exploited extensively for solving queueing and related probability models, but only rarely was numerical inversion attempted. Indeed, the conventional wisdom was that numerical transform inversion was very difficult. Even numerical analysts were often doubtful of the numerical stability of inversion algorithms. In queueing, both theorists and practitioners lamented about the "Laplace curtain".

In this section we present a way to numerically invert the LT in order to calculate $f(t)$ for any given t. We have assumed that f is a real-valued, as when f is a *pdf*, *cdf*, but inversion also applies when f is complex-valued. Indeed, if

$$f(t) = f_1(t) + if_2(t), t \geq 0$$

$$for \mathrm{i} = \sqrt{-1}$$

then, $L(s) = L_1(s) + L_2(s)$, so it is clear that inversion extends easily to complex-valued functions.

A natural starting point for numerical inversion of Laplace Transform is the Bromwich inversion integral.

Theorem 2.0.1. *(Bromwich Inversion Integral) Given the Laplace transform $L(s)$, the function value $f(t)$ can be recovered from the contour integral*

$$f(t) = \frac{1}{2\pi i} \int_{b-\infty}^{b+\infty} e^{st} L(s)\, ds$$

where b is a real number to right of all singularities of $fL(s)$, and the contour integral yields the value 0 for $t < 0$.

As usual with contour integrals, there is flexibility in choosing the contour provided that it is the right of all singularities of $L(s)$. However, there is no need to bothered by complexities of complex variables. If we choose a specific contour and perform a change of variables, then we obtain an integral of a real-valued function of a real variable. First, by marking the substitution $s = b + iu$ in

$$f(t) = \frac{1}{2\pi i} \int_{b-\infty}^{b+\infty} e^{st} L(s)\, ds,$$

we obtain

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{b+iu} L(b + iu)\, du$$

Then, since

$$e^{b+iu} = e^{bt}(\cos ut + i \sin ut)$$

, $\sin ut = -\sin(-ut), \cos ut = \cos(-ut), Im(L(b + iu)) = -Im(L(b - iu))$ and $Re(f^*(b + iu)) = Re(L(b - iu))$, and from the fact that the integral in

$$f(t) = \frac{1}{2\pi i} \int_{b-\infty}^{b+\infty} e^{st} L(s)\, ds$$

is 0 for $t < 0$, we obtain

$$f(t) = \frac{2e^{bt}}{\pi} \int_0^\infty Re(L(b+iu)) \cos(ut)\, du$$

$$f(t) = -\frac{2e^{bt}}{\pi} \int_0^\infty Im(L(b+iu)) \sin(ut)\, du$$

Above functions imply that $f(t)$ can be calculated from the transform $L(s)$ by performing a numerical integration algorithms."

Chapter 3

Inverting Laplace transform using probability interpretation

Numerical methods provide clear computational ways to invert Laplace transforms. However, the interpretation of Laplace transform using probability knowledge could be very explicit.

Given that a random variable X will occur prior to some catastrophe Y, assume the time of catastrophe follows an exponential distribution with rate s. More specifically, we can assume a situation where X and Y are two racers in a competition. Then $L(s)$ is the probability that X wins the race.

We intend to approximate the $p.d.f$ of X by a discrete probability mass function ($p.m.f$). The procedure of our method is as follows. First we pick several different values of s at random and find $L(s)$ for each value. Then choose the same number of $x's$. Since $x's$ are chosen, there is an equation corresponding to formula $L(s) = P(X < Y)$

for every value s. Thus we establish a system of linear equations. Solving the system of linear equations satisfying constraints, we can compute the probabilities of the $x's$. Based on the probabilities, we can draw a sketch of the mass function to approximate the original function.

Algorithm to Invert $L(s)$

Step 1.

Choose values $s_1, s_2, \ldots, s_n$

Step 2

Choose values x_1,$x_2, \ldots, x_n$

Step 3

Find $L(s_1), L(s_2), \ldots, L(s_n)$

Step 4

Let $p_i = P(X = x_i)$ for $i = 1, \ldots, n$ to be determined.

We know

$L(s_i) = P(X < Y)$ for $Y \sim exp(s_i)$

Then

$$L(s_1) = P(X < Y_1) = p_1 * P(x_1 < Y_1) + p_2 * P(x_2 < Y_1) + \dots$$

$$+ p_{n-1} * P(x_{n-1} < Y_1) + p_n * P(x_n < Y_1)$$

$$L(s_2) = P(X < Y_2) = p_1 * P(x_1 < Y_2) + p_2 * P(x_2 < Y_2) + \dots$$

$$+ p_{n-1} * P(x_{n-1} < Y2) + p_n * P(x_n < Y_2)$$

$$\dots$$

$$L(s_{n-1}) = P(x < Y_{n-1}) = p_1 * P(x_1 < Y_{n-1}) + p_2 * P(x_2 < Y_{n-1}) + \dots$$

$$+ p_{n-1} * P(x_{n-1} < Y_{n-1}) + p_n * P(x_n < Y_{n-1})$$

$$L(s_n) = P(X < Y_n) = p_1 * P(x_1 < Y_n) + p_2 * P(x_2 < Y_n) + \dots$$

$$+ p_{n-1} * P(x_{n-1} < Y_n) + p_n * P(x_n < Y_n)$$

Recall that if Y $\sim exp(s)$, then

$f(y) = se^{-sy}, F(y) = 1 - e^{-sy}, \bar{F}(y) = P(Y \geq y) = e^{-sy}$.

Step 5

Our system of equations is :

$$\begin{cases} L(s_1) = p_1 e^{-x_1 s_1} + \dots p_n e^{-x_n s_1} \\ L(s_2) = p_1 e^{-x_1 s_2} + \dots p_n e^{-x_n s_2} \\ \vdots \\ L(s_{n-1}) = p_1 e^{-x_1 s_{n-1}} + \dots p_n e^{-x_n s_{n-1}} \\ L(s_n) = p_1 e^{-x_1 s_n} + \dots p_n e^{-x_n s_n} \end{cases}$$

Then $p_j = P(X = x_j)$ gives an approximation for $f(x)$. ∎

We can also write the system of equations in matrix form :

$$\begin{cases} L(s_1) = \sum_{j=1}^{j=n} a_{1j} * p_j \\ L(s_2) = \sum_{j=1}^{j=n} a_{2j} * p_j \\ L(s_3) = \sum_{j=1}^{j=n} a_{3j} * p_j \\ \vdots \\ L(s_n) = \sum_{j=1}^{j=n} a_{nj} * p_j \end{cases}$$

where a_{ij} is a coefficient equal to $P(X_j < Y_i), Y_i \sim exp(s_i)$. So $a_{ij} = e^{-s_i x_j}$. A is a coefficient matrix.

$$L(s)^T = \begin{bmatrix} L(s_1) & \dots & L(s_n) \end{bmatrix}$$

$$p^T = \begin{bmatrix} p_1 & \dots & p_n \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \ddots & \vdots & \\ a_{n1} & a_{102} & \dots & a_{nn} \end{bmatrix}$$

So $L(\vec{s}) = A\vec{p}$.

Since $p_j = P(X = x_j)$ we have $\sum_{j=1}^{j=n} p_j = 1$ and $p_j \geq 0$.

It is worth mentioning a failed attempt.It seems that we can directly invert matrix A to solve for p_i. But it doesn't work.

Taking $L(s) = \frac{3}{3+s}$ as an example, we first choose 4 points to estimate the probability function.

i	1	2	3	4
x_i	0.1	0.2	0.3	0.4
s_i	1	2	3	4

We invert the coefficient matrix to solve for p. It works on 4 points. We can calculate $p = (0.56978132, 0.05099405, 0.11781196, 0.25213524)$. But when we try to get 10 points, the answers no longer satisfy the condition that $p_j = P(X = x_j)$ is a probability value. The results are as follows:

$p = (0.9068893, -5.2194752, 42.9964258, -225.4731352, 803.227295,$

-1937.3807260,3110.7217508,-3174.7380936,1857.93149,-473.29424)

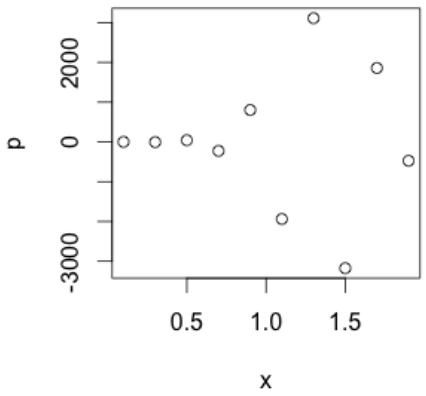

Figure 3.1: The failure example using inverse matrix

So we will convert this problem into an optimization problem.

