Iterated Grid Search Algorithm on Unimodal Criteria

by

Jinhyo Kim

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Statistics

APPROVED:

Co-advisor: Dr. Richard G. Krutchkoff

Co-advisor: Dr. George R. Terrell

Dr. Jesse C. Arnold

Dr. Clint W. Coakley

Dr. Robert V. Foutz

June, 1997

Blacksburg, Virginia

Iterated Grid Search Algorithm on Unimodal Criteria

Jinhyo Kim

(ABSTRACT)

The unimodality of a function seems a simple concept. But in the Euclidean space $R^m, m = 3, 4, \ldots$, it is not easy to define. We have an easy tool to find the minimum point of a unimodal function.

The goal of this project is to formalize and support distinctive strategies that typically guarantee convergence. Support is given both by analytic arguments and simulation study. Application is envisioned in low-dimensional but non-trivial problems. The convergence of the proposed iterated grid search algorithm is presented along with the results of particular application studies.

It has been recognized that the derivative methods, such as the Newton-type method, are not entirely satisfactory, so a variety of other tools are being considered as alternatives. Many other tools have been rejected because of apparent manipulative difficulties. But in our current research, we focus on the simple algorithm and the *guaranteed* convergence for unimodal function to avoid the possible chaotic behavior of the function. Furthermore, in case the loss function to be optimized is not unimodal, we suggest a weaker condition: almost (noisy) unimodality, under which the iterated grid search finds an *estimated* optimum point.

Subject Classification: statistical computing, nonlinear estimation, statistical optimization, statistical simulation

Key Words: Iterated Grid Search, grid, dichotomous search, unimodality, quasi-convexity, envelope, condition number, derivative-free

ACKNOWLEDGEMENTS

I would like to thank my family and friends, especially Euikyu Kim and Chawhan Chung, for their support, patience and encouragement.

I would especially like to thank Dr. Richard G. Krutchkoff whose coaching and encouragement have been there for me always and also like to thank Dr. George R. Terrell who gave me mathematical help.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 0

Introduction

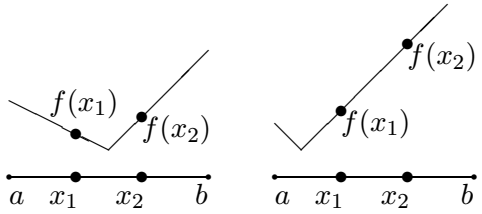

Figure 0.1: Test Points

Consider finding the minimum point of a unimodal function. Select two test points, x_1 and x_2, in each iteration step. If $f(x_1) < f(x_2)$ as shown in Figure 0.1, then the optimum point *must* exist in $[a, x_2]$; however, if $f(x_1) > f(x_2)$, then the optimum point *must* exist in $[x_1, b]$. Thus depending on the values of $f(x_1)$ and $f(x_2)$, the new interval is reduced in each iteration. Repeating this in the new range provides the Iterative Grid Search algorithm for finding the optimum point, within any $\epsilon > 0$.

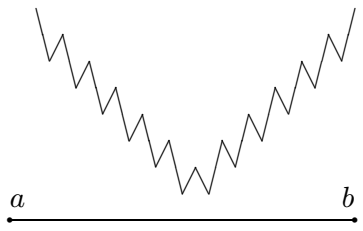

Figure 0.2: almost unimodality

We begin with some examples to show why this iterated grid search algorithm is needed. 1)In the case of regression with a bad condition number, the usual procedures (QR decomposition in *Splus* and *SAS*) *cannot* find the solution, but the iterated grid search can. 2)the maximum likelihood estimation (MLE) problem in the Cauchy distribution cannot be solved by the usual means, such as the Newton-type methods; however, the iterated grid search has *guaranteed* convergence. 3)Other problems, such as the Best Order Statistics, use criteria that are not unimodal but are almost unimodal, as shown in Figure 0.2. Here derivative methods cannot be used at all. The *derivative-free* iterated grid search procedure does, however, obtain a reasonable solution.

Chapter 2 has a full description of the iterated grid search algorithm along with some modifications: we start with a one-dimensional iterated grid search; with a special golden number ratio, we

then show how to use the previous internal test point to save computations; this is then extended to the m-dimensional case. Furthermore, due to the *strong* condition of unimodality, significant improvement in speed is presented; its overall speed is proportional to $m + 1$, which is a lot less than 2^m in the case of iterated grid search. Other modifications, using different number of test points, in each iteration step, are proposed and compared. We show that the iterated grid search algorithm has *guaranteed* convergence for a unimodal function; furthermore it will find a reasonable solution for an almost unimodal function.

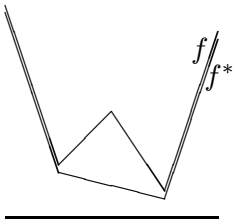

Figure 0.3: Envelope

Since the iterated grid search algorithm works so well for unimodal criteria, we spend chapter 3 discussing unimodality. We show methods for finding a unimodal function based on a function which itself is not unimodal. The convex envelope, as shown in Figure 0.3 for example, is one of these methods. The minimum of the original function f is the same as the convex envelope f^*.

Chapter 4 gives the details of the simulation studies for the applications and examples. Using the codes provided, we find the optimum point for the examples presented in chapter 1.

Chapter 1

Examples

In this chapter, we will present some examples of situations where the usual procedures do not provide answers to simple problems. These will serve as a motivation to the procedure developed in this dissertation. Our new procedure, the Iterated Grid Search (IGS) method, to be described in chapter 2, will provide a solution to these problems. [1]

1.1 Multiple Linear Regression with a bad condition number

In this section, we begin with a fairly simple problem to show that the usual procedures do not work while our new procedure will.

Consider the Multiple Linear Regression (MLR) with standard assumptions

$$y = b_0 + b_1 x_1 + \ldots + b_p x_p + \epsilon \tag{1.1}$$

$$\underline{\hat{y}} = X\underline{\hat{b}} \tag{1.2}$$

where $x_1, \ldots, x_p$ represent the independent variables and $b_0, \ldots, b_p$ are unknown parameters. Using the Squared Error Loss (SEL) criterion, the object function to be minimized is $f(\underline{b}) = (\underline{y} - X\underline{b})^{'}(\underline{y} - X\underline{b})$. By the Gauss-Markov theorem [35], the Least Square Estimate (LSE) is $\underline{\hat{b}} = (X^{'}X)^{-1}X^{'}\underline{y}$,where $\underline{y}$ is the vector of observations.

If the *condition number*[2] of $X^{'}X$ is bad(i.e. too large), what procedure is recommended for finding the inverse of $X^{'}X$? A condition that can cause an erratic behavior in the Newton-

[1] This dissertation was typeset using LaTeX. [23]

[2] **Condition number** [39] : It is used in cases of the ill-conditioned linear regression. This number is proposed for the measure of the difficulty for finding the inverse of $X^{'}X$. When we say that the condition for finding the inverse

type methods is the singularity of the design matrix X caused by the collinearity of the columns. Previous research shows that the QR decomposition[3] is recommended and is commonly used.[4] The usual inverse of $(X^{'}X)^{-1}$ is unstable if the condition number is bad, and the example proposed in this section will still show that the QR decomposition is sometimes not reliable for evaluating $(X^{'}X)^{-1}$. Many statistical packages use the Marquart method, [5] but it is known that this method does not always provide a stable solution.

---

of $X^{'}X$ is bad, we typically mean that the smallest singular value of $X^{'}X$ is too small relative to the others. Such an interpretation naturally leads to quantities such as the condition number, which for a square matrix, is defined as the ratio of the smallest to the largest nonzero singular value. Let $\lambda_{min} \leq \ldots \leq \lambda_{max}$ represent the eigenvalues of $X^{'}X$. The condition number $\kappa(X^{'}X)$ of the square matrix $X^{'}X$ is defined by

$$\kappa(X^{'}X) = \frac{\lambda_{max}}{\lambda_{min}} \tag{1.3}$$

[3]**QR decomposition** [39] : For MLR in formula 1.2, consider a decomposition $X = QR$ for a design matrix X, not necessarily square, where Q is an orthogonal matrix and R is an upper triangular matrix. We have $X^{'}X = R^{'}R$ since $Q^{'}Q = I$.

Then the Normal equation

$$X^{'}X\underline{b} = X^{'}\underline{y} \tag{1.4}$$

is equivalent to

$$R^{'}R\underline{b} = R^{'}Q^{'}\underline{y} \tag{1.5}$$

since $(QR)^{'}(QR)\underline{b} = (QR)^{'}\underline{y}$ in which $R^{'}R$ has a *better* condition number than $X^{'}X$. However note that there is still a possibility of numerical error in the decomposition $X = QR$.

[4]Refer to the built-in function `lsfit` in *Splus 3.1* [34].

[5]**Levenburg(1944) and Marquardt(1963) methods** [19] : Consider a least square estimate $\hat{b}$ as a function λ

$$\hat{b}_1(\lambda) = (X^{'}X + \lambda I)^{-1}X^{'}\underline{y} \tag{1.6}$$

$$\hat{b}_2(\lambda) = (X^{'}X + \lambda D)^{-1}X^{'}\underline{y} \tag{1.7}$$

for some good choice of $\lambda \in R^1$ so that $(X^{'}X + \lambda I)$ and $(X^{'}X + \lambda D)$ are not ill-conditioned (better-conditioned) where D is a diagonal matrix with entries equal to the diagonal elements of $X^{'}X$.

Note that the Marquart method is a compromise between the Gauss-Newton method and the steepest descent method. (Marquart 1963) As $\lambda \longrightarrow 0$, the estimate approaches the Gauss-Newton method; as $\lambda \longrightarrow \infty$, the estimate approaches the steepest descent method. This method may still cause a difficulty in finding a matrix inverse even with a good choice of λ.

Example 1 *Suppose we have a multiple linear regression model* $\underline{\hat{y}} = X\underline{\hat{b}}$ *with*

$$X = \begin{bmatrix} 1 & 2 \\ 1+\epsilon & 2 \\ 1+2\epsilon & 2 \end{bmatrix}, \ \underline{y} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \tag{1.8}$$

where ϵ *is a perturbation factor. Consider the design matrix* X [6] *with* $\epsilon = 10^{-\alpha}$ *and* $\alpha = 1, 2, 3, \ldots$.[7] *For* $\alpha \geq 4$*, due to the finite length of the mantissa, the 486 IBM PC Compatible machine does not provide the required inverse using the commonly used procedures, as shown in appendices 5.3.*

Solution1 QR decomposition

Consider $\epsilon =$ `eps` provided in the code in appendices 5.3. The QR decomposition in *Splus 3.1*, [6] [33] [34] as shown in the source code in appendices 5.8, gives the result in appendices 5.3. Note that, with $\alpha = 4$ ($\epsilon = 10^{-4}$), the usual procedure `lsfit` using the QR decomposition produces the following error message:

`apparently singular matrix`. [8]

This means that the package *Splus 3.1* understands that the design matrix is singular and cannot continue computation.

Solution2 SAS procedure

Given the MLR model with in 1.8, we use the *SAS* [32] procedure `proc reg` to find the least square

---

[6] Other examples like this can be easily constructed, e.g.

$$X_1 = \begin{bmatrix} 1+\epsilon & -1 \\ -1 & 1+\epsilon \\ 2 & -2 \end{bmatrix}, \ X_2 = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix} \tag{1.9}$$

The only requirement is the singularity of the column rank, which is the same as the row rank.

[7] We note that this $\alpha = 1, 2, 3, \ldots$ is intentionally selected so that X matrix is almost singular (producing a bad condition number).