3.1 Convex quadratic programming (QP) problem

We simplify our problem as:

$$\min ||L - Ap||^2$$

subject to :

$$\begin{cases} p_1 + p_2 + p_3 \ldots \cdots + p_n = 1 \\ p_1 \geq 0 \\ p_2 \geq 0 \\ \ldots \\ p_{n-1} \geq 0 \\ p_n \geq 0 \end{cases}$$

Now,

$$||L - Ap||^2 = (L - Ap)^T(L - Ap) = L^TL - L^TAp - p^TA^TL + p^TA^TAp$$

and since L^TL is known,

$$\begin{aligned} \arg\min_p \ ||L - Ap||^2 &= \arg\min_p(p^TA^TAp - L^TAp - p^TA^TL) \\ &= \arg\min_p(p^TA^TAp - 2L^TAp) \end{aligned}$$

Here A^TA is positive definite. So the problem becomes a standard convex quadratic programming (QP) problem.

Taken from (1).

"That is, we are concerned with problem which is to determine the n-vector x that will

$$min\left\{Q(x) = c^Tx + \tfrac{1}{2}x^THx | A_1x = b_1, A_2x < b_2\right\}$$

where c is an n-vector containing the coefficients of the linear term of the objective function, H is an (n,n) positive (n,n) positive semi-definite matrix which can also be

called Hessian matrix, containing the coefficients of the quadratic term of the objective function, A_1 is an (r,n) matrix whose $i-$th row, a_i^T, is the normal vector for the $i-$th equality constraint $a_i^T x = b_i$, b_1 is an r-vector containing the right hand sides for the equality constraints, A_2 is an (m,n) matrix whose $i-$th row, a_{r+i}^T, is the normal vector for the $i-$th inequality constraint $a_{r+i}^T x = b_{r+i}$, b_2 is an m-vector containing the right hand sides for the inequality constraints. We assume that rank $A_1 = r$ and we note that $A_1^T = [a_1, a_2, \ldots a_r]$. We can also write $b_1 = (b_1, b_2, \ldots b_r)^T$ being sure not to confuse the vector b_1 with its first component b_1. The content will always make the distinction clear. We can write $A_2^T = [a_{r+1}, a_{r+2}, \ldots, a_{r+m}]$ and we note that the $i-$th inequality constraint is actually the $(r+1)-$st constraint, we also write $b_2 = (b_{r+1}, b_{r+2}, \ldots, b_{r+m})^T$."

"The problem is to minimize $Q(x)$ that is, to find that minimum value of $Q(x)$ a quadratic function of n variables, where the n variables must satisfy r linear equality constraints as well as m linear inequality constraints. We call $Q(x)$ the objective function and we call R the feasible region, where

$$R = \{x \in R^n | A_1 x = b_1, A_2 x b_2\} \text{ "}$$

All quadratic programming problems can be written in this general form. So we can deduce that $c = -2L^T A$, $H = 2A^T A$, $x = p$ corresponding to the standard form.

3.2 Generalized Reduced Gradient Algorithm

"Solver" in EXCEL provides a desired solution to this problem. The algorithm employed in "Solver" in EXCEL is "Generalized reduced gradient method". The Generalized reduced gradient (GRG) is a generalization of the reduced gradient method by allowing nonlinear constraints and arbitrary bounds on the variables. Taken from (5) "The form is:

$$\{maxf(x) : h(x) = 0, L \leq x \leq U\} \text{ where } h \text{ has dimension } m.$$

The method supposes we can partition $x = (v, w)$ such that: v has dimension m(and w has dimension $n - m$); the values of v are strictly within their bounds: $L_v \leq v \leq U_v$(this is a nondegeneracy assumption); $h(x)$ is nonsingular at $x = (v, m)$. As in the linear case, for any w there is unique value, $v(w)$, such that $h(v(w), w) = 0$ (c.f., Implicit Function Theorem), which implies that $\frac{dv}{dw} = (\dot{vh}(x)^{-1}\dot{wh}(x))$. The idea is to choose the direction of the independent variables to be the reduced gradient: $\dot{w(f(x)} - y^T h(x))$, where $y = \frac{dv}{dw} = (\dot{vh}(x)^{-1}\dot{wh}(x))$. Then, the step size is chosen and a correction procedure applied to return to the surface, $h(x) = 0$.

Theorem 3.2.1. *(Implicit Function Theorem): Suppose:h: $R^n \to R^m$, where $n > m$, and h is in smooth. Further, suppose we can partition the variables, $x = (y, z)$, such that y is m-dimensional with $\dot{yh}(x)$ nonsingular at $x^* = (y^*, z^*)$. Then, there exists $\varepsilon > 0$ for which there is an implicit function, f, on the neighborhood, $N_\varepsilon(z^* = z : ||z - z^*||e)$ such $zf(z^*) = -(\dot{yh}(x^*))^{\dot{-1}}z\dot{h}(z^*)$. "*

3.3 Best-Ritter Algorithm

The following pages in this chapter are taken from (1) "We are interested in the parametric quadratic programming problem (QQP) given by

$$min\left\{Q(x) = c^T x + \tfrac{2}{1} x^T H x | A_1 x = b_1, A_2 x < b_2\right\}$$

We now provide the basic definitions.

Definition 3.3.1. *The Hessian matrix of $Q(x)$ is the matrix of second derivatives and is given by the (n, n) matrix H. The matrix H can always be taken as symmetric since, if H is not symmetric, then $x^T H x$ can be replaced with the equivalent term $x^T[\frac{1}{2}(H + H^T)]x$ and $\frac{1}{2}(H + H^T)$is symmetric.*

Definition 3.3.2. *The (n, n) symmetric matrix H is said to be positive definite if $x^T H x > 0$, $\forall x \neq 0$. It is positive semi-definite if $x^T H x \geq 0$, $\forall x$. A matrix is positive definite if and only if all of its eigenvalues are positive and H is positive semi-definite if and only if it has only non-negative eigenvalues. The function $Q(x)$ is convex if and only if H is positive semi-definite. The function $Q(x)$ is strictly convex if and only if H is positive definite.*

Definition 3.3.3. *The $i-th$ constraint is said to be active at if $a_i^T x^T = b_i$. The inequality constraint $a_i^T x \leq b_i$ is said to be violated at $\bar{x}$ if $a_i^T \bar{x} > b_i$. The inequality constraint $a_i^T x \leq b_i$ is said to be satisfied at $\bar{x} if\, a_i^T x \leq b_i$. The equality constraint $a_i^T x = b_i$ is violated at $\bar{x}$ if $a_i^T x = b_i$,and is satisfied at $\bar{x}$ if $a_i^T x = b_i$. If $\bar{x} \in R$,then all of the equality constraints are active.*

We start each iteration of the algorithm with an x_j. If x_j is not optimal (i.e., not a minimizer) we want to generate a direction vector s_j , calculate a non-negative step size σ_j and calculate the next iterate using the formula $x_{j+1} = x_j - \sigma_j s_j$. Since $\sigma_j \geq 0$, the actual direction of search is $-s_j$. For an algorithm to be a descent method we want to have $Q(x_{j+1}) \leq Q(x_j)$. To achieve this goal, we pick a search direction that is a descent direction. For convenience, we introduce the notation $g_j = \Delta Q(x_j) = c + Hx_j$ and $Q_j = Q(x_j)$.

Theorem 3.3.4. *Descent Direction.*

If $g_j^T s_j > 0$, then there exists a $\delta > 0$ such that $Q(x_j - \sigma s_j) \leq Q_j$ for all $0 \leq \sigma_j \leq \delta$.

Proof *From the definition of derivative $g_j^T(-s_j) = \lim_{\sigma \to 0} \frac{Q(x_j - \sigma s_j) - Q_j}{\sigma}$. Since $g_j^T(-s_j) < 0$ and $\sigma > 0$ then the definition of limit establishes the existence of a $\delta > 0$ with $\frac{Q(x_j - \sigma s_j) - Q_j}{\sigma} < 0 \Leftrightarrow Q(x_j - \sigma s_j) < Q_j, 0 \leq \sigma \leq \delta$* ∎

Definition 3.3.5. *We say that s_j is a descent direction of $Q(x)$ at x_j (in the direction $-s_j$) if $g_j^T s_j > 0$. Before we can calculate x_{j+1}, however, we need to determine the step size σ_j.*

Definition 3.3.6. *Given x_j and a descent direction s_j the optimal step size σ_j is defined by $Q(x_j - \sigma - js_j) = \min \sigma Q(x_j - \sigma sj)$.*

Theorem 3.3.7. *The Optimal Step Size.*

Given x_j and a descent direction s_j the optimal step size is

$$\sigma_j = \frac{g_j^T s_j}{s_j^T H s_j},$$

provided, of course, that $s_j^T H s_j \neq 0$.

Proof *Using Taylor's theorem, we have*

$$Q(x_j - {}_j) = Q_j - (g_j^T s_j)\sigma + (\frac{s_j^T H s_j^T}{2})\sigma^2.$$

This is a function of a single variable, and we set the derivative to zero giving

$$\frac{\mathrm{d}Q(x_j - \sigma s_j)}{\mathrm{d}\sigma} = 0 \Longleftrightarrow \sigma = \frac{g_j^T s_j}{s_j^T H s_j^T},$$

providing, of course, that $s_j^T H s_j \neq 0$. *Now, we also have, since* H *is positive semi-definite,* $s_j^T H s_j > 0$ *so that the value of* σ_j *gives the required minimum value of* $Q(x_j - \sigma s_j)$.

Theorem 3.3.8. *Conjugate Direction Methods.*

Let $x_0 \in R^n$ *be arbitrary but fixed and let* $s_0, s_1, ..., s_{n-1}$ *be a set of* H *conjugate directions where* H *is positive definite. Define* $x_{j+1} = x_j - \sigma_j s_j$ *for* $j = 0, ..., n-1$, *where* σ_j *is the optimal step size. We assume, without loss of generality that* $g_j^T s_j > 0$. *(If* $gj^T sj < 0$ *we can replace* s_j *with* $-s_j$.*) Then* $g_j = 0$ *for some* $j \leq n$.

Proof *If* $g_j = 0$ *for some* $j \leq n$ *then we are done. Suppose not. From the optimal step size property we have that* $g_n^T s_{n-1} = 0$. *We have, for* $j = 0, ..., n-2$

$$g_n^T s_j = (g_{j+1} - \sum_{i=j+1}^{n-1} \sigma_i H s_i)^T s_j = g_{j+1}^T s_j - \sum_{i=j+1}^{n-1} \sigma_i s_i^T H s_j = 0.$$

The equality to zero is obtained using the optimal step size property and the property

of conjugate directions. Consequently, g_n is orthogonal to all of the n conjugate vectors. Since conjugate vectors are linearly independent, we conclude that $gn = 0$. The proof uses the equality

$$g_n = g_{j+1} - \sum_{i=j+1}^{n-1} \sigma_i H s_i.$$

This equality is easily established using the iteration formula and Taylors theorem for quadratics. We have

$$\begin{aligned} g_n =& g_{n-1} + H(x_n - x_{n-1}) \\ =& g_{n-1} - \sigma_{n-1} H s_{n-1} \\ =& g_{n-2} + H(x_{n-1} - x_{n-2}) - \sigma_{n-1} H s_{n-1} \\ =& g_{n-2} - \sigma_{n-2} H s_{n-2} - \sigma_{n-1} H s_{s_{n-1}} \\ =& g_{j+1} - \sum_{i=j+1}^{n-1} \sigma_i H s_i \end{aligned}$$

At a general iteration of the Best-Ritter algorithm we have

$$\begin{aligned} D_j^T &= (d_{1j}, \ldots, d_{k-1}j, d_{kj}, d_{k+1}j, \ldots, d_{nj}), \\ D_j^{-1} &= (c_{1j}, \ldots, c_{k-1}j, c_{kj}, c_{k+1}j, \ldots, c_{nj}) and \\ J_j &= \{\alpha_{1j}, \ldots, \alpha_{k-1}j, \alpha_{kj}, \alpha_{k+1}j, \ldots, \alpha_{nj}\} \end{aligned}$$

where

$$y = \begin{cases} -0, & \text{if } c_{ij} \text{ is a free column} \\ -1, & \text{if } c_{ij}\text{is a conjugate direction column} \\ > 0, & \text{if } c_{ij} \text{ corresponds to constraint } \alpha_{ij}. \end{cases}$$

We describe what is meant by the above description on J_j. If $\alpha_{ij} > 0$ we say that c_{ij} corresponds to constraint α_{ij}.That means that the corresponding column in D_j^T is

$$d_{ij} = a_{\alpha_{ij}}.$$

If $\alpha_{ij} = -1$ we say that c_{ij} is a conjugate direction. That means that corresponding column in D_j^T is

$$d_{ij} = \frac{Hc_{ij}}{\sqrt{c_{ij}^T H c_{ij}^T}}.$$

If $\alpha_{ij} = 0$ then there is nothing special about either c_{ij} or d_{ij} and we call it a free column.

The thing to remember, which helps in the understanding of the algorithm, is that $D_j D_j^{-1} = I$. Above we give D_j^T so column d_{ij} of D_j^T is row d_{ij}^T of D_j. So we have

$$d_{ij}^T = \begin{cases} -0 & \text{if } i \neq j \\ 1 & \text{if } i = j. \end{cases}$$

Algorithm: The Best-Ritter Algorithm for Convex QPs.

We start with $x_0 \in R$, a nonsingular (n, n) matrix D_0^{-1} and index set $J_0 = \{\alpha_{10}, \alpha_{20}, \ldots, \alpha_{n0}\} \supset \{1, 2, ..., r\}$. We assume that there are no degenerate

quasi-stationary points.

Step 1.1: Search a free column.

If $\alpha_{ij} \neq 0$ for i = 1,. . . , n, then go to Step 1.2 as x_j is a quasi-stationary point. Otherwise, let k be the smallest integer such that

$$|g_j^T c_{kj}| \geq \left\{|g_j^T c_{kj}| : \forall \text{ i with } \alpha_{ij} = 0\right.$$

If $g_j^T c_{kj} \neq 0$ set

$$s_j = \begin{cases} c_{kj}, & \text{if } g_j^T c_{kj} \geq 0 \\ -c_{kj}, & \text{if } g_j^T c_{kj} \leq 0. \end{cases}$$

and go to Step 2. Otherwise go to Step 1.2

Step1.2: Drop a constraint.

If there is no i with $r+1 \leq \alpha_{ij} \leq r+m$ go to Step4. Otherwise, let k be the smallest integer with$r+1 \leq \alpha_{kj} \leq r+m$ and $-g_j^T c_{kj} \leq -g_j^T c_{ij}$ for all i with $r+1 \leq \alpha_{ij} \leq r+m$. If$-g_j^T c_{kj} \geq 0$, then go to Step 4. Otherwise, set $s_j = c_{kj}$ and go to Step 2.

Step 2: Step Size.

If $s_j^T H s_j = 0$, set $\tilde{\sigma}_j = +\infty$. Otherwise, set

$$\tilde{\sigma}_j = \frac{g_j^T s_j}{s_j^T H s_j}.$$

If $a_i^T s_j \geq 0$ for all $i = r+1, \ldots, r+m$, set $\hat{\sigma}_j = +\infty$. Otherwise, let l be the

smallest integer such that

$$
\begin{aligned}
_j &= \frac{a_l^T x_j - b_l}{a_l^T s_j} \\
&= \min_{i=r+1,\ldots,r+m} \left\{ \frac{a_i^T x_j - b_i}{a_i^T s_j} | a_i^T s_j < 0 \right\}
\end{aligned}
$$

If $\hat{\sigma} = \tilde{\sigma} = +\infty$, then stop as the QP is unbounded from below. Otherwise, and go to Step 3 with $\sigma_j = min\{\hat{\sigma_j}, \tilde{\sigma_j}\}$.

Step 3: Update.

Set $x_{j+1} = x_j - \sigma_j s_j$ and compute $D_{j+1}^{-1} = [c_{1j+1}, c_{2j+1}, \ldots, c_{nj+1}]$ and $J_{j+1} = \{\alpha_{1j+1}, \alpha_{2j+1}, \ldots, \alpha_{nj+1}\}$ as follows. If $\sigma_j = \tilde{\sigma_j} \leq \hat{\sigma_j}$, go to Step 3.1. Otherwise, go to Step 3.2.

Step 3.1: Conjugate direction update.

$$
\begin{aligned}
\alpha_{kj+1} &= \text{-1 and } \alpha_{ij+1} = \alpha_{ij}, \forall i \neq k \\
c_{ij+1} &= c_{ij}, \forall \text{i such that } \alpha_{ij} = -1 \\
c_{kj+1} &= \frac{1}{\sqrt{c_{kj}^T H c_{kj}}} c_{kj} \\
c_{ij+1} &= c_{ij} - \frac{c_{kj}^T H c_{ij}}{c_{kj}^T H c_{kj}} c_{kj}, \forall i \neq k \text{ with } \alpha_{ij} \geq 0.
\end{aligned}
$$

Compute $g_{j+1} = c + Hx_{j+1}$. Replace j with $j+1$ and go to Step 1.1

Step 3.2: Adding a constraint.

$$\alpha_{kj+1} = l$$

$$\alpha_{ij+1} = \alpha_{ij}, \forall i \neq k \text{ and } \alpha_{ij} \neq -1$$

$$\alpha_{ij+1} = 0, \forall \text{ i with } \alpha_{ij} = -1$$

$$c_{kj+1} = (a_l^T c_{kj})^{-1} c_{kj}$$

$$c_{ij+1} = c_{ij} - \frac{a_l^T c_{ij}}{a_l^T c_{kj}} c_{kj}, \forall i \neq k.$$

Compute $g_{j+1} = c + Hx_{j+1}$. Replace j with $j+1$ and go to Step 1.

Step 4: The Dual Variables.

Set $u_{\alpha ij} = -g^T c_{ij}, \forall$ i with $\alpha_{ij} > 0.''$

We can plot a procedure for this algorithm.