[8] it means that the QR decomposition in 32-bit digital computers considers the X matrix as a singular matrix. The condition number of $X^{'}X$ with $\alpha = 4$ is 1.06×10^{-9} as shown in appendices 5.3.

estimate of $\underline{b}$. See the output in appendices 5.4 using `proc reg` in *SAS*. The message in the output is the following:

```
Model is not full rank.
Some statistics will be misleading
```

Solution3 Exact calculation

Note that the exact hand calculation gives the correct solution, which is the same answer as the solution using the iterated grid search algorithm, to be discussed later. With $\alpha = 4$ and $\epsilon = 10^{-4}$,

$$X = \begin{bmatrix} 1 & 2 \\ 1+\epsilon & 2 \\ 1+2\epsilon & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \\ 1.0002 & 2 \end{bmatrix} \quad , \quad \underline{y} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \tag{1.10}$$

$$\begin{aligned} \hat{b} &= (X'X)^{-1}X'\underline{y} \\ &= \begin{bmatrix} 10^4 \\ -4999 \end{bmatrix} \end{aligned} \tag{1.11}$$

Solution4 IGS algorithm

We show that the output using the main code IGS includes the required answer in 1.11 as follows:

```
This is output of ft2
 xmin=  0 -100000
 xmax=  100000 0
 grid search   10000.0000000133 -4999.00000000663
```

We see that the Iterated Grid Search method, which we shall discuss in chapter 2, produces a correct solution. Note that the function $f : R^2 \longrightarrow R^1$ to be minimized by the iterated grid search procedure was $f(\underline{b}) = (\underline{y} - x\underline{b})'(\underline{y} - x\underline{b})$, $\underline{b} \in R^2$.

Therefore, we conclude that, for this ill-conditioned problem, $(X^{'}X)^{-1}$ is not reliably computed using the usual procedures implemented in the *Splus 3.1* or the *SAS* packages. However the iterated grid search algorithm does provide a reasonable solution.

1.2 MLE of the Cauchy distribution

In this section,[9] we present an example where the Newton-type methods cannot be used, but the proposed iterated grid search method can.[10]

Newton methods[11] are often used to solve problems in maximum likelihood estimation. This example will illustrate a situation in which the Newton methods cannot be used to solve a maximum likelihood problem due to its divergence behavior, but the iterated grid search method will provide the solution.

Consider the problem of finding the Maximum Likelihood Estimator(MLE) of the location parameter θ in the Cauchy distribution with location parameter θ. The likelihood function can be written as

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{1+(x_i-\theta)^2} \tag{1.14}$$

where $x_i \in R^1$, $\theta \in R^1$. The log-likelihood $l(\theta) = \log L(\theta) = -\sum_{i=1}^{n} \log[1+(x_i-\theta)^2]$ has the same maximum point[12] as the likelihood function $L(\theta)$. Taking a derivative of $l(\theta)$, we obtain

$$\frac{\partial l}{\partial \theta} = 2\sum_{i=1}^{n} \frac{(x_i-\theta)}{1+(x_i-\theta)^2} \tag{1.15}$$

---

[9] The example in this section was partially quoted from an exercise in [37] on page 140.

[10] Of course, other kinds of methods like the Secant method may be used.

[11] **Newton method** : Consider the problem of finding a zero of the equation $f(x) = 0$. Expanding f in a Taylor series at x_i gives

$$f(x) = f(x_i) + (x-x_i)f^{'}(x_i) + \dots \tag{1.12}$$

The iteration scheme is

$$x_{i+1} = x_i - \frac{f(x_i)}{f^{'}(x_i)} \tag{1.13}$$

A plausible starting value is the median or a trimmed mean of the observations.

[12] We can maximize either $L(\theta)$ or $l(\theta)$ since $\log(x)$ is a monotone function.

To find the MLE, we set it to zero to get the equation

$$g(\theta) \equiv \sum_{i=1}^{n} \frac{(x_i - \theta)}{1 + (x_i - \theta)^2} = 0 \tag{1.16}$$

Unfortunately, there does not exist a closed-form solution to this equation, and it has been shown that the Newton method of finding the solution of $g(\theta) = 0$ diverges. [37] The Newton method is known to exhibit chaotic behavior in this problem.[13] This problem can be solved using the Secant method. However for guaranteed convergence, the iterated grid search[14] method is recommended since the graphs are *unimodal* as shown in Figure 4.1. [15]

We note that 1)the Newton method uses the exact slope using the derivative of one point at each iteration, 2)the Secant method[16] uses the approximated slope using two points of $(x_1, f(x_1))$ and $(x_2, f(x_2))$, while 3)IGS will use the shape of the whole function via four points[17] in each iteration. Also we note that the result of the Newton method depends on the initial point; and the result of the Secant method depends on the initial *two* points. However, if we have any unimodal function, as in this example of the Cauchy distribution, then the iterated grid search method will work with guaranteed convergence, as will be *mathematically* shown in theorem 1 in chapter 2.

1.3 An example in which IGS must be used

In this example, we introduce a new criterion, Best Order Statistic (BOS), to present an example in which the iterated grid search method *should* be used. In this case, none of the other methods,

---

[13] See Reeds(1985) [28] for a discussion of the Cauchy likelihood.

[14] It is easy to make other examples in which the Newton method does not converge but the iterated grid search converges.

[15] In chapter 2, it will be *mathematically* shown that the unimodality guarantees the convergence of the iterated grid search.

[16] Ralston and Jenrich (1978) introduced the routine, DUD(Doesn't Use Derivative), which is based on using a secant plane approximation to the expectation surface rather than a tangent plane approximation. [3]

Secant method : Given two values x_i and x_{i-1}, the secant of $f(x)$ is the line intersecting the curve $f(x)$ at $f(x_i)$ and $f(x_{i-1})$. The iteration is

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \tag{1.17}$$

Note that the derivatives are estimated from the previous iterations rather than being supplied analytically.

[17] including two test points in the interval of current iteration step

e.g. Newton-type or Secant methods, can be used to find the optimum point.

In Figure 4.3, we see that the object function of BOS criterion to be optimized is almost unimodal.[18] Obviously, derivative-type methods cannot be applied to this kind of object function. [19] We are not interested in identifying[20] the exact structure of the object function in BOS but only in finding the estimated optimum point. To this end, we can use the iterated grid search method for problems, e.g. BOS, that display an almost unimodal function, [21] which will be defined in chapter 3.

The BOS estimator $\hat{\theta}$ using the BOS criterion can now be defined as the estimate of θ which minimizes

$$g(\theta) = \sum_{i=1}^{n} (FX_{[i]} - \frac{i}{n+1})^2 \tag{1.18}$$

where $FX_{[i]}$ is the ith order statistics of the empirical distribution. Since $g(\theta)$ displays almost unimodality, as seen in Figure 4.3, we can use the iterated grid search procedure to find the approximate estimate. A more detailed description of the BOS estimator is given in section 3.5.

1.4 An example of unimodal criteria

In this section, we shall introduce a simple example of a situation with a new kind of criterion. As we shall see later, the iterated grid search method can be used with any unimodal (or almost unimodal) criteria function.

Consider the simple linear regression. In this section, we do not employ the widely-used criterion SSE, but we want to find a new estimate $\hat{b}$ such that the object function $g(b)$ in formula 1.19 using

[18] At this point, the terminology of almost unimodality is not defined. Other terminology, e.g. noisy unimodal, can be used. More discussion is given in chapter 3.

[19] Of course, smoothing function technique for the crude BOS function can be proposed to use the derivative-type methods. But this is another area and left to further study.

[20] Actually in this dissertation it is not the main focus to identify the reason why this example of BOS shows almost unimodality.

[21] Note that no other methods can be recommended and we want to get an *approximate* estimate. At this point the precision of the estimate is not important.

a new criterion has a minimum value. We suggest the object function [22] to be minimized:

$$g(b) = \sum_{i=1}^{n} \exp(-\exp(-\frac{1}{2}(y_i - bx_i)^2)) \tag{1.19}$$

The graph of this function is shown in Figure 4.4. Therefore we can apply the iterated grid search[23] method to this example due to its unimodality.

1.5 Nonlinear optimization review implemented on SAS

In this section, [24] we summarize the procedure `proc nlin` in the commonly-used statistical package *SAS* for the review of the nonlinear optimization methods implemented in *SAS*.

The procedure `proc nlin` fits nonlinear regression models using 4 selected methods. This procedure first examines the starting value specifications of the parameters. If a grid of values is specified, the procedure `proc nlin` evaluates the residual sum of squares, at each combination of values, to determine the best set of values to start the iterative algorithm. The procedure currently uses one of the following four methods: [25]

- modified Gauss-Newton method
- Marquardt method
- gradient or steepest-descent method
- multivariate secant of false position

[22] More will be discussed on how this criterion $f(b)$ can be derived.

[23] Probably other methods works fine in this nonlinear programming. It is noted that this example is just to show an *unimodal* criterion.

[24] This section is partially quoted from [32].

[25] SAS also specifies the relative convergence criterion. The iterations are said to have converged if

$$\frac{SSE_{i-1} - SSE_i}{SSE_i + 10^{-1}} < \epsilon \tag{1.20}$$

This convergence criterion is called *relative distance*. The default of the ϵ value is 10^{-8}. And it has a `bound` statement for specifying the initial interest region.

Consider a general nonlinear model, $Y = F(\underline{\beta}, \underline{X}) + \epsilon = F(\underline{\beta}) + \epsilon$. The nonlinear "normal" equations for the nonlinear optimization are $X^{'}F(\underline{\beta}) = X^{'}\underline{e}$ where $X = \partial F/\partial\underline{\beta}$.

In many nonlinear situations, both X and $F(\beta)$ are functions of β, and a closed-form solution does *not* exist. Thus, the procedure `proc nlin` uses an iterative process: a starting value for β is chosen and continually improved until the error sum of square (SSE) $\hat{\epsilon}^{'}\hat{\epsilon}$ is minimized. The iterative process begins at some point β_0 specified by the end-user. Then X and $\hat{y}$ are used to compute the value $\triangle$ such that, for $k \in R^1$ and $\triangle \in R^p = dim(\beta)$,

$$SSE(\beta_0 + k\triangle) < SSE(\beta_0) \tag{1.21}$$

We note that $\triangle$ represents the search direction and k represents the search step size. The methods differ in how $\triangle$ is computed to change the vector of parameters. The four methods described have different schemes to find the direction and distance:

- Steepest Descent : $\triangle = X^{'}e$ (direction)
- Gauss-Newton : $\triangle = (X^{'}X)^{-1}X^{'}e$ (direction and distance)
- Marquardt : $\triangle = (X^{'}X + \lambda I)^{-1}X^{'}e$ (direction and distance) for some appropriate $\lambda \in R^1$
- Secant : $\partial y/\partial b_{i+1} \equiv (y_i - y_{i-1})/(b_i - b_{i-1})$

Note that *SAS* does not employ the derivative-free methods except for the Secant method. Therefore, *SAS* cannot find the solution 1)for the example presented in section 1.1 as the numerical output using *SAS* is shown in appendices 5.4 ; 2)for the example in section 1.3 obviously. These facts support the use of the iterated grid search algorithm.

1.6 Multiple Linear Regression (general case)

In this section, we revisit the most popular case of the multiple linear regression. This shows that the iterated grid search can be used to find the solution of the normal equation[26] due to its

[26]Of course, other derivative methods work fine as well.

exact convexity structure. However, as will be discussed in chapter 2, the iterated grid search is not optimal here due to its relatively slow speed.