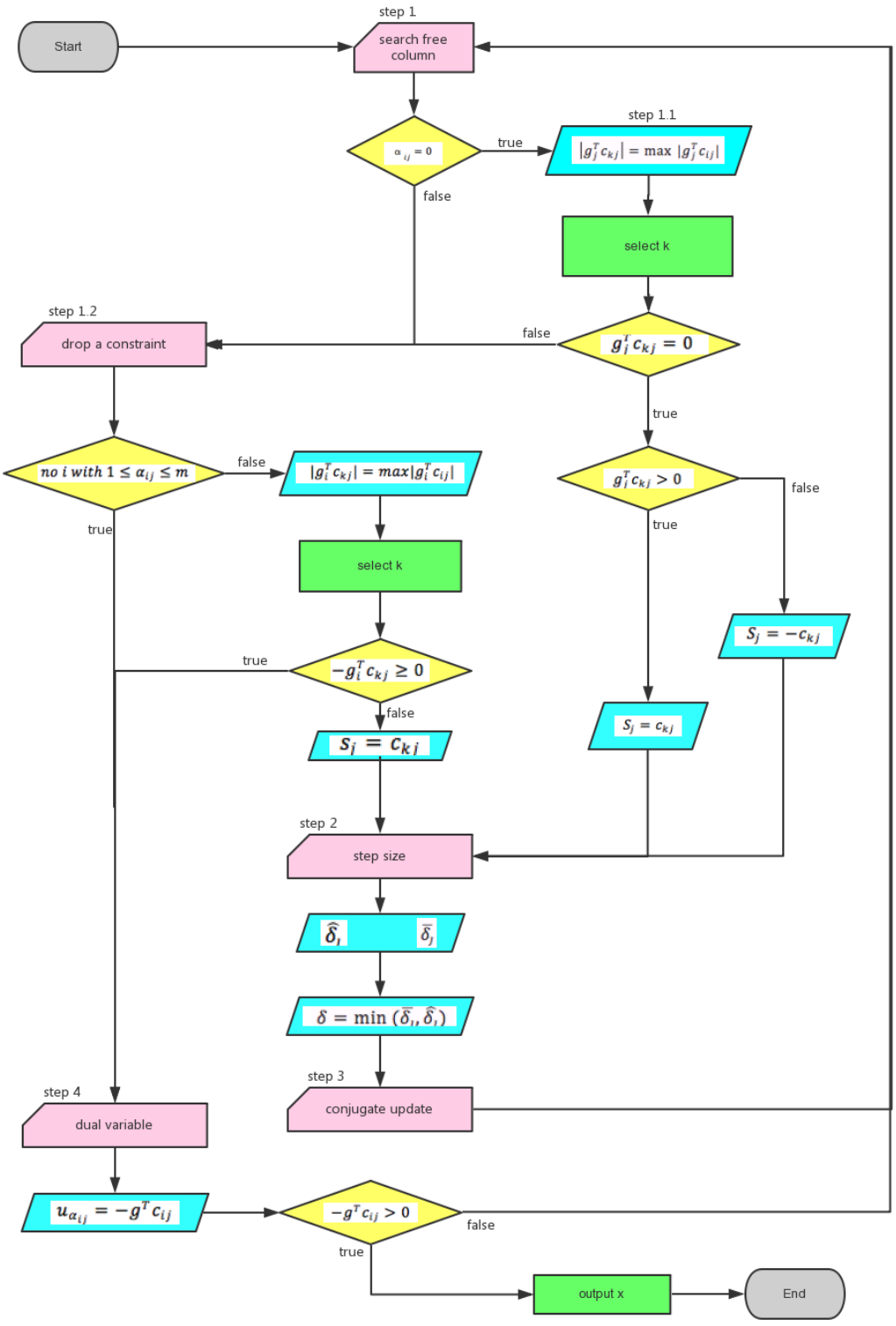

Figure 3.2: Procedure Plot

Chapter 4

Numerical Computations

We will consider two examples.

First we consider the Laplace transform of an exponential distribution. The L.T of X, where $X \sim ex(3)$, is $L(s) = \dfrac{3}{3+s}$

Another example is the Laplace transform L.T of $max(X1, X2)$, where X_1 $\sim$exp(0.3),X_2 $\sim$exp(0.5).

Then $L(s) = \dfrac{.3 * .5}{.3 + .5 + s} * (\dfrac{1}{.3+s} + \dfrac{1}{.5+s})$

4.1 Generalized Reduced Gradient Algorithm

4.1.1 Example 1

Because for the exponential distribution, we know that :

$$E(X) = \int_0^\infty x\lambda e^{-\lambda x}\,dx = \lambda[\frac{-xe^{-\lambda x}}{\lambda}\big|_0^\infty + \tfrac{1}{\lambda}\int_0^\infty e^{-\lambda x}\,dx]$$
$$= \lambda[0 + \frac{1}{\lambda}\frac{-e^{-\lambda x}}{\lambda}|_0^\infty] = \lambda\frac{1}{\lambda^2} = \frac{1}{\lambda} = \frac{1}{3}$$

$$MomentGeneratingFunction\phi(t):$$

$$\phi(t) = E[e^{tx}] = \frac{\lambda}{\lambda - t}, \text{ where } t < \lambda.$$
$$E(X^2) = \frac{d^2}{dt^2}\phi(t)|_{t=0} = \frac{2}{\lambda^2}$$
$$Var(X) = E(X^2) - (E(X))^2 = \frac{1}{\lambda^2} = \frac{1}{9}$$

According to chebyshev's theorem, we know that at least $\frac{8}{9}$ points are concentrated on $(\mu - 3*\sigma,\ \mu + 3*\sigma)$ since $P(|X - \mu| < k\sigma) > 1 - \frac{1}{k^2}$. Thus, we set six out of the ten points of $x's$ ranging from $(0\ ,\ \mu + 3*\sigma) = (0,\ 4/3)$, and choose the other four points ranging side out the bound.

i	1	2	3	4	5	6	7	8	9	10
s_i	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
x_i	0.1	0.2	0.4	0.6	0.8	1.2	2	4	6	8

Table 4.1: Coefficient Matrix for example 1

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	0.9990	0.9980	0.9970	0.9960	0.9950	0.9940	0.9930	0.9920	0.9910
1	0.9980	0.9960	0.9940	0.9920	0.9900	0.9880	0.9861	0.9841	0.9822
1	0.9960	0.9920	0.9881	0.9841	0.9802	0.9763	0.9724	0.9685	0.9646
1	0.9940	0.9881	0.9822	0.9763	0.9704	0.9646	0.9589	0.9531	0.9474
1	0.9920	0.9841	0.9763	0.9685	0.9608	0.9531	0.9455	0.9380	0.9305
1	0.9881	0.9763	0.9646	0.9531	0.9418	0.9305	0.9194	0.9085	0.8976
1	0.9802	0.9608	0.9418	0.9231	0.9048	0.8869	0.8694	0.8521	0.8353
1	0.9608	0.9231	0.8869	0.8521	0.8187	0.7866	0.7558	0.7261	0.6977
1	0.9418	0.8869	0.8353	0.7866	0.7408	0.6977	0.6570	0.6188	0.5827
1	0.9231	0.8521	0.7866	0.7261	0.6703	0.6188	0.5712	0.5273	0.4868

We assume a set of initial points: $p = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)$, then calculate using Solver,

p=(0.2876, 0.2597, 0.2047, 0.1505, 0.0974, 7.7E-07, 7.5E-07, 7.0E-07, 3.3E-06, 5.6E-06).

From following plot, we can see that the values of $x's$ are not equally spaced, so we need to adjust the p values.

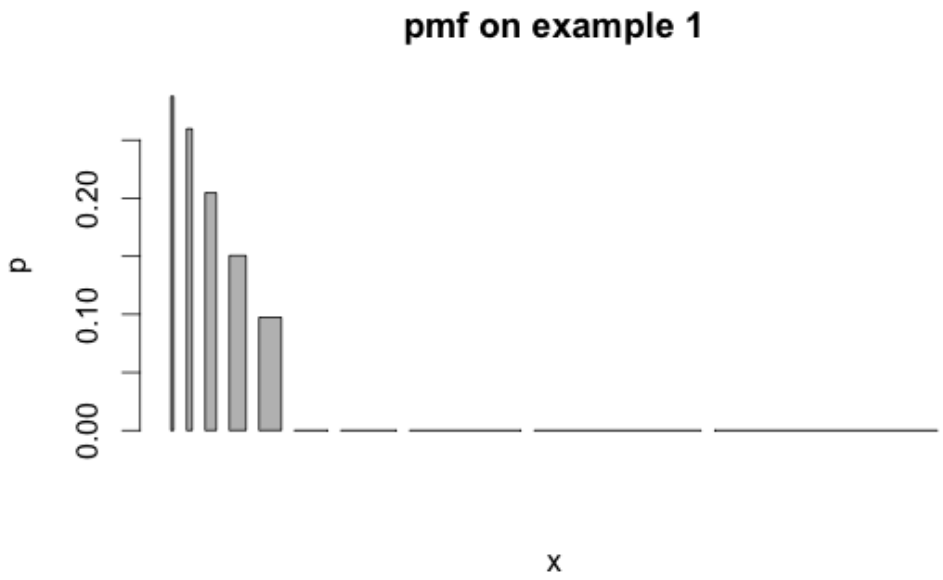

Figure 4.1: pmf of example1

From these three plots, we can see "GRG" method works efficiently on this problem. And we calculate the minimal difference is $4.27E-10$. This value is pretty ideal.