Consider the Gauss Markov Model $\underline{y} = X\underline{b} + \underline{\epsilon}$ with standard assumptions where X is an $n \times p$ design matrix and $E\underline{\epsilon} = 0$, $Var\underline{\epsilon} = \sigma^2 I$. Minimize SSE $f(\underline{b})$ as a function of $\underline{b}$ given a design matrix X and an observation vector $\underline{y}$.

$$f(\underline{b}) = (\underline{y} - X\underline{b})'(\underline{y} - X\underline{b}) = \underline{y}'\underline{y} - \underline{y}'X\underline{b} - \underline{b}'X'\underline{y} + \underline{b}'X'X\underline{b} \tag{1.22}$$

Note the vector differentiation gives

$$\frac{\partial}{\partial \underline{b}}\underline{b}'x = \frac{\partial}{\partial \underline{b}}\underline{x}'b = \underline{x} \tag{1.23}$$

$$\frac{\partial}{\partial \underline{b}}\underline{b}'A\underline{b} = 2A\underline{b} \tag{1.24}$$

Therefore

$$\frac{\partial f}{\partial \underline{b}} = -2X'\underline{y} + 2X'X\underline{b} \tag{1.25}$$

and

$$\frac{\partial^2 f}{\partial \underline{b}^2} = 2X'X \tag{1.26}$$

which is a positive definite matrix for $X \neq 0_{n\times p}$ almost everywhere. Therefore the object function f of SSE is convex with respect to $\underline{b}$.

Thus, in this MLR situation, the criterion in formula 1.22 does provide a unimodal function. Therefore the iterated grid search method can solve this problem, but so can many other procedures. The only advantage here is that a computer program can be written to solve a wide range of problems, including this and others, that the other procedures cannot solve.

Chapter 2

The Iterated Grid Search Method

In this chapter, we formulize and discuss the Iterated Grid Search(IGS) algorithm, giving the definition of the algorithm along with its improvements and modifications. We also present the principle used to analyze the algorithm, and we give the analysis of the algorithm with comparisons to other well-known algorithms. In sections 2.1 and 2.2, we describe the grid search algorithm for a general function before considering the unimodality of a function in section 2.3. The important features of this chapter are described in sections 2.3, 2.4 and 2.5.

2.1 A Grid Search

In this section[1], we begin with a n-point grid search[2] in one and m-dimensional parameter space. This section includes the preliminary review for the Iterated Grid Search procedure that will be described in section 2.5.

The 1-dimensional grid search

In the 1-dimensional parameter case, consider the optimization of a general function $f : R^1 \longrightarrow R^1$. If we have n test points $x_1, \ldots, x_n$ such that $L \leq x_1 \leq \ldots \leq x_n \leq U$, then the set of values, $f(x_1), \ldots, f(x_n)$, gives some indication of the behavior of f. In many cases, n, the number of test points in each iteration step, will be crucial in the algorithm because, in many cases, the speed of the algorithm depends on n. As we shall see later, two test points, $f(x_1)$ and $f(x_2)$, are sufficient

[1] In this section, see [37] for more detail.

[2] In this dissertation, the terminology *grid search* is referred from [37], but the *iterated grid search* is new terminology.

to give an indication of the location of the optimum point when f has the special condition of *unimodality*.

To adopt a grid search method based on the grid defined above, we need to specify 1)how many points we need to check in each iteration step (What is n?) and 2)how we pick the n test points $x_1, \ldots, x_n \in [L, U]$.

Consider the equi-distance[3] grid such that

$$\mathcal{L} = \{x_k \,|\, x_k = L + \frac{k}{n+1}(U - L),\ k = 1, \ldots, n\} \tag{2.1}$$

in a situation where we are given the initial region as $\mathcal{I} = [L, U]$. Using the grid $\mathcal{L}$ with *card*($\mathcal{L}$)=n, the values of $f(x_1), \ldots, f(x_n)$ will be used in the grid search algorithm. In this chapter, the number n and the rule used to pick the grid $\mathcal{L}$ will be described with respect to an appropriate assumption on the object function.

The m-dimensional grid search

In the m-dimensional parameter case, when we are given the initial region[4] $\mathcal{I} = [a_1, a_2] \times [b_1, b_2] \times \ldots \in R^m$, we can consider the equi-distance grid

$$\begin{aligned}\mathcal{L} = \{(x_1, \ldots, x_m)^T \mid\ & x_1 = a_1 + \frac{k_a}{n+1}(a_2 - a_1)\ ,\ k_a = 1, \ldots, n\ , \\ & x_2 = b_1 + \frac{k_b}{n+1}(b_2 - b_1)\ ,\ k_b = 1, \ldots, n,\ \ldots\}\end{aligned} \tag{2.2}$$

We can choose a specific grid, as in the 1-dimensional parameter case. Since *card*($\mathcal{L}$)=n^m grows exponentially in m, the full grid search (when n is large) is impractical in higher dimensional problems. Even for a small m, most of the function evaluations will be wasted. [5]

In the next section, we calculate the optimal n in terms of the expense (computing time).

---

[3]Note that the non-equidistance grid algorithm may *possibly* provide better efficiency than equi-dist grid algorithm. This will be discussed in the next sections.

[4]Different kinds of initial regions other than rectangular, e.g. triangular or circular, can be suggested. But it is found that those produce some complexity in the region of the next iteration and do not provide a better reduction ratio, which will be defined in next sections. In other words, the shape of the interest region in the second iteration step is *not* triangular any more if we start with a triangular initial region.

[5]Therefore, we consider the case of small n. This proposed algorithm has a convergence rate of n^m, which is 2^m or 3^m for example. However under the assumption of *unimodality* we can reduce this convergence rate to $m + 1$, which will be discussed in section 2.7.

2.2 Optimal choice of the number of test points

In this section, we answer the question: how many initial test points n in each dimension are optimal for the minimization[6] of the total number of computations(flops)? The total number of test points required when n is small is obviously smaller than the number of test points when n is large. However the reduction ratio, which will be introduced later, when n is small, will be higher than when n is large. So what is the optimal n when we have an assumption of equi-dist grid for a general function? [7]

Note that the cost of computation usually means the number of computations. In the 1-dimensional case, consider $f : R^1 \longrightarrow R^1$ with the equi-dist grid $\mathcal{L}$ in formula 2.1 when we are given the initial interest region $\mathcal{I} = [L, U]$. Also in a similar way, in the m-dimensional case, the equi-dist grid $\mathcal{L}$ can be defined as in formula 2.2.

Let 1)m=dimension of the parameter space (fixed number), 2)n=# of grids selected in each parameter (variable), 3)r=# of iterations (# of changes of grid) (variable) and 4)N=# of total flops (object function to be minimized).

The total number of computation is represented by

$$N(n, r) = r \times n^m \tag{2.3}$$

because, in each iteration step, we have n^m values to compare. Since formula 2.3 has 2 variables, n and r, we eliminate one variable so that we can have N as a function of one variable. Note that we want to find the minimum number of n until we reach the same relative range remaining.

The relative range k remaining in each dimension compared with the initial length in each dimension after the $r\underline{th}$ iteration is represented by $k = (\frac{2}{n+1})^r$. Equivalently,

$$r = \frac{\log k}{\log (\frac{2}{n+1})} \tag{2.4}$$

If we fix k, then $N = r \times n^m$ can be written as a function of only one variable n:

---

[6] In other words, minimization of time spent to reach the same error that remains

[7] The reader should note that there is no assumption on the object function, e.g. unimodality.

$$N(n) = n^m \frac{\log k}{\log\left(\frac{2}{n+1}\right)} \tag{2.5}$$

Consider the minimization of the function [8]

$$g(t) = \frac{t^m}{\log(t+1) - \log 2} \tag{2.6}$$

where $t \in \mathcal{Z} = \{1, 2, 3, \ldots\}$. [9] See appendices 5.2 and Figure 4.5 for the minimization of $g(t)$.

The output in appendices 5.2 for the minimization of $g(t)$, $t \in \mathcal{Z}$ shows that

- for 1-dimension ($m = 1$), 3 test points ($n = 3$)[10] are optimal; and
- for m-dimension ($m \geq 2$), 2 test points ($n = 2$) are optimal.

[11]

2.3 Dichotomous Search

In previous sections 2.1 and 2.2, we dealt with the general concept of the grid search without the assumption of unimodality. From this section, [12] we shall assume that the domain of the object function for the IGS procedure has the property of *unimodality*. It will be shown that the number of test points in each iteration step, $n = 2$, is sufficient to locate the optimum point as long as the object function f is unimodal. [13]

---

[8] Note that $k > 0$ is a constant; therefore, it is not relevant to the minimization of $g(t)$.

[9] Letting $g'(t) = 0$ is equivalent to

$$m \log 2 + \frac{t}{t+1} = m \log(t+1) \tag{2.7}$$

which is difficult to find the analytical solution. Therefore we go back to $g(t)$ and use some numerical way to minimize $g(t)$.

[10] Make sure to remember this is a result *without* the assumption of unimodality. With the assumption of unimodality, it will be shown that $n = 2$ is optimal for $m = 1$.

[11] **(For further study):** FIND THE MEANING OF THE FACF THAT NOT ALWAYS 2 TEST POINTS ARE OPTIMAL.

[12] For general reference, see [1], [2], [5], [8], [14], [17], [40], [43].

[13] What is a grid search without unimodality? Without the assumption of unimodality, a grid search may be used to find such information as identifying the region in which the minimization of f is likely to lie, discovering how smooth it is, and determining whether f may have multiple local minima. [37]

We begin with the description of the algorithm and present a crucial main theorem for the application of unimodal functions. We introduce the dichotomous search before we discuss IGS, which is a direct multivariate extension of the dichotomous search, in section 2.5. The line search is the backbone of the multivariate optimization for solving a nonlinear programming problem since many multivariate algorithms assume the complete one-dimensional search.

Given an initial region of interest $\mathcal{I} = [a, b]$, as shown in Figure 0.1 on page 1, we consider the function evaluation of the two[14] test points, $x_1 < x_2$, in a given iteration step. If $f(x_1) > f(x_2)$, then the new interval of uncertainty is $[x_1, b]$ since the optimum point cannot exist in $[a, x_1]$. However, if $f(x_1) \leq f(x_2)$, then the new interval of uncertainty is $[a, x_2]$ by similar reasoning. Thus, depending on the values of f at x_1 and x_2, the length of the new interval of uncertainty is equal to $b - x_1$ or $x_2 - a$. [15]

In a *dichotomous search*[16], we place [17] each of the first two observations, x_1 and x_2, symmetrically at a distance ϵ from the midpoint $(b-a)/2$. Depending on the values of f at x_1 and x_2, a new interval of uncertainty is obtained. The process is then repeated by placing two new observations.

[14]The algorithm with $n = 3$ is fully described in section 2.6.

[15]Note, however, that we do not know in advance whether $f(x_1) < f(x_2)$ or $f(x_2) < f(x_1)$. Thus, the *optimal strategy* is to place x_1 and x_2 in such a way as to *guard* against the worst possible outcome, that is, to minimize the maximum of $x_2 - a$ and $b - x_1$. This can be done by placing x_1 and x_2 at the midpoint of the interval $[a, b]$. If we do this, we have *only one* check point and cannot reduce the interval of uncertainty. The two values x_1 and x_2 are placed symmetrically, each at a distance $\epsilon > 0$ from the midpoint. Here $\epsilon > 0$ is a scalar that is sufficiently small so that the new length of uncertainty $\epsilon + (b - a)/2$ is close enough to the theoretical optimal value of $(b - a)/2$ and, in the meantime, would make the functional evaluations $f(x_1)$ and $f(x_2)$ computationally distinguishable on digital computer. We will discuss more about the constant ϵ in next sections. [7] [20] [21]

[16]Note that the terminology *dichotomous search* is the same as *1-dimensional iterated grid search*.

[17]There can be three kinds of suggestions for how to place the two test points symmetrically. Let l be the length of the interval and ϵ be the distance between the middle and placed test points.

1. equi-distance, $\epsilon = l/6$
2. ϵ sufficiently small so that $\rho = (\epsilon + (1/2)l)/l$ is close to $l/2$.
3. any other, $\epsilon = (\alpha - 1/2)l$, where $\alpha = 0.618$ for example, will be shown later to be *optimal*.

Algorithm for the Dichotomous Search

Initialization step

Choose a distinguishability constant,[18] $2\epsilon > 0$, and an allowable final length of uncertainty, $l > 0$. Let $\mathcal{I} = [a, b]$ be the initial interval of uncertainty and assume that the initial interval $[a_1, b_1] = [a, b]$ includes the optimum point and let $k = 1$, and go to the main step.

Main Step

1. If $b_k - a_k < l$, then stop; [19] the minimum point lies in $[a_k, b_k]$.

   Otherwise consider x_1 and x_2 defined below; go to step 2.

$$x_1 = \frac{a_k + b_k}{2} - \epsilon \tag{2.8}$$

$$x_2 = \frac{a_k + b_k}{2} + \epsilon \tag{2.9}$$

2. If $f(x_1) < f(x_2)$, let $a_{k+1} = a_k$ and $b_{k+1} = x_2$. Otherwise let $a_{k+1} = x_1$ and $b_{k+1} = b_k$. Replace k by $k + 1$, go to step 1.

To show the validity of the algorithm, we use the following theorem with the assumption of unimodality.

Definition 1 *A function f is* **strictly quasi-convex** [5] *iff for each x^1, x^2 with $f(x^1) \neq f(x^2)$*

$$f(\lambda x^1 + (1 - \lambda)x^2) \leq max\ \{f(x^1), f(x^2)\} \tag{2.10}$$

Theorem 1 *Let $f : R^1 \longrightarrow R^1$ be strictly quasi-convex over $[a, b]$. Let $x_1 \leq x_2 \in [a, b]$. If $f(x_1) > f(x_2)$, then $f(z) \geq f(x_2)$ for all $z \in [a, x_1)$; and if $f(x_1) \leq f(x_2)$, then $f(z) \geq f(x_1)$ for all $z \in (x_2, b]$.* [5]

Proof 1 *Suppose that $f(x_1) > f(x_2)$ and let $z \in [a, x_1)$. Assume, on the contrary, that $f(z) < f(x_2)$. Since x_1 can be written as a convex combination of z and x_2, and by strict quasi-convexity of f, we have $f(x_1) < \underline{max}\ \{f(z), f(x_2)\} = f(x_2)$ using definition 1, contradicting $f(x_1) > f(x_2)$. Hence $f(z) \geq f(x_2)$. The second part of this theorem can be proved similarly. Q.E.D.*

[18] In this section, we consider a small $\epsilon > 0$ so that the efficiency is close to 1/2.

[19] This convergence is called *absolute* convergence. This also implies $f(b_k) \cong f(a_k)$ since f is assumed continuous.

From the previous theorem, we note that if $f(x_1) > f(x_2)$, then there must not exist an optimum point in $[a, x_1]$ since $f(z) \geq f(x_2)$ for all $z \in [a, x_1)$. Now we eliminate the region $[a, x_1)$ to get the new interval of uncertainty $[x_1, b]$ for the next iteration step. In this way, our region of interest will be reduced in each step until we reach the optimum point within an allowable final length of uncertainty. In the same way, if $f(x_1) \leq f(x_2)$, then a similar argument follows.

Theorem 2 *Consider the problem of minimizing a strictly quasi convex $f(x)$ subject to $x \in S \subset R^p$. If x is a local optimal solution, then x is also the global solution.* [5]

Proof 2 *Assume, on the contrary, that there exists an $\hat{x} \in S$ with $f(\hat{x}) < f(x)$. By the convexity of S, $\lambda\hat{x} + (1-\lambda)x \in S$ for each $\lambda \in (0,1)$. Since $\hat{x}$ is a local minimum by assumption, $f(\hat{x}) \leq f(\lambda\hat{x} + (1-\lambda)x)$ for all $\lambda \in (0, \delta)$ and for some $\delta \in (0,1)$. However because f is strictly quasi convex and $f(\hat{x}) < f(x)$, $f(\lambda\hat{x} + (1-\lambda)x) < f(x)$ for each $\lambda \in (0,1)$. This contradicts the local optimality of x and completes the proof. Q.E.D.*

Therefore, from the previous theorem, we note that if we find *one* local optimum point of a unimodal function $f(x)$, then it is necessarily *the* global optimum point. In the above two theorems, if the function to be minimized is unimodal, then we can mathematically prove that the dichotomous search algorithm finds the optimum point.

2.4 An improvement of the Dichotomous Search

A naturally-arising question from the previous section is *how shall we choose the two test points x_1 and x_2*? Two suggestions are made.

1. If the constant $\epsilon > 0$ is very small[20], then the efficiency (reduction ratio of the interval of the contiguous iteration steps) is going to change almost to 1/2, which is mathematically the best value. However this may give a computational problem in a digital computer depending on how many bits that are used in CPU. Yet note that we need 2 new test points in each iteration for the reduction ratio of almost 1/2.

[20] In other words, $x_1 \cong x_2$.

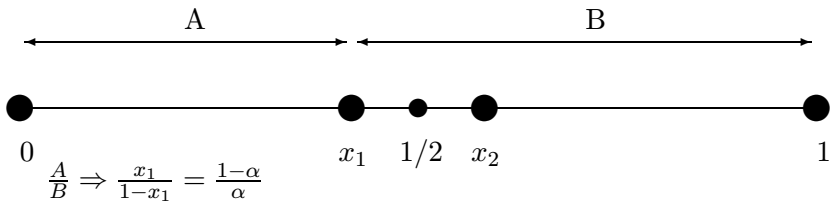

Figure 2.1: Using the fixed ratio α at kth iteration

2. Under some conditions on ϵ, we can re-use the internal test point from the previous iteration as one of the next interior test points. It is, therefore, best to place the points symmetrically from the midpoint of the interval. In this case we need *only* 1 point in each iteration step since the other test point will coincide with the previous internal test point.

From the two arguments described above, we choose the second method since it requires only 1 new test point in each iteration; thus, the efficiency[21] (regarding the ratio of interval) of the second method is better than the first one, as will be shown in formula 2.19.

Given, without loss of generality, the initial interval of uncertainty $\mathcal{I} = [0, 1]$[22], we discuss how to use the internal point. We place x_1, x_2 symmetrically around the coordinate 1/2 (midpoint of $[0, 1]$) and take two points x_1, x_2 with a fixed ratio α at the kth iteration, as we can see in Figure 2.1. To re-use the internal test point from the previous iteration as one of the next interior test points, we need the following two equations as described in Figure 2.1.

$$x_1 = 0 + (1 - \alpha) \times 1 = 1 - \alpha \tag{2.11}$$

$$x_2 = 0 + \alpha(1 - 0) = \alpha \tag{2.12}$$

[21] The efficiency is the same as the reduction ratio. Note that small efficiency implies better method.
[22] For a general case of $[L, U]$, a similar argument can be followed.

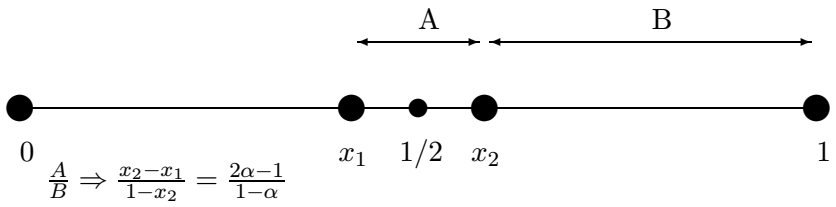

Figure 2.2: Using the fixed ratio α at *(k+1)*th iteration

In the next $(k+1)$th iteration step, x_2 coincides with the smaller test point of the two, in case the new interval of uncertainty is $[x_1, 1]$, as we can see in Figure 2.2.

$$\frac{x_2 - x_1}{1 - x_2} = \frac{1 - \alpha}{\alpha} \tag{2.13}$$

The three conditions, 2.11, 2.12 and 2.13, give the equation

$$\alpha^2 + \alpha - 1 = 0 \tag{2.14}$$

Since α must be in $(0, 1)$,

$$\alpha = \frac{3 - \sqrt{5}}{2} \cong 0.618^{23} \tag{2.15}$$

Therefore the efficiency ρ (relative ratio) using this method with $\alpha \cong 0.618$ is

$$\rho \equiv \frac{e_{n+1}}{e_n} = \frac{0.618}{1} = 0.618 \tag{2.16}$$

Using the derived α given above, we present the algorithm[24] for the improved dichotomous search.

[23]This number $\alpha = 0.618$ is called the *golden number* and the algorithm described here is called the *Golden Section Search* in industrial engineering. In industrial engineering, this algorithm can be considered easy. Meanwhile because in IE they assume m, the dimension of parameter space, is fairly big, the multivariate extension is meaningless due to the heavy time-consuming computation. However in Statistics, we are aware that many estimation problems have at most a dimension of few parameters. (e.g. Beta distribution with 3 or 4 parameters or multiple regression)

[24]This is similar to the algorithm presented on page 18 in the previous section, except the specific constant $\alpha = 0.618$.

Algorithm for the Improved Dichotomous Search

Initialization step

Choose a constant α (=0.618) and an allowable final length of uncertainty, $l > 0$. Let $[a_1, b_1] = [a, b]$ be the initial interval of uncertainty, and note that the initial interval $\mathcal{I} = [a_1, b_1]$ includes the optimum point, and let $k = 1$, and go to the main step.

Main Step

1. If $b_k - a_k < l$, then stop; the minimum point lies in $[a_k, b_k]$. Otherwise consider x_1 and x_2 defined below; go to step 2.

$$x_1 = a_k + \alpha(b_k - a_k) = (1 - \alpha)a_k + \alpha b_k \tag{2.17}$$

$$x_2 = b_k - \alpha(b_k - a_k) = \alpha a_k + (1 - \alpha)b_k \tag{2.18}$$

2. If $f(x_1) < f(x_2)$, let $a_{k+1} = a_k$ and $b_{k+1} = x_2$. Otherwise let $a_{k+1} = x_1$ and $b_{k+1} = b_k$. Replace k by $k + 1$, go to step 1.

From equations 2.17 and 2.18, note that one value of the two test points can be re-used in the next iteration. It is mathematically correct, but it will not be correct computationally, in a digital computer, due to the roundoff error.[25] Therefore, it is recommended to store 1)the location of the test point and 2)the value of one test point that will be coincided in the next iteration, and use them without computing it over again in the next iteration.

Analysis of the improved Dichotomous algorithm

Remember from section 2.3 the reduction ratio of the dichotomous search is $\rho = 1/2 + \epsilon$. Given the same $2n$ points in the two methods, the relative efficiency between the improved dichotomous search and the dichotomous search is as follows:

$$\frac{(1/2 + \epsilon)^n}{0.618^{2n}} \longrightarrow (1.309)^n > 1 \tag{2.19}$$

[25] It is almost close, but not the same digit in the computer storage of a number. A simple example of the roundoff error is shown in appendices 5.12.

as $\epsilon \longrightarrow 0$.

We conclude that, based on the $2n$ points available, the improved dichotomous search has $(1.309)^n$-times better efficiency than the dichotomous search.

2.5 Description of the IGS algorithm

In this section, we discuss a simple generalization of the improved dichotomous search. A straightforward extension of the dichotomous search requires 2^m test points[26], but using the internal points, IGS requires fewer than 2^m test points. An improvement of IGS will be discussed in section 2.7. The domain of the function on which IGS can be used might be the set of unimodality functions; however, in the Euclidean parameter space R^m, it is not easy to define a unimodal function. The terminology of unimodality is not well-defined in the area of multivariate mathematics. We choose the following definitions of unimodality, which is a straightforward extension of the definition of 1-dimensional unimodality.