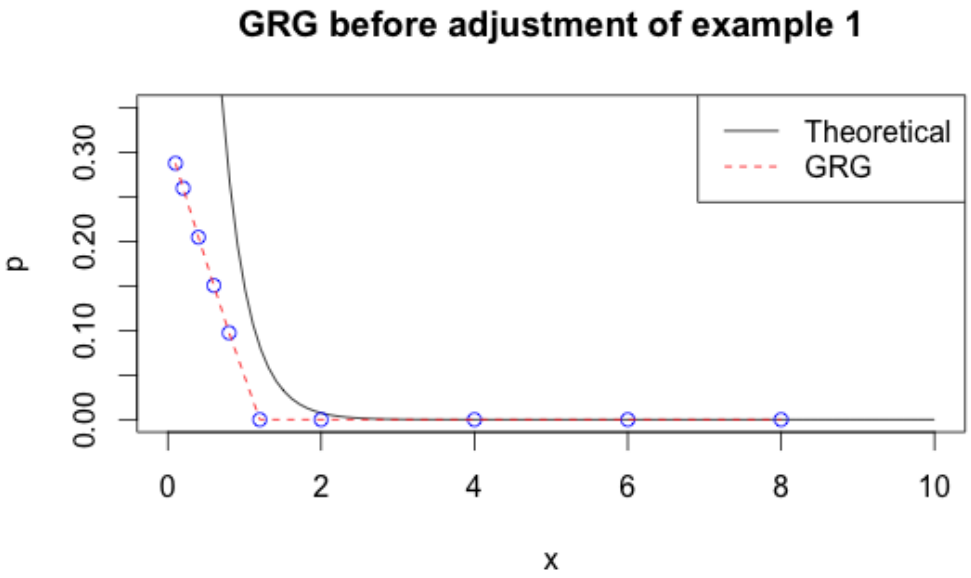

Figure 4.2: GRG before adjustment of example 1

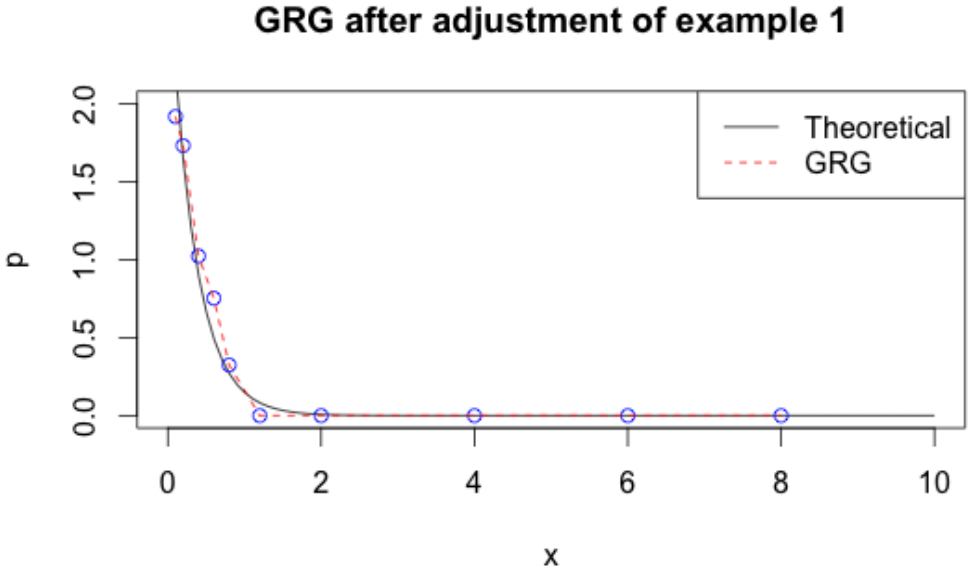

Figure 4.3: GRG Algorithm after adjustment of example 1

4.1.2 Example 2

Normally, the Laplace transform is a decreasing function from 1 to 0. As shown in the figure below, it is obvious that the more s approaches to positive infinity, the values of the $L(s)$ become closer to 0. So when considering the point values of s, we have an intuition that s should be close to 0.

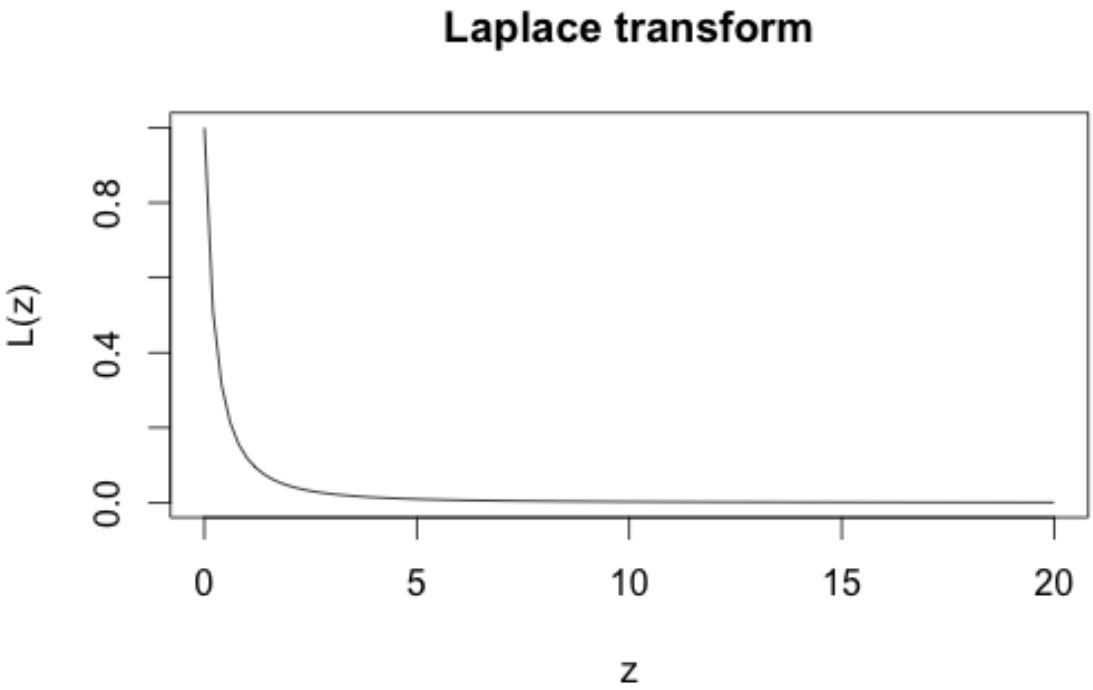

Figure 4.4: The graph of $L(s)$ in example 2

When considering the choice of values of x, we use properties of Laplace transform. Applying two approximations, we have:

$$L'(0)=\int_0^\infty (-x)e^{-sx}f(x)\,dx|_{s=0}=\int_0^\infty (-x)f(x)\,dx= \text{-}\mu$$

$$L'(0) = -\mu \approx \frac{L(0.01) - L(0)}{0.01 - 0} = -3.952023$$

$$\sigma^2 = E(X^2) - \mu^2$$

$$L''(0) = \int_0^\infty (x^2) e^{-sx} f(x)\, dx|_{s=0} = \int_0^\infty x^2 f(x)\, dx = E(X^2)$$

$$L''(0) \approx \frac{\Delta L(0.02) - \Delta L(0.01)}{0.02 - 0.01} = 24.69163$$

$$\text{where } \Delta L(0.01) = L(0.01) - L(0), \Delta L(0.02) = L(0.02) - L(0.01)$$

$$\sigma^2 \approx 9.073143$$

Take ten points of $x's$ ranging from $(\mu - 3\sigma\ ,\ \mu + 3\sigma)$

$= (-5.084474, 12.98852)$. Choose 10 points as following::

i	1	2	3	4	5	6	7	8	9	10
s	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
x_i	1	2	3	4	6	8	10	12	15	20

Table 4.2: Coefficient Matrix for example 2

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	0.9901	0.9802	0.9704	0.9608	0.9512	0.9418	0.9323	0.9231	0.9139
1	0.9802	0.9608	0.9418	0.9231	0.9048	0.8869	0.8694	0.8521	0.8353
1	0.9704	0.9418	0.9139	0.8869	0.8607	0.8353	0.8106	0.7867	0.7634
1	0.9608	0.9231	0.8869	0.8521	0.8187	0.7866	0.7558	0.7261	0.6977
1	0.9418	0.8869	0.8352	0.7866	0.7408	0.6977	0.6570	0.6188	0.5827
1	0.9231	0.8521	0.7866	0.7261	0.6703	0.6188	0.5712	0.5273	0.4868
1	0.9048	0.8187	0.7408	0.6703	0.6065	0.5488	0.4966	0.4493	0.4066
1	0.8869	0.7866	0.6977	0.6188	0.5488	0.4868	0.4317	0.3829	0.3396
1	0.8607	0.7408	0.6376	0.5488	0.4724	0.4066	0.3499	0.3012	0.2592
1	0.8187	0.6703	0.5488	0.4493	0.3679	0.3012	0.2466	0.2019	0.1653