Definition 2 *A function $f : R^m \longrightarrow R^1$ is called multivariate unimodal with optimum point $x^* = (a^*, b^*, \ldots)^T$ if for any 2 test points $(a_1, b_1, \ldots)$ and $(a_2, b_2, \ldots)$ where*

$$\begin{bmatrix} & a^* \le a_1 \le a_2, & \underline{or}\ a_2 \le a_1 \le a^* \\ \underline{and} & b^* \le b_1 \le b_2, & \underline{or}\ b_2 \le b_1 \le b^* \\ \underline{and} & \ldots & \ldots \end{bmatrix}, \ f(a_1, b_1, \ldots) \le f(a_2, b_2, \ldots) \text{ holds.}$$

Also we can define another definition of multivariate unimodality:

Definition 3 *A function $f : R^m \longrightarrow R^1$ is strictly quasi-convex iff for each x^1, x^2 with $f(x^1) \neq f(x^2)$*

$$f(\lambda x^1 + (1-\lambda)x^2) \le max\ \{f(x^1), f(x^2)\} \tag{2.20}$$

Consider the unimodal function $f : R^m \longrightarrow R^1$ in the multivariate parameter space with dimension $m \ge 2$; therefore, we have 2^m test points[27] possible in each iteration step. Among the

[26] since there are 2 test points in each dimension

[27] We may not need all 2^m test points. A partial list of the points is enough to apply to an unimodal function. More is discussed in section 2.7.

values for $f(x)$ at each of the 2^m test points, the surrounding cell that gives the smallest $f(x)$ will be the region of interest in the next iteration. This causes time-consuming computations. There is a cure for this problem, and an improvement will be discussed in section 2.7. How to proceed to the next iteration can be summarized as follows:

Algorithm for the IGS

Initialization step

Choose a constant, $\alpha = 0.618$, and an allowable final length of uncertainty, $l > 0$. Let $\mathcal{I} = \{(a, b, \ldots) \mid a_1 \le a \le a_2\ ,\ b_1 \le b \le b_2\ ,\ \ldots\}$ be the m-dimensional initial region of uncertainty, and let $\underline{a}^1 = a_1$, $\overline{a}^1 = a_2$, $\underline{b}^1 = b_1$, $\ldots$, and note that the initial region $\mathcal{I}$ should include the optimum point, and let $k = 1$, and go to the main step[28].

Main Step

1. If

$$max\{\overline{a}^k - \underline{a}^k, \overline{b}^k - \underline{b}^k, \ldots\} < l \tag{2.21}$$

then stop; the minimum point lies in

$$(\overline{a}^k, \underline{a}^k) \times (\overline{b}^k, \underline{b}^k) \times \ldots \in R^m \tag{2.22}$$

Otherwise consider 2^m test points (the combination of $(x_{1,1}, x_{1,2}), \ldots, (x_{m,1}, x_{m,2})$ defined below); go to step 2.

$$\begin{aligned} x_{1,1} = (1-\alpha)\underline{a}^k + \alpha\overline{a}^k \quad &, \quad x_{1,2} = \alpha\underline{a}^k + (1-\alpha)\overline{a}^k \\ x_{2,1} = (1-\alpha)\underline{b}^k + \alpha\overline{b}^k \quad &, \quad x_{2,2} = \alpha\underline{b}^k + (1-\alpha)\overline{b}^k \\ \ldots \quad &, \quad \ldots \\ x_{m,1} = \ldots \quad &, \quad x_{m,2} = \ldots \end{aligned} \tag{2.23}$$

[28]Notation: $\underline{a}^k$ and $\overline{a}^k$ represent the lower and upper bounds of variable a at k$\underline{\text{th}}$ iteration, respectively.

2. Among the values at the 2^m points of

$$(x_{1,1}, x_{2,1}, \ldots, x_{m,1}), (x_{1,2}, x_{2,1}, \ldots, x_{m,1}), \ldots, (x_{1,2}, x_{2,2}, \ldots, x_{m,2}) \tag{2.24}$$

find one point that gives the smallest value of $f(x)$, and take the surrounding cell $[a^{k+1}, b^{k+1}] \in R^m$ as the new cell for the next iteration. Replace k by $k+1$, go to step 1.

[29]

Consider the algorithm with $\alpha = 0.618$ in the $m \geq 2$ dimensional parameter space. For $m = 2$, $2^2 - 1$ new test points are required in each iteration step, and for $m = 3$, $2^3 - 1$ new test points are required in each iteration step. Thus, in the $m \geq 2$ dimensional parameter case, there is only a small improvement in the order of magnitude because $2^m - 1 \cong 2^m$ when m is a large number. This means that re-using the previous test point does not help greatly with the efficiency of IGS asymptotically. Note that, given $2^m - 1$ test points, the reduction ratio is represented by $\rho_{IGS} = \alpha^m$. The programming code of IGS is included in appendices 5.7.

2.6 3-point Grid Search

This section discusses a version of the grid search using three points in every iteration. The 3-point grid search requires three test points at every iteration step. The 3-point equi-dist grid search (3GS) in the parameter space R^1 will be shown to be *less* efficient than the improved dichotomous search.

Consider optimization of a strictly unimodal function $f : R^1 \longrightarrow R^1$. A new grid search algorithm using three test points[30] in each iteration step can be described as follows. Given equi-dist grid with $n = 3$, take values $f(x_1), f(x_2), f(x_3)$ of three test points in each iteration step and let $\mathcal{S} = \{f(x_1), f(x_2), f(x_3)\}$ as we see in Figure 2.3.

[29] Note that it is more desirable, if possible, to include the optimum point in the initial grid. If the optimum point is not in the initial grid, then IGS will find the boundary point that is, actually, the minimum value of the initial grid.

[30] without use of the re-use point

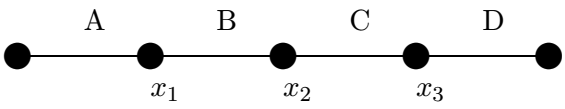

Figure 2.3: Equi-distance grid in 3GS

1. If $f(x_1)$ has the smallest value among $\mathcal{S}$, then take the interval AB for the next iteration.
2. If $f(x_2)$ has the smallest value among $\mathcal{S}$, then take the interval BC for the next iteration.
3. If $f(x_3)$ has the smallest value among $\mathcal{S}$, then take the interval CD for the next iteration.

That is, take the surrounding cell where the smallest value is located and use it in the next iteration. To take care of equality, if $f(x_1) = f(x_2) > f(x_3)$, obviously take CD, and if $f(x_1) = f(x_2) < f(x_3)$, then take[31] ABC.

But from the second iteration step on, we do not need all three test points, since the middle test point in the next iteration coincides with one of the previous test points due to the equi-dist grid. We can, therefore, re-use one test point. Thus in every iteration step except the first iteration,[32] we need two new test points.

Analysis of the 3-point grid search algorithm

Note that in the first iteration step, we need three test points, but from the second iteration step on, we need two additional test points since one test point in the previous iteration will coincide in the next iteration due to the equi-dist grid. And we note that $\rho = e_{n+1}/e_n = 1/2$ with two points required in every iteration.

[31] If f is strictly unimodal, then obviously take only B.
[32] Note this does not affect the overall efficiency asymptotically.

Comparison with the improved Dichotomous Search in the 1-dimensional parameter space

Remember that the improved Dichotomous Search gives the efficiency $\rho = 0.618$ with one test point. The 3-point grid search method gives the efficiency $\rho_{3GS} = 0.5$ with two test points.

Thus, for $2n$ available points[33] in each method, 1)improved Dichotomous Search gives the reduction of interval $(0.618)^{2n}$ after $2n$ iterations while 2)3GS gives the reduction of interval $(0.5)^n$ after n iterations.

The relative efficiency between the two procedures is as follows:

$$1 < \frac{0.5^n}{0.618^{2n}} = 1.32^n \tag{2.25}$$

We conclude that the improved Dichotomous Search is better[34] than the 3-point grid search by 1.32 times in the 1-dimensional parameter space.

Comparison with IGS in m-dimensional parameter space

Given the parameter dimension m, the number of test points required in each iteration is as follows:

dim	IGS	3GS
$m = 1$	$2^1 - 1 = 1$	$3^1 - 1 = 2$
$m = 2$	$2^2 - 1 = 3$	$3^2 - 1 = 8$
$m = 3$	$2^3 - 1 = 7$	$3^3 - 1 = 26$
...	...	...
$m = m$	$2^m - 1$	$3^m - 1$

Note that, from the above chart, only 1 test point can be re-used in the next iteration for each method. The reduction ratio in IGS is given by $\rho = \alpha^m$ with $2^m - 1$ test points; the reduction

[33] same computing time for comparison of the two methods.

[34] as far as the comparison of the ratio. Of course both methods provide linear convergence with $p = 1$, described in section 2.8.

ratio in the 3GS is given by $\rho = (1/2)^m$ with $3^m - 1$ test points. [35]

Given the same number of test points k, 1)IGS requires $k/(2^m - 1)$ iterations,[36] 2)3GS requires $k/(3^m - 1)$ iterations. Therefore given k, IGS and 3GS have reductions of region, respectively:

$$r_{IGS} = (\alpha^m)^{\frac{k}{2^m-1}} \ , r_{3GS} = (0.5^m)^{\frac{k}{3^m-1}} \tag{2.27}$$

We now evaluate the ratio:

$$1 < \frac{r_{3GS}}{r_{IGS}} = \frac{0.5^{\frac{mk}{3^m-1}}}{\alpha^{\frac{mk}{2^m-1}}} \tag{2.28}$$

To eliminate the constant k, we use the new measure[37] (ratio of the logs)

$$1 < \frac{\log r_1}{\log r_2} = \frac{3^m - 1}{2^m - 1} \times \frac{\log \alpha}{\log 0.5} \ \longrightarrow \infty \tag{2.29}$$

as $m \longrightarrow \infty$

Using $\log \alpha / \log 0.5 = 0.7176$ we get

dimension	ratio
$m = 1$	1.43
$m = 2$	1.91
$m = 3$	2.63
$\ldots$	$\ldots$
$m = \infty$	∞

We conclude that IGS is better than the 3-point grid search with the ratio values given above. [38]

---

[35] Also in a similar way, the relative efficiency between the two procedures in m-dimensional case is *not*

$$\frac{(1/2)^{mn}}{(0.618)^{2mn}} = (1.32^m)^n \tag{2.26}$$

since $3^m - 1 \neq 2 \times (2^m - 1)$ if $m > 1$. Note that it is not easy to figure out the least common multiple of the $(2^{m_1}, 3^{m_2})$ for $m_1 = 1, 2, \ldots,\ m_2 = 1, 2, \ldots$.

[36] Of course we disregard that the first iteration has one more test point. For more exact calculation use gauss function $f(x) = [x]$ (biggest integer less than x). But the result does not change asymptotically.

[37] It is hard to find the *physical* meaning in the measure of this ratio. We note that the ratio of the logs is used only to compare the 2 methods. Also note that the intentional use of the logarithm makes the new measure free from the constant k.

[38] **Another method of comparison**
Another way to analyse the performance of 3GS in comparison with the IGS can be proposed as follows: Given the

2.7 An improvement of the IGS - a sequential approach

In this section, we discuss an improvement of the IGS algorithm. By the proposed definition of a unimodal function, the sequential iterated grid search (SIGS) gives the same solution as the IGS algorithm but has a significant improvement in speed over IGS.

Note the notation $a = (a(1), a(2)) \in R^2$, $A, B, C, D \in R^1$ in Figure 2.4.

Theorem 3 *Consider the optimization of the k-convex function $f : R^2 \longrightarrow R$, given the initial region*[39] $[a(1), b(1)] \times [a(2), b(2)]$ *with two points $a = (a(1), a(2)),\ b = (b(1), b(2))$ as shown in Figure 2.4 on page 30. Consider $A, B \subset R^1$ where $a(1) < A < B < b(1)$. If $f(A, a(2)) > f(B, a(2))$*[40]*, then the optimum point exists in the region $\mathcal{S} = [A, b(1)] \times [a(2), b(2)]$ and therefore we can delete the region $\mathcal{S}^C = [a(1), A] \times [a(2), b(2)]$. Furthermore it holds for m-dimensional k-convex function*

---

same relative range remaining, we can compare the number of computations(flop) required. That is, if one algorithm requires a smaller number of computations than the other, then the first algorithm is better than the second.

For both IGS and 3GS, the total number of flops can be written by

$N(n, r)$	when
$= n + (n-1) + \ldots + (n-1)$	$m = 1$
$= n^2 + (n^2-1) + \ldots + (n^2-1)$	$m = 2$
$\ldots$	$\ldots$
$= n^m + (n^m-1) + \ldots + (n^m-1)$	$m = m$

$$N(n, r) = n^m + (r-1)(n^m - 1) = r \cdot n^m + (1 - r) \tag{2.30}$$

- For IGS with $n = 2$, the relative range remaining is $k = \alpha^r$, which is equivalently $r = (\log k)/(\log \alpha)$. The total number of flops in IGS is

$$\begin{aligned} N_1 \quad = N(n = 2, r) \quad &= 2^m + (r-1)(2^m - 1) \\ &= 2^m \log_\alpha k + (1 - \log_\alpha k) \end{aligned} \tag{2.31}$$

- For 3GS with $n = 3$, the relative range remaining is $k = (1/2)^r$, which is equivalently $r = -\log_\alpha k$. The total number of flops in 3GS is

$$\begin{aligned} N_2 \quad = N(n = 3, r) \quad &= 3^m + (r-1)(3^m - 1) \\ &= 2^m(-\log_2 k) + (1 + \log_2 k) \end{aligned} \tag{2.32}$$

- We can show either $0 < N_2 - N_1$ or $1 < N_2/N_1$ with respect to m, k (function of two variables).

[39] There is a change of notation since this notation will simplify to the description of algorithm.

[40] This can be called *marginal condition.*

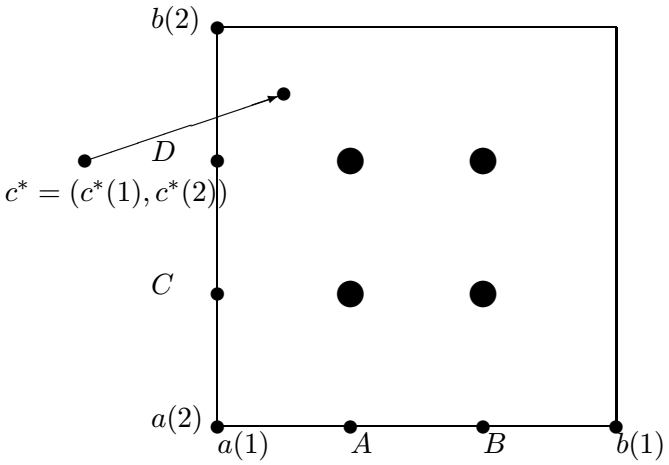

Figure 2.4: For theorem 3 in SIGS

$f : R^m \to R^1$, $m > 2$.

Proof 3 *Assume 1)f is k-convex, 2)*$f(A, a(2)) > f(B, a(2))$*, and 3)there exist an optimum* c^* *outside* $\mathcal{S}$*. Then* $c^*(1) < A < B$*. This is a contradiction*[41]*. Therefore the optimum point* c^* *cannot exist outside* $\mathcal{S}$*. A similar argument can be given for dimensions* $m = 3, 4, \ldots$*. Q.E.D.*

Therefore we conclude that, due to the strong condition of unimodality defined in definition 2 on page 23, even the sequential modification (simplification) works, as shown in theorem 3. This theorem will be used to simplify the IGS programming.

Algorithm for the SIGS [42]

[41] Since $c^*(1) < A < B$, it should be $f(A, a(2)) < f(B, a(2))$ by definition of k-convexity. This can be easily verified by letting $a^* = c^*(1)$, $a_1 = A$, $a_2 = B$ and $b_1 = b_2 = a(2)$ in definition 2 on page 23.

[42] We have presented

- Algorithm for the Dichotomous search in section 2.3 on page 18
- Algorithm for the improved Dichotomous search in section 2.4 on page 22
- Algorithm for the IGS in section 2.5 on page 24
- Algorithm for the SIGS in section 2.7 here.

Initialization step

Choose an allowable final length of uncertainty, $l > 0$. Let $2 \times m$ matrix I be the initial region of uncertainty, and note that the region I includes the optimum point. Get the constant $\alpha = 0.618$, let $k = 1$, and go to the main step.

Main Step

1. If

$$max\{\overline{a}^k - \underline{a}^k, \overline{b}^k - \underline{b}^k, \ldots\} < l \tag{2.33}$$

then stop; the minimum point lies in

$$(\overline{a}^k, \underline{a}^k) \times (\overline{b}^k, \underline{b}^k) \times \ldots \in R^m \tag{2.34}$$

Otherwise consider 2^m test points (the combination of $(x_{1,1}, x_{1,2}), \ldots, (x_{m,1}, x_{m,2})$ defined below); go to step 2.

$$\begin{aligned} x_{1,1} = (1-\alpha)\underline{a}^k + \alpha\overline{a}^k \quad &, \quad x_{1,2} = \alpha\underline{a}^k + (1-\alpha)\overline{a}^k \\ x_{2,1} = (1-\alpha)\underline{b}^k + \alpha\overline{b}^k \quad &, \quad x_{2,2} = \alpha\underline{b}^k + (1-\alpha)\overline{b}^k \\ \ldots \quad &, \quad \ldots \\ x_{m,1} = \ldots \quad &, \quad x_{m,2} = \ldots \end{aligned} \tag{2.35}$$

2. Among the values at the $m+1$ points [43] of

$$(x_{1,1}, x_{2,1}, \ldots, x_{m,1}), (x_{1,2}, x_{2,1}, \ldots, x_{m,1}), \ldots, (x_{1,1}, x_{2,1}, \ldots, x_{m,2}) \tag{2.36}$$

find one point that gives the smallest value of $f(x)$, and take the surrounding cell $[a^{k+1}, b^{k+1}] \in R^m$ as the new cell for the next iteration. Replace k by $k+1$, go to step 1.

In m-dimensional parameter space, the new algorithm, SIGS, requires $m+1$ test points in each iteration step; whereas the IGS requires 2^m ($\gg m+1$) test points, as discussed in section 2.5. But note that the reduction ratio $\rho = e_{n+1}/e_n$ of SIGS will be the same as the previous IGS.

---

[43] Reader should note that IGS, presented in section 2.5, need 2^m test points whereas SIGS need *only* $m+1 \ll 2^m$ test points.

We conclude that, among the algorithms described in this chapter, the SIGS is recommended since it is reasonably simple and fairly efficient.

2.8 Comparisons with other algorithms

In this section we discuss the effectiveness of the proposed algorithms, IGS and SIGS, together with some important factors that must be considered when assessing the effectiveness of these algorithms and when comparing them. These factors are 1)generality, reliability, and precision; 2)sensitivity to parameters and data; 3)preparational and computational effort; and 4)mathematical order of convergence. [10]

Reliability and Precision

Given any algorithm, it is not difficult to construct a test problem (counterexample) that it cannot solve effectively. Reliability, or robustness, means the ability of the procedure to solve most of the problems in the class for which it is designed with reasonable accuracy. Usually, this characteristic should hold regardless of the initial region used.

Convergence of nonlinear programming algorithms usually occurs in a limiting sense, if at all. Thus, we are interested in measuring the quality of the estimated points produced by the algorithm after a reasonable number of iterations. Algorithms that quickly produce feasible solutions with good objective values are preferred.

In view of the reliability, the IGS algorithm is highly recommended for a strictly quasi-convex function optimization, since the algorithm has guaranteed convergence as discussed in section 2.3.

Sensitivity to parameters and data

For most algorithms the user must set initial values for certain parameters, such as the starting vector, the step size, the acceleration factor and conditions for terminating the algorithm. Some procedures are quite sensitive to these parameters and to the problem data. This sensitivity may produce different results or stop the procedure prematurely. In particular, for a fixed set of selected

parameters, the algorithm should solve the problem for a wide range of problem data and should be scale invariant, that is, insensitive to any constraint or variable scaling that might be used. Likewise, for a given set of problem data, one would prefer that the algorithm not be sensitive to the selected values of the parameters.

For unimodal functions, the IGS algorithm finds a true solution if the initial guess of the interval contains the real solution, as we have shown with a mathematical proof. If the optimum value is outside the interval, then the point obtained would be the optimum point in the region which should be on the border of the region, indicating the poor choice of the initial region.

For almost unimodal functions to be defined in chapter 3, the IGS algorithm may give slightly different answers depending on the selection of the initial interval. The solution using the IGS algorithm is valuable, since except for the IGS, virtually no other algorithm can produce a reasonable answer. We also show that, for an almost unimodal function, the IGS algorithm provides a solution close to the correct solution.

Preparational and Computational Effort

Another basis for comparing algorithms is the total effort, both preparational and computational, expended in solving problems. The effort of preparing the input data should be taken into consideration when evaluating an algorithm. An algorithm that uses first- or second-order derivatives, especially if the original functions are complicated, requires a considerably larger amount of preparation time than one that only uses functional evaluations. The computational effort of an algorithm is usually assessed by the computer time, the number of iterations, and the number of functional evaluations. However, any of these measures, by itself, is not entirely satisfactory. The computer time needed to execute an algorithm depends not only on its efficiency, but also on the type of matching used, the character of the measured time, the existing load on the machine, and the efficiency of coding. Also, the number of iterations cannot be used as the only measure of effectiveness of an algorithm because the effort per iteration may vary considerably from one procedure to another.

The simple structure of the IGS algorithm gives merit to its preparational effort, since the IGS

algorithm does not require derivatives. The IGS algorithm, however, is relatively slow since it gives only a linear convergence [4], which will be discussed below. This fact is a disadvantage of the IGS algorithm.

Order of Convergence

The *theoretical convergence* of algorithms to the point in the solution set is a highly desirable property. Given two competition algorithms that converge, a theoretical comparison could be made on the basis of the mathematical order or the speed of convergence.

Let the sequence[44] $\{x_k, k = 1, 2, \ldots\} \subset R^1$ converge to $\overline{x} \in R^1$. The *order of convergence* [4] [5] [27] [16] of the sequence is defined as the supremum of the nonnegative numbers p satisfying

$$\limsup_{k \to \infty} \frac{|x_{k+1} - \overline{x}|}{|x_k - \overline{x}|^p} = \beta < \infty \tag{2.37}$$

- If $p = 1$, the sequence is said to have *linear convergence* [45] if the convergence ratio β is less than 1.
- If $p > 1$, or $\beta = 0$, the sequence is said to have *superlinear convergence* [46] which means asymptotically faster than linear convergence.
- In particular, if $p = 2$ and $\beta < \infty$, then the sequence is said to have a second-order or a *quadratic* rate of convergence.[47]

[44] In fact this sequence is defined in R^1. In a higher dimension, $x_k \in R^m$, we shall have the appropriate convergence with the corresponding norm $\| \cdot \|$.

[45] **Example of a sequence with linear convergence**: Consider a sequence $x_{k+1} = (x_k + 1)/2$, where $\{x_k\} \longrightarrow 1$. Hence $(x_{k+1} - 1) = (x_k - 1)/2$; and so, with $p = 1$, the limit in definition 2.37 is $\beta = 1/2$. However, for $p > 1$, this limit is infinity. Consequently, $\{x_k\} \longrightarrow 1$ only linearly.