We assume a set of initial points :

$p = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)$,

then we calculate:

$p = (0.146, 0.250, 0.204, 0.139, 0.0919, 0.0617, 0.049, 0.043, 0.014, 0)$

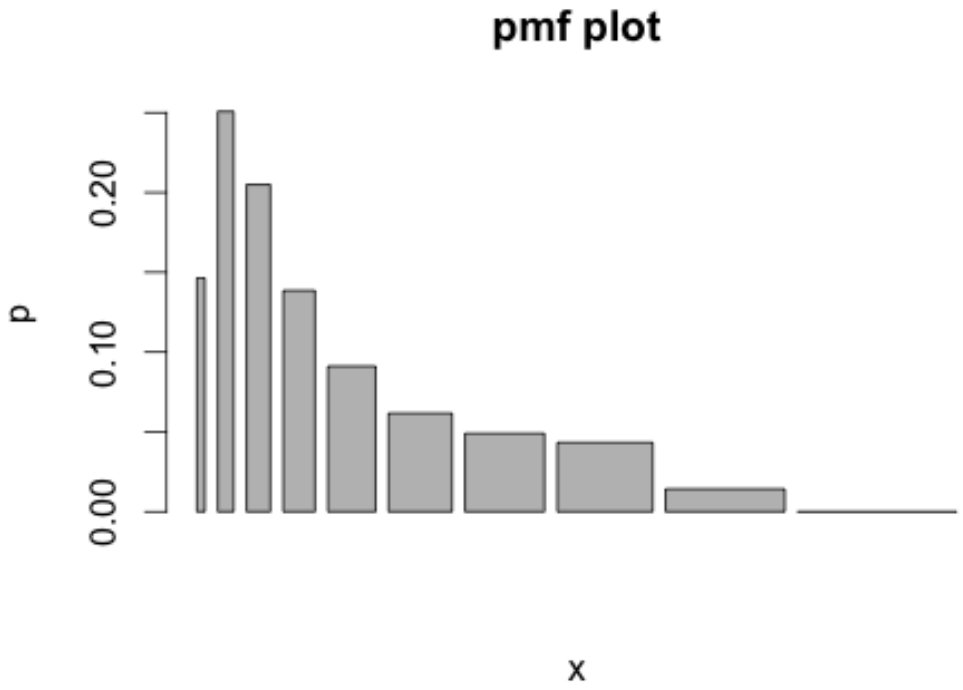

Figure 4.5: pmf plot of example 2

Because the values of x are not equally spaced, we must make some adjustment. After adjustment, we have:

$p* = (0.146, 0.251, 0.205, 0.139, 0.046, 0.031, 0.025, 0.022, 0.005, 0)$

We obtained a set of points to sketch the original function, which appears as the solid curve in the following graphs and compare it with theoretical function. Then we sketch values of p when x is equally spaced. From the graphs, we can see that after adjustment, it is closer to the original function. And we calculate the minimal differences $5.31E-09$.

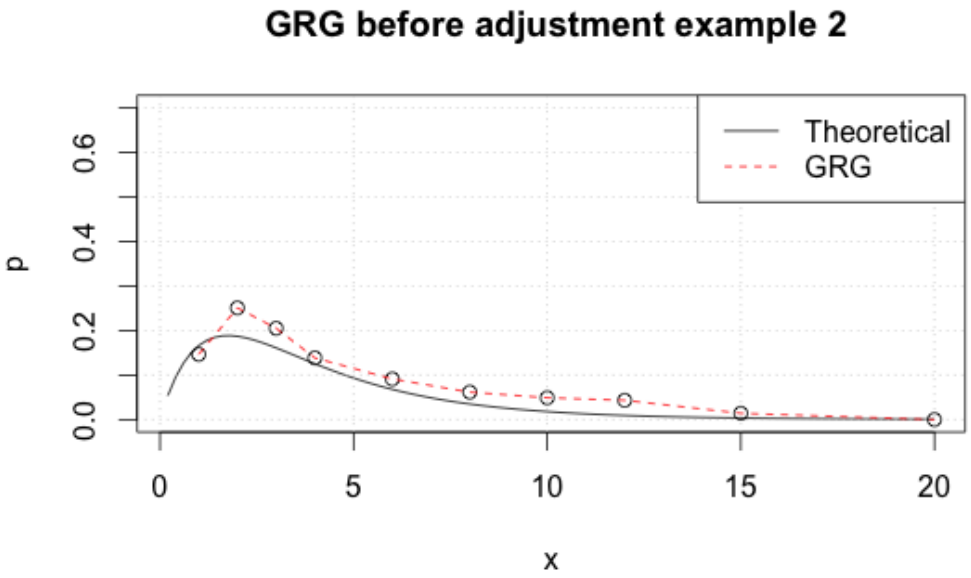

Figure 4.6: GRG before adjustment example 2

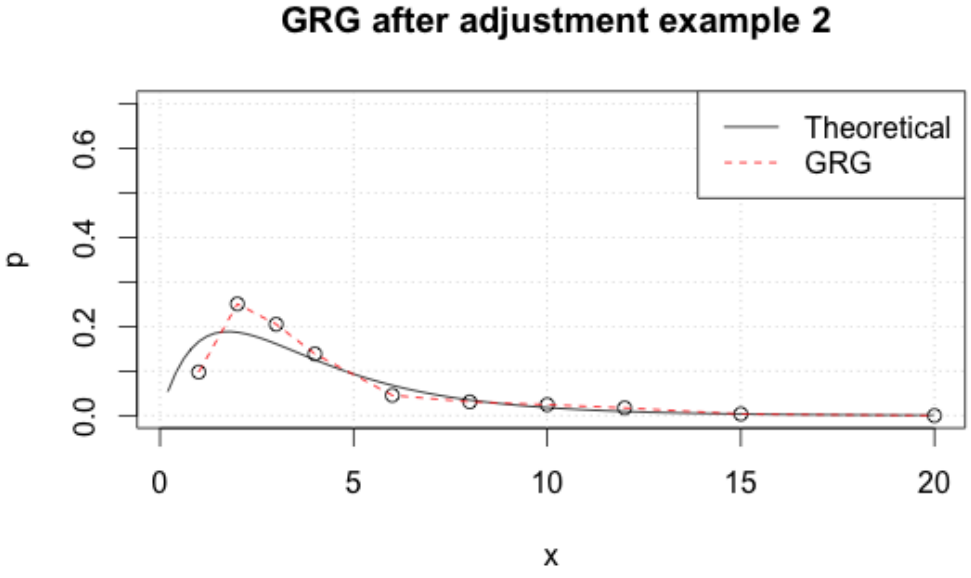

Figure 4.7: GRG after adjustment example 2

4.2 Best-Ritter Algorithm

Based on the Best-Ritter Algorithm, we have programmed a function named "$Inverselaplace$" in R. We input the Laplace transform function and choose the numbers (n) of points we want to have to estimate original function. Then, we command "$Inverselaplace(L, n)$" to get the values p and graphs in R.

4.2.1 Example 1

For $L(s) = \dfrac{3}{3+s}$,we input "$Inverselaplace(L, 20)$" in R, the values of p are:

$x = (0.8329, 0.0493, 0.0520, 0.0488, 0.0168, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$

0, 0, 0, 0).

Since we have adjustment in "$Inverselaplace$" command, the output is the final results. The graphs are as following:

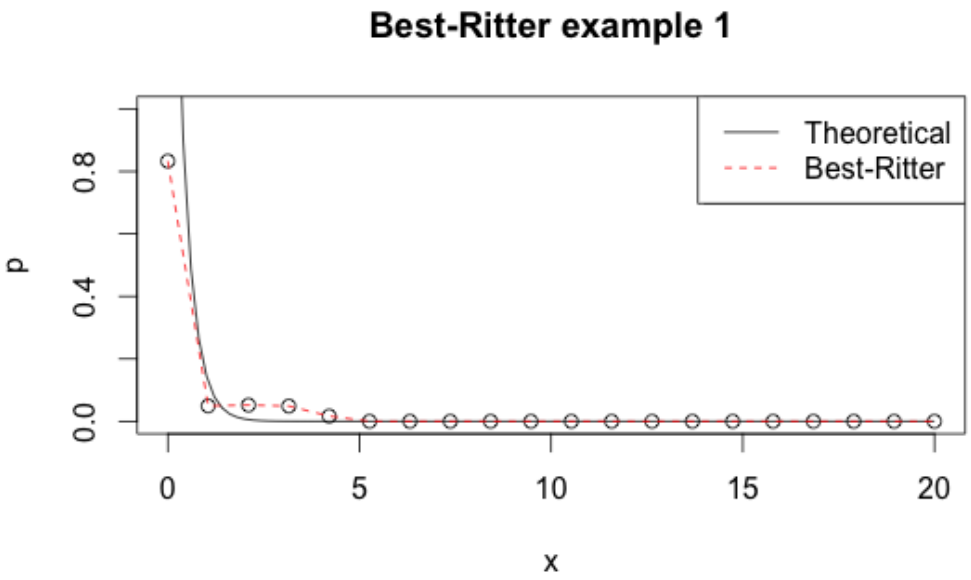

Figure 4.8: Best-Ritter on example 1

4.2.2 Example 2

Taking $L(s) = \frac{0.3 * 0.5}{0.3 + 0.5 + s} * (\frac{1}{0.3 + s} + \frac{1}{0.5 + s})$ as an example. We input "$Inverselaplace(L, 10)$", the answer is :

$x = (0.2393, 0.0996, 0.0996, 0.1001, 0.0511, 0.0496, 0.0497, 0.0163,$

0.0000, 0.0000,0).The graphs are as following :

From above plots, we can see that two algorithm works effectively on solving this problem.