[46] **Example of a sequence with superlinear convergence**: On the other hand, suppose that we have $x_{k+1} = 1 + (x_k - 1)/2^k$, for $k = 1, 2, \ldots$, where $x_1 = 4$, say. For the sequence $\{4, 2.5, 1.75, 1.375, 1.1875, \ldots\}$ obtained above, we now produce the sequence $\{4, 2.5, 1.375, 1.046875, \ldots\}$. The sequence is readily verified to be converging to unity. However, we now have $\|x_{k+1} - 1\|/\|x_k - 1\| = 1/2^k$, which approaches 0 as $k \to \infty$. Hence, $x_k \longrightarrow 1$ superlinearly in this case. Note that the sequence in the previous example has the same limit with a different convergence rate.

[47] If the limit in definition 2.37 exists, then, for large values of k, we asymptotically have $|x_{k+1} - \overline{x}| = \beta |x_k - \overline{x}|^p$, which indicates faster convergence for larger values of p. For the same value of p, the smaller the convergence ratio β

Note that our IGS algorithm provides *linear convergence* with $p = 1$ and with a constant[48] $\beta = \alpha^1, \alpha^2, \ldots, \alpha^m, \ldots$ depending on the parameter dimension m. The algorithm SIGS also has the same mathematical order of convergence as IGS but is less time-consuming than IGS. This rate is the linear convergence where $p = 1$, $\beta < 1$ which is described in definition 2.37. This means that the IGS algorithm has a slower rate of convergence than the majority of the derivative methods, many of which provide a quadratic convergence rate. But we should note that the derivative method does not converge in many cases whenever there is chaotic behavior in the object function, as shown in an introductory example in chapter 1.

2.9 Discussion

The procedures discussed above all depend on the unimodal assumption. In many problems, this assumption does not hold true, and furthermore, it cannot be easily verified in the Euclidean space R^m, $m \geq 3$. We propose two ways of solving the optimization problem if we do not have enough confidence in the unimodality of a function.

One way to handle this, especially if the initial interval of uncertainty is large, is to divide it into smaller intervals, to find the minimum over each subinterval, and then to choose the smallest of the minima over the subintervals if we think the object function is piecewise unimodal. Alternately, one can simply apply the IGS method assuming the unimodality of the function in the selected small interval and allow the procedure to converge to some local minimum solution.

A second way to handle this difficulty can be found using a *convex envelope*. For a given function, we attempt to find an outer (lower) relaxation (approximation) for the given function. The upper bound of the set of the lower approximations is called the convex envelope.[49] When we find the convex envelope of the given function, we can use our proposed IGS algorithm since the

---

is, the faster the convergence. It should be noted, however, that the order of convergence and the ratio of convergence must *not* be solely used for evaluating algorithms that converge, since they represent the progress of the algorithm only as the number of iterations approach infinity.

[48] Note that in the case of equi-dist grid we use $\alpha = 2/3$.

[49] The convex envelope will be introduced in chapter 3.

envelope function is unimodal. [50] It has been shown that there exist *only one* convex envelope of a given function and the envelope has the *same* optimization (minimum) point as in the original function. [13] But in many cases, finding *the* convex envelope of a given function is more difficult than directly finding the minimum of the object function. Further discussion of the convex envelope is postponed until chapter 3. [51]

[50] It is even convex.

[51] If we consider many algorithms that are used in Industrial Engineering, the IGS algorithm may not be considered as one of the very best overall, due to its slow performance of the speed, but as far as statistics is concerned for the special case of an almost unimodal function and for relatively smaller parameter dimensions, we have enough reason to present this special case of an almost unimodal function as discussed earlier.

Chapter 3

Literature Review On Finding Convex Criteria

In this chapter, we introduce and discuss the property of a class of functions that can be solved using the IGS algorithm. We begin with the definition of convexity and discuss its modifications[1] together with the relation between the various types of convexity, as will be shown in sections 3.1 and 3.2.

Given any function, it is known that there exist a convex envelope that is *the* best convex approximation lower than the original function. Therefore, if we can find the convex envelope, then we can use the IGS algorithm to find the optimization point, since the envelope has the same minimum as the original function.

3.1 Literature Review

In this section, we begin with the basic definition of a convex function, which will be the basis for the modifications of convexity. We need to define a convex set since most convex functions are defined on a convex set.

Definition 4 *A set $K \in R^m$ is called convex iff for any finite subset $\underline{x}_1, \underline{x}_2 \in K$ and for any real numbers, $0 < \lambda < 1$, $\lambda \underline{x}_1 + (1-\lambda)\underline{x}_2 \in K$*

Thus, a set K is convex iff it is closed under taking any convex combinations. It can be shown that this holds for any finite combination. Geometrically, a set K is convex iff, for every two points in K, the line segment between the two points is contained in K. [29] [30]

[1] weaker condition than convexity

Definition 5 *The* **Convex hull** *of a set H is defined as the collections of all convex combination of pairs of elements of the set H.*

This definition of the convex hull of a set H is equivalent to the minimal convex set containing H. The convex hull will be used to find the convex envelope discussed in section 3.3.

Convex function

Definition 6 *A function $f : K \longrightarrow R^1$ is convex* [2] *on a convex set $K \subset R^m$ iff for $\forall \underline{x}_1, \underline{x}_2 \in K$ and $0 < \lambda < 1$,*

$$f(\lambda \underline{x}_1 + (1-\lambda)\underline{x}_2) \ \leq \ \lambda f(\underline{x}_1) + (1-\lambda) f(\underline{x}_2) \tag{3.5}$$

We can now give an explicit condition of *sufficiency* for the convexity of a function. A *sufficient* condition for the convexity of a function is that K be open and the Hessian matrix be nonnegative definite for all $\underline{x} \in K$. Furthermore, if the Hessian matrix H is strictly positive definite for all $\underline{x} \in K$, then the function f is strictly convex.

Theorem 4 Nonnegative linear combination of convex functions
Let f_1, f_2 be convex and let a_1, a_2 be positive or zero. Then $f = a_1 f_1 + a_2 f_2$ is convex. Furthermore, if f_1 is strictly convex, f_2 convex, and a_1 is positive, then f is strictly convex.

Lemma 1 *Let f_i , $i = 1, \ldots, n$ be convex on a open convex set K. If $a_i \geq 0$, $i = 1, \ldots, n$, then $f(x) = \sum a_i f_i(x)$ is convex on K.*

Theorem 5 Convex Composite
Let f be a convex function from R^p to $(-\infty, \infty]$ and let φ be a convex function from R^1 to $(-\infty, \infty]$ that is <u>non-decreasing</u>. Then $h(\underline{x}) = \varphi(f(\underline{x}))$ is convex on R^p.

---

[2] Some simple examples of convex functions are

$$f(x) = x^{-m}, \ x \in (0, \infty), m \geq 0 \tag{3.1}$$

$$f(x) = e^{ax} \ , \ x \in R^1 \tag{3.2}$$

$$f(\underline{x}) = \|\underline{x}\|^m, \ \underline{x} \in R^p, m \geq 1 \tag{3.3}$$

$$f(\underline{x}) = \underline{x}' Q \underline{x}, \ \underline{x} \in R^p \tag{3.4}$$

where a matrix Q is nonnegative definite.

Proof 1 *For* $\underline{x}, \underline{y} \in R^p$ *and* $0 < \lambda < 1$, *we have* $f((1-\lambda)\underline{x} + \lambda\underline{y}) \leq (1-\lambda)f(\underline{x}) + \lambda f(\underline{y})$. *Applying* φ *to both sides of this equality, we get*

$$\begin{aligned} h((1-\lambda)\underline{x} + \lambda\underline{y}) &\leq \varphi((1-\lambda)f(\underline{x}) + \lambda f(\underline{y})) \\ &\leq (1-\lambda)h(\underline{x}) + \lambda h(\underline{y}) \end{aligned} \tag{3.6}$$

which proves the convexity of $h : R^p \longrightarrow R^1$. *Q.E.D.*

Definition 7 *The* **epigraph** *of a function* $f : R^p \longrightarrow R$ *is defined as the set* $epi\ (f) = \{(u, a) \in R^p \times R \mid f(u) \leq a\} \in R^{p+1}$.

Proposition 1 *A function is convex iff its epigraph is convex.*

Definition 8 *A differentiable function* $f : R^1 \longrightarrow R^1$ *is called* **pseudo-convex** *iff* $f'(x_1)(x_2 - x_1) \geq 0$ *implies* $f(x_1) \leq f(x_2)$ *for* $x_1, x_2 \in C$.

Definition 9 *A function* f *is called* **quasi-convex** *iff for* $x_1, x_2 \in C$ *and* $0 < \lambda < 1$, $f(x_2) \leq f(x_1)$ *implies* $f(x) \leq f(x_1)$ *for all* $x \in [x_1, x_2]$, *i.e. iff* $f(x_2) \leq f(x_1)$ *implies* $f((1-\lambda)x_1 + \lambda x_2) \leq f(x_1)$.

Theorem 6 *A function* f *is quasi-convex iff, for* $0 \leq \lambda \leq 1$, $f(x^2) \leq f(x^1)$ *implies*

$$f(\lambda x^1 + (1-\lambda)x^2) \leq f(x^1) \tag{3.7}$$

Definition 10 *A function* f *is said to be* **quasi-convex** *iff*

$$f(\lambda x^1 + (1-\lambda)x^2) \leq max\ \{f(x^1), f(x^2)\} \tag{3.8}$$

Definition 11 *A function* f *is* **strictly quasi-convex** *iff for each* x^1, x^2 *with* $f(x^1) \neq f(x^2)$

$$f(\lambda x^1 + (1-\lambda)x^2) \leq max\ \{f(x^1), f(x^2)\} \tag{3.9}$$

Definition 12 *A function* f *is* **strongly quasi-convex** *iff for each* x^1, x^2 *with* $x^1 \neq x^2$

$$f(\lambda x^1 + (1-\lambda)x^2) \leq max\ \{f(x^1), f(x^2)\} \tag{3.10}$$

By the definition of a function, $x_1 = x_2$ implies $f(x_1) = f(x_2)$. Using the contraposition, $f(x_1) \neq f(x_2)$ implies $x_1 \neq x_2$. Therefore the condition of strong q-convexity is *stronger* than that of strict q-convexity. Note that 1)there is *no* assumption of differentiability, 2)with differentiability assumptions, the definition of q-convexity can be given in terms of three *collinear* points, and 3)q-convex functions may be discontinuous. Any increasing function and decreasing function are both q-convex and p-convex. The hierarchy of the convexities [11] is given in Figure 3.1.

Proposition 2 *If a function f is convex, then f is p-convex. And if a function f is p-convex, then f is q-convex.*

Proposition 3 *A function $f : R^p \longrightarrow R^1$ is q-convex iff $f(\underline{x}^2) \leq f(\underline{x}^1)$ implies*

$$(\underline{x}^2 - \underline{x}^1)^T \bigtriangledown f(\underline{x}^1) \leq 0 \tag{3.11}$$

Proposition 4 *If a function f is convex, then f is q-convex.*

One way to define the almost unimodality can be done by finding δ in δ-unimodality. [8]

Definition 13 *A function $f : R^1 \longrightarrow R^1$ is called* **strictly δ-monotonic increasing** *iff $x_1 + \delta < x_2$ implies $f(x_1) < f(x_2)$.*

Definition 14 *A function f is called* **δ-unimodal** *(almost unimodal with δ) iff for $x_0, x_1, x_2 \in D$, and for $x_0 + \delta < x_1$ and $x_1 + \delta < x_2$, 1)$f(x_0) \leq f(x_1)$ implies $f(x_1) < f(x_2)$* **or** *2)$f(x_1) \geq f(x_2)$ implies $f(x_0) > f(x_1)$.*

3

4

[3]Given a constant δ, the IGS algorithm works fine on δ-unimodal function f as long as the distance between the two test points is bigger than δ.

[4]**(Further Study) A stochastic measure of almost unimodality** As we can see in the previous sections, we need to define the measure of how much almost unimodal the object function f is. A version of the proposed measure for $f : R^1 \longrightarrow R^1$ can be suggested as follows.