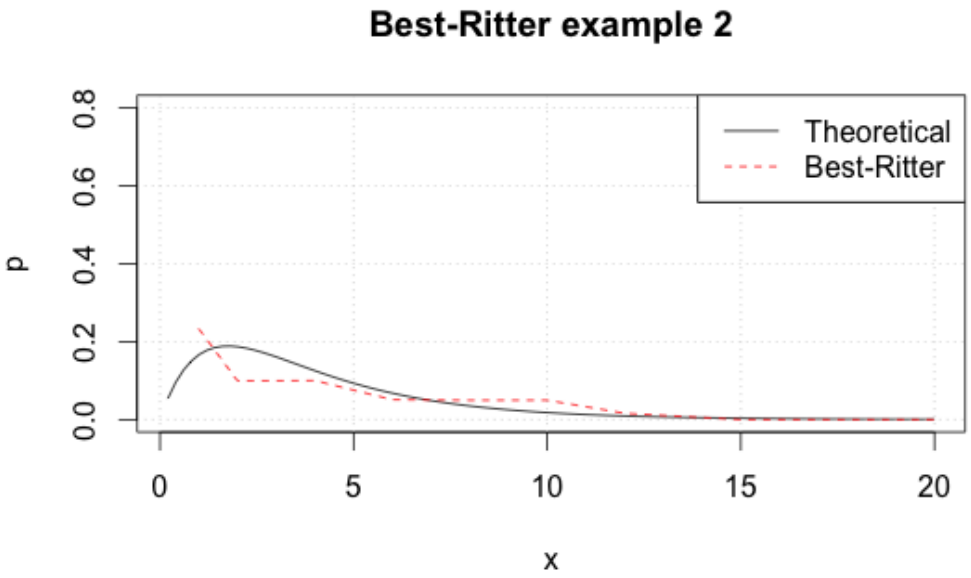

Figure 4.9: Best-Ritter on example 2

Bibliography

[1] Caron Richard, *Class notes from Non-linear Programming*, University of Windsor, (2015).

[2] Myron Hlynka, *class notes from Schocastic Process*, University of Windsor, (2015).

[3] Roy, Kirk Andrew, *Laplace transforms, probabilities and queues*, MSc thesis, Department of Math & Stat, University of Windsor, (1997).

[4] Grassmann, Winfried (Ed), *Computational Probability*, Springer,(2000).

[5] Venkataraman Panchapakesan, *Applied optimization with MATLAB programming*, John Wiley & Sons, (2009).

[6] Ridout Martin S, *Generating random numbers from a distribution specified by its Laplace transform*, Statistics and Computing, 2009, pp.(439–450).

Appendix A

R code

```
    ..............  Input  ..............
L  =..............
n  =..............

    ..............  Function  ..............
InverseLaplace=function(L,n)
{
     ..............  Set Initials  ..............
  z=seq(0,0.2,length=n)#could change
   slope0=(L(.01)−L(0))/.01 #estimates slope at 0 a
  slope1=(L(.02)−L(.01))/.01 # estimates slope at .01
  ex2= (slope1−slope0)/.01 #estimates 2nd deriv equals E(X^2)
  var=ex2−slope0^2 #estimate of variance
  sd=sqrt(var) #estimates stand devariance
  p=seq(0,slope0+3*sd,length=n)#could change
  u=−t(t(p))%*%z
  U=t(exp(u))
  H=2*t(U)%*%U
  c=−2*t(U)%*%L(z)
  ifelse(length(c)==ncol(H)&ncol(H)==nrow(H),
          assign('n',length(c)),stop('dimension is incorrect'))
  A=list()
  A=rbind(rep(1,n),−diag(n))
  B=t(t(c(1,rep(0,n))))
```

```
r=1
m=n
j=0
x=list()
x[[1]]=t(t(rep(1/n,n)))
D=diag(n)
D[1,]=rep(1,n)
C=list()
C[[1]]=solve(D,diag(n))
J=list()
J[[1]]=c(1,rep(0,n-1))
g=list()
g[[1]]=c+H%*%x[[1]]
s=list(0)
sigma=c()
sigmatilde=c()
sigmabar=c()
Q.fn=function(x)
{
  if(length(x)!=n)
    stop('length(x) incorrect')
  t(c)%*%x+t(x)%*%H%*%x/2
}
Q=list()
Q[[1]]=Q.fn(x[[1]])

 ..............  main body  ..............
step1.2=function(j,x,C,J,g)
{
  if(length(J[[j+1]][J[[j+1]]>=(r)&J[[j+1]]<=(r+m)])==0)
    step4(1,k,j,x,C,J,g)
  gc2=t(g[[j+1]])%*%C[[j+1]]
  gc2[which(J[[j+1]]==0|J[[j+1]]==-1)]=NaN
  k=min(which(gc2==max(gc2,na.rm=T)),na.rm=T)
  if(as.numeric(t(g[[j+1]])%*%C[[j+1]][,k])<=0)
    step4(2,k,j,x,C,J,g)
  s[[j+1]]=C[[j+1]][,k]
  step2(k,j,x,C,J,g,s)
}

step1.1=function(j,x,C,J,g)
{
  gc1=abs(t(g[[j+1]])%*%C[[j+1]]*(J[[j+1]]==0))
  k=min(which(gc1==max(gc1)))
```

```
  if(as.numeric(t(g[[j+1]])%*%C[[j+1]][,k])!=0)
  {
    s[[j+1]]=ifelse(rep(t(g[[j+1]])%*%C[[j+1]][,k]>0,n),
    C[[j+1]][,k],-C[[j+1]][,k])
    step2(k,j,x,C,J,g,s)
  }
  else step1.2(j,x,C,J,g)
}

step1=function(j,x,C,J,g)
{
  if(length(which(J[[j+1]]==0))==0)
    step1.2(j,x,C,J,g)
  else step1.1(j,x,C,J,g)
}

step2=function(k,j,x,C,J,g,s)
{
  if(as.numeric(t(s[[j+1]])%*%H%*%s[[j+1]])==0)
    sigmatilde[j+1]=Inf
  sigmatilde[j+1]=as.numeric((t(g[[j+1]])%*%
  s[[j+1]])/t(s[[j+1]])%*%H%*%s[[j+1]])
  if(sum(A[(r+1):(r+m),]%*%s[[j+1]]>=0)==m)
    sigmabar[j+1]=Inf
  cirno=(A%*%x[[j+1]]-B)/(A%*%s[[j+1]])*
  ((A%*%s[[j+1]]<0))
  sigmabar[j+1]=min(cirno[cirno>0],na.rm=T)
  l=min(which(cirno==sigmabar[j+1]))
  if(sigmabar[j+1]==Inf&sigmatilde[j+1]==Inf)
    stop('QP is unbounded from below')
  else
  {
    sigma[j+1]=min(sigmabar[j+1],sigmatilde[j+1])
    sigma[j+1]=round(sigma[j+1],10)
    step3(k,l,j,x,C,J,g,s,sigma,sigmabar,sigmatilde)
  }
}

step3=function(k,l,j,x,C,J,g,s,sigma,sigmabar,sigmatilde)
{
  x[[j+2]]=x[[j+1]]-sigma[[j+1]]*s[[j+1]]
  if(sigma[j+1]==sigmatilde[j+1]&sigmatilde[j+1]<=sigmabar[j+1])
  {
    J[[j+2]]=J[[j+1]]
```

```
  J[[j+2]][k]=-1
  C[[j+2]]=C[[j+1]]
  C[[j+2]][,k]=C[[j+1]][,k]/as.numeric(sqrt(t(C[[j+1]][,k])
  %*%H%*%C[[j+1]][,k]))
  b=which(J[[j+2]]!=-1)
  for(i in b)
  {
    C[[j+2]][,i]=C[[j+1]][,i]-C[[j+1]][,k]*
    as.numeric((t(C[[j+1]][,k])
    %*%H%*%C[[j+1]][,i])/(t(C[[j+1]][,k])%*%
    H%*%C[[j+1]][,k]))
  }
  C[[j+2]]=round(C[[j+2]],10)
  g[[j+2]]=c+H%*%x[[j+2]]
  print(paste(paste0('j=',j),
   paste0('x=(',paste0(round(x[[j+1]],2),collapse=','),')'),
    paste0('J=(',paste0(round(J[[j+1]],2),collapse=','),')'),
    paste0('g=(',paste0(round(g[[j+1]],2),collapse=','),')'),
    paste0('s=(',paste0(round(s[[j+1]],2),collapse=','),')'),
      paste0('sigma=',sigma[j+1]),sep='; '))
  j=j+1
  step1.1(j,x,C,J,g)
}
else
{
  a=which(J[[j+1]]==0)
  b=which(J[[j+1]]==-1)
  J[[j+2]]=J[[j+1]]
  J[[j+2]][k]=l
  J[[j+2]][b]=0
  C[[j+2]]=C[[j+1]]
  C[[j+2]][,k]=C[[j+1]][,k]/as.numeric(A[l,]%*%C[[j+1]][,k])
  for(i in (1:n)[-k])
  {
    C[[j+2]][,i]=C[[j+1]][,i]-C[[j+1]][,k]*
    as.numeric((A[l,]%*%C[[j+1]][,i])/(A[l,]%*%C[[j+1]][,k]))
  }
  C[[j+2]]=round(C[[j+2]],10)
  g[[j+2]]=c+H%*%x[[j+2]]
  print(paste(paste0('j=',j),
   paste0('x=(',paste0(round(x[[j+1]],2),collapse=','),')'),
   paste0('J=(',paste0(round(J[[j+1]],2),collapse=','),')'),
    paste0('g=(',paste0(round(g[[j+1]],2),collapse=','),')'),
    paste0('s=(',paste0(round(s[[j+1]],2),collapse=','),')'),
```

```
            paste0('sigma=',sigma[j+1]),sep='; '))
        j=j+1
        step1(j,x,C,J,g)
      }
    }


    step4=function(nn,k,j,x,C,J,g)
    {
      w=which(J[[j+1]]>0)
      u=-as.numeric(t(g[[j+1]])%*%C[[j+1]][,w])
      if(length(u[u<=0])==0)
        stop(sprintf('%s is a optimal solution %s',
          paste0('x=(',paste0(x[[j+1]],collapse=','),')'),nn))
    }

    while(j<=500)
      step1(j,x,C,J,g)
}


    ..... Failure Example in chapter 3.....
A=matrix(1:16,4,4)
s=1:4; x=c (.1,.3,.5,.7)
for(i in 1:4){  for (j in 1:4){ A[i,j]=exp(-s[i]*x[j]) }  }
L=3/(3+s)
L
p=solve(A)%*%L
sum(p)
plot(x,p)

A=matrix(1:100,10,10)
s=1:10; x=seq(.1,1.9,.2)
for(i in 1:10){  for (j in 1:10){ A[i,j]=exp(-s[i]*x[j]) }  }
L=3/(3+s)
L
p=solve(A)%*%L
sum(p)
plot(x,p)

    ..... example 1 in solver .....

p=c(0.287640317,0.259733475,0.204655291,
0.150544387,0.097384895,7.69399E-07,7.46514E-07,
```

```
7.0295E−07,3.26131E−06,5.59638E−06)
x=c  (.1,.2,.4,.6,.8,1.2,2,4,6,8)

  ..... pmf .....
barplot(p,x, main="pmf on example 1", xlab="x",ylab="p")
  ..... before aadjustment .....
lines(x,3*exp(−3*x), type="l")
c=seq(0,10,length=100)
plot(c,3*exp(−3*c),type='l',ltd=1,col=1,main=
"GRG before adjustment of example 1",
      xlab="x",ylab="p",ylim=c(0,0.35))
points(x,p,col='blue')
lines(x,p,lty=2,col=2)
legend('topright',c('Theoretical', 'GRG'),lty=1:2,col=1:2)

  ..... after adjustment .....
p=c(0.287640317,0.259733475,0.204655291,
0.150544387,0.097384895,7.69399E−07,7.46514E−07,
7.0295E−07,3.26131E−06,5.59638E−06)
x=c  (.1,.2,.4,.6,.8,1.2,2,4,6,8)
adjp=p/c  (.15,.15,.2,.2,.3,.6,1.4,2,2,4)
plot(c,3*exp(−3*c),type='l',ltd=1,col=1,main="GRG after
adjustment of example 1",xlab="x",ylab="p",ylim=c(0,2))
points(x,adjp,col='blue')
lines(x,adjp,lty=2,col=2)
legend('topright',c('Theoretical', 'GRG'),lty=1:2,col=1:2)

   ..... example 2 in solver .....
  ..... Excel pmf .....
x=c (1,2,3,4,6,8,10,12,15,20)
m<−c(0.146433679, 0.250772019,0.204922863,
0.138530899, 0.091139671,0.061652548,0.0490086,
      0.043311492, 0.014213687,1.12787E−06)
barplot(m,x, main="pmf plot",
         xlab="x",ylab="p")

  ..... excel example plot before adjustment .....
x=c (1,2,3,4,6,8,10,12,15,20)
m<−c(0.146433679, 0.250772019,0.204922863,
0.138530899, 0.091139671,0.061652548,0.0490086,
      0.043311492, 0.014213687,1.12787E−06)
F=function(s){0.3*0.5/(0.3+0.5+s)*(1/(0.3+s)+1/(0.5+s))}
L1 <− invlap(F, 0, 20, 100)
plot(L1 [[1]],  L1 [[2]],  type = "l",xlab="x",ylab="p",
```

```
      main="␣GRG␣before␣adjustment␣example␣2",ylim=c(0,.7))
grid()
points(x,m)
lines(x,m,col=2,lty=2)
legend('topright',c('Theoretical','GRG'),lty=1:2,col=1:2)

  ..... example plot after adjustment .....
x=c (1,2,3,4,6,8,10,12,15,20)
m1=m/c(1.5,1,1,1,2,2,2,2.5,4,10)
F=function(s){0.3*0.5/(0.3+0.5+s)*(1/(0.3+s)+1/(0.5+s))}
L1 <- invlap(F, 0, 20, 100)
plot(L1 [[1]], L1 [[2]], type = "l",xlab="x",ylab="p",
      main="␣GRG␣after␣adjustment␣example␣2",ylim=c(0,0.7))
grid()
points(x,m1)
lines(x,m1,col=2,lty=2)
legend('topright',c('Theoretical','GRG'),lty=1:2,col=1:2)

    ..... example 1 in Best−Ritter .....
    ..... set initials .....
z=c (0,.01,.02,.03,.04,.05,.06,.07,.08,.09)
p=c (1,2,3,4,6,8,10,12,15,20)
L=function(z){3/(3+z)}
n=10
InverseLaplace(L,10)
f=function(x){3*exp(−3*x)}
x=seq(0,20,length=100)
plot(x,f(x),type='l',main="Best−Ritter␣example␣1",
ylim=c(0,1),ylab="p")
p=seq(0,20,length=20)
InverseLaplace(L,20)
x1=c(0.8329725034,0.0493455166,0.0520754288,
  0.0488347012,0.01677185,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
points(p,x1)
lines(p,x1,col=2,lty=2)
legend('topright',c('Theoretical','Best−Ritter'),lty=1:2,col=1:2)

    ..... example 2 in Best−Ritter .....
    ..... set initials .....
z=c (0,.01,.02,.03,.04,.05,.06,.07,.08,.09)
 p=c (1,2,3,4,6,8,10,12,15,20)
L=function(z){(0.3*0.5/(0.3+0.5+z))*(1/(0.3+z)+1/(0.5+z))}
n=10
x=c(0.35,0.0995903591,0.0996450011,0.1001132749,
```

```
0.1022192148,0.0992374972,0.099391751,0.0408140589,0,0)
   ..............  laplace plot  ..............
z=seq(0,20,length=100)
plot(z,L(z),type="l",main="Laplace transform")

L1=invlap(L, 0, 20, 100)
plot(L1 [[1]], L1 [[2]], type = "l", col ="1",xlab="x",ylab="p",
      main="Best-Ritter example 2", ylim=c(0,0.8))
x1=x/c (1.5,1,1,1,2,2,2,2.5,4,10)
grid()
#points(p,x1)
lines(p,x1,type='l', lty=2,col=2)
legend('topright',c('Theoretical', 'Best-Ritter'),lty=1:2,col=1:2)
```

VITA AUCTORIS

Zishan Huang was born in 1991 in Kunming, Yunnan province, China. After graduating from high school in 2010, she wen to the Hunan University where he obtained a B.sc in Mathematics and Applied Mathematics in 2014. She is currently a candidate for the Master's degree in Statistics at the University of Windsor and hopes to graduate in spring 2016.