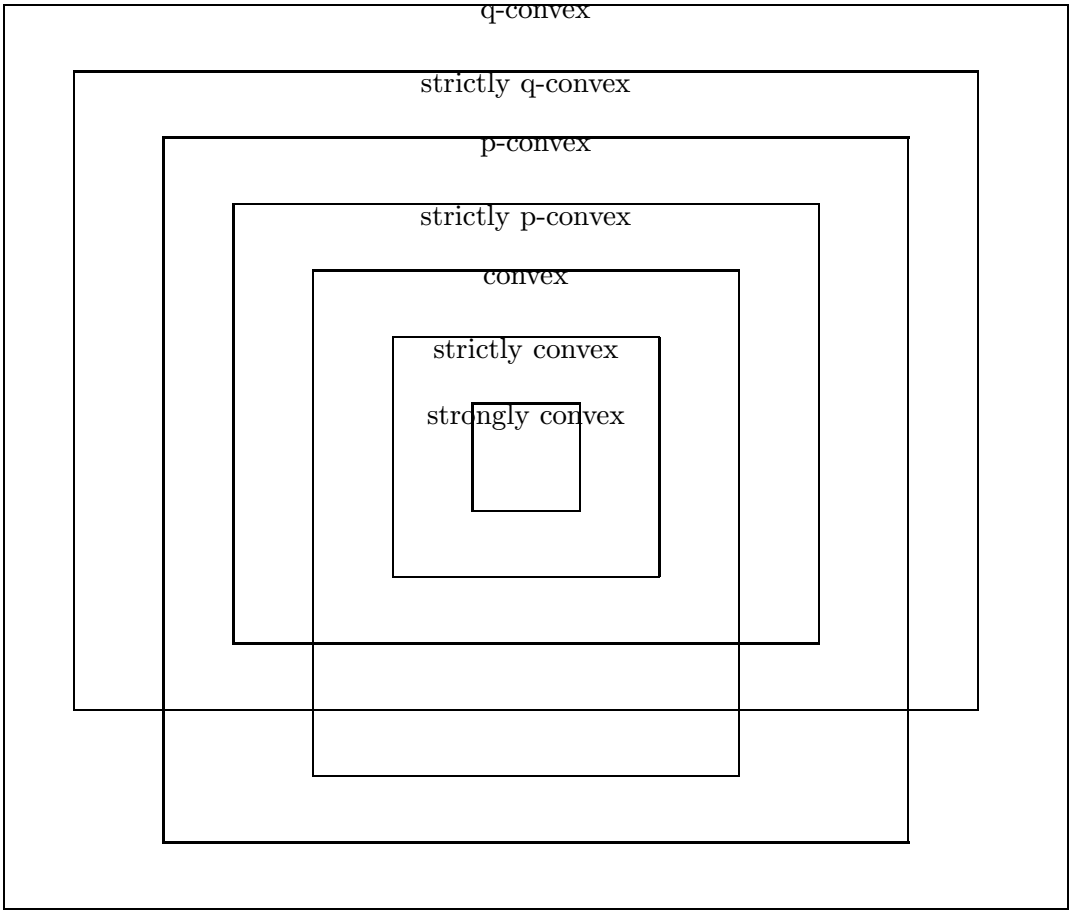

Figure 3.1: Hierarchy of the modifications of convexity

5

Multivariate unimodality (A parametric approach)

We shall now discuss a version of a multivariate generalization of the univariate unimodality. [40] For the parametric approach, we have the mathematical definition of a *path*. By a path from a point b_1 and a point b_2 in the experimental region, we mean any continuous curve whose end-points are the points b_1 and b_2 and which lies entirely in the experimental region.

Consider a parametric form of the object function $Y(\lambda) = Y(x_1(\lambda), \ldots, x_p(\lambda))$, to be optimized, with parameter $\lambda \in R^1$. Of course, not all optimization problems can be represented as a parametric approach. For the optimization problem $Y^* = \underline{min}\, Y(\vec{x})$, where $Y(\vec{x^*}) \equiv Y^*$ and $\vec{x} \in R^p$, we shall define the unimodality for a function of p variables.

Definition 15 *$Y(\vec{x})$ is called unimodal if, for every pair of points $\vec{a}$ and $\vec{b}$ in the experimental region, there exists* a *path from $\vec{a}$ through the optimum point $\vec{x^*}$ to $\vec{b}$ on which Y is unimodal.*

Definition 16 *The path is called strictly rising if $\lambda_1 < \lambda_2$ implies $Y(\lambda_1) < Y(\lambda_2)$.*

Now we can show that $Y(\vec{x})$ is unimodal if, for every point $\vec{a}$ in the experimental region, there

---

Given an interval $\mathcal{I} = [a, b]$, consider an equi-distance grid of $\mathcal{L} = \{x_1, \ldots, x_n\}$ of size n at each iteration step as defined in formula 2.1. Consider the set of n points $\{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$. If the discrete graph connecting $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$ shows unimodality, then the object function f is said to be *discretely unimodal* up to n. That is,

$$x_i = \frac{1}{2}\frac{b-a}{n} + (i-1)\frac{b-a}{n} = \frac{b-a}{n}(i - \frac{1}{2}) \tag{3.12}$$

where $i = 1, \ldots, n$. If it is discretely unimodal up to n, then we can say it is almost unimodal with n on $[a, b]$. And consider $\alpha = 1 - 1/n$. Note that the proposed measure α, $0 < \alpha < 1$, is suggested so that $\alpha \longrightarrow 1$ if $n \longrightarrow \infty$ and $\alpha \longrightarrow 0$ if n is a small integer. Obviously $\alpha = 1$ produces the proper unmodality. And note that this definition depends on the interval $\mathcal{I} = [a, b]$.

5

Example 2 *Find an example of a function $f : R^2 \longrightarrow R^1$ that is q-convex but not convex.*

$$f(x, y) = x(x-1)^3 + y^3 \tag{3.13}$$

defined on $\mathcal{I} = \{(x, y) \mid -1 \leq x \leq 2,\ -1 \leq y \leq 1\}$.

exists a strictly decreasing path to the minimum at $\vec{X}^*$.[6] Any search method that keeps to strictly decreasing paths must eventually reach the minimum point of the unimodal function.

A function is called *strongly unimodal* if the straight line to the base x^* from *any* point in the experimental region is a decreasing path. A function is called *linearly unimodal* if *any* straight line in the experimental region (not just those through the base) is unimodal.

3.2 K-convexity

In this section we discuss a property of a function that allows the application of the IGS algorithm.

one-dimensional K-convexity

We begin with the following definition before we introduce a couple of unimodality modifications. The basic assumption on the object function of the current project is as follows.

Definition 17 *Suppose* $f(a^*) = min\ f(a)$. [7] *A function* $f : R^1 \longrightarrow R^1$ *is called* **k-convex** *if* $f(a_1) \leq f(a_2)$ *for* $a^* \leq a_1 \leq a_2$ *or* $a_2 \leq a_1 \leq a^*$.

In other words, on the left side of optimum point a^*, the object function f is monotone decreasing, and on the right side of optimum point a^*, the object function f is monotone increasing. In the case of 1-dimensional parameter space, the previous definition is equivalent to the definition: there exists some x^* such that if $(x_1 - x^*)(x_2 - x^*) < 0$, then $f'(x_1)f'(x_2) < 0$ under the assumption of differentiability of f.

The definition of **K-convexity** seems a natural suggestion for the IGS algorithm, although it is not readily verified. Actually k-convexity is proposed so that, no matter where the initial region of interest containing the optimum point, the IGS algorithm converges after some appropriate rule on the iterations. In reverse, it is quite reasonable to ask what required condition of the function will

[6] This definition is equivalent to the previous definition.

[7] Keep the notation of a rather than x, since it will be extended to $a, b, c, \ldots$ later.

ensure that the iterated grid search converges. The answer might be even weaker than k-convexity. Other convexities, e.g. p-convexity and q-convexity, related to k-convexity can be considered. Note that k-convexity is close to strictly quasi-convexity. Usually the form of the loss function and the fitting is differentiable. That is, we do not have a non-differentiable point. So we are free to choose any differentiable functions.

Definition 18 *A function* $f : R^1 \longrightarrow R^1$ *is called* **strongly unimodal** [5] *on* $[a, b]$ *if there exists a* x^* *that minimizes* f *and for any* x_1, x_2 *such that* $x_1 < x_2 \in [a, b]$ *we have 1)*$x_2 \leq x^*$ *implies* $f(x_1) > f(x_2)$ *and 2)*$x^* \leq x_1$ *implies* $f(x_1) < f(x_2)$.

Therefore k-convexity is equivalent to strong unimodality.

Definition 19 *A function* $f : R^1 \longrightarrow R^1$ *is called* **strictly unimodal** *on* $[a, b]$ *if there exists a* x^* *that minimizes* f *and for any* x_1, $x_2 \in [a, b]$ *such that* $f(x_1) \neq f(x^*), f(x_2) \neq f(x^*)$ *we have 1)*$x_2 \leq x^*$ *implies* $f(x_1) > f(x_2)$ *and 2)*$x_1 \geq x^*$ *implies* $f(x_1) < f(x_2)$.

Proposition 5 *If a function* f *is strongly unimodal then* f *is strictly unimodal.*

Definition 20 *A function* $f : R^1 \longrightarrow R^1$ *is called* **unimodal** [8] [9] *on* $[a, b]$ *if there exists a* x^* *that minimizes* f *and for any* x_1, $x_2 \in [a, b]$ *such that* $f(x_1) \neq f(x^*)$, $f(x_2) \neq f(x^*)$ *and* $x_1 < x_2$ *we have 1)*$x_2 \leq x^*$ *implies* $f(x_1) > f(x_2)$ *and 2)*$x^* \leq x_1$ *implies* $f(x_1) < f(x_2)$.

Multivariate k-convexity

The multivariate extension of the univariate k-convexity can now be given for the current project.

Definition 21 *A function* $f : R^p \longrightarrow R^1$ *is called multivariately k-convex with optimum point* $x^* = (a^*, b^*, \ldots)^T \in R^p$

[8] Show that if f is strongly unimodal, then f is strongly quasi-convex. Conversely, show that if f is strongly quasiconvex and has a minimum in this interval, then it is strongly unimodal over the interval. [5]

[9] Show that if f is unimodal and continuous, then f is strictly quasiconvex. Conversely, show that if f is strictly quasiconvex and has a minimum in this interval, then it is unimodal over this interval. [5]

if for any $(a_1, b_1, \ldots)$ *and* $(a_2, b_2, \ldots)$ *with* $\begin{bmatrix} & a^* \le a_1 \le a_2, & \underline{or}\ a_2 \le a_1 \le a^* \\ \underline{and} & b^* \le b_1 \le b_2, & \underline{or}\ b_2 \le b_1 \le b^* \\ \underline{and} & \ldots & \ldots \end{bmatrix}$ *, then* $f(a_1, b_1, \ldots) \le f(a_2, b_2, \ldots)$ *holds.*

Definition 22 Unimodality by Ben-Tal

A differentiable function $f : R^p \longrightarrow R^1$ *is said to be* **unimodal** *if every critical point is a global minimizer, e.g.* $\bigtriangledown f(\underline{x}) = \underline{0} \ \Rightarrow f(\underline{y}) \ge f(\underline{x})$ *for* $\forall\, \underline{y} \in D$. [10]

Suppose we define the unimodality by the condition that f has one stationary point, then this may not be a good definition when f is not differentiable, e.g. $f(x) = |x|$.

Example 3 Multiple Linear Regression

$$f(\underline{b}) = (\underline{y} - X\underline{b})^{'}(\underline{y} - X\underline{b}) \tag{3.14}$$

1. $f(\underline{b})$ *is convex with respect to* b *since* $\bigtriangledown^2 f(\underline{b}) = 2X^{'}X$ *is positive semi-definite.*

2. $f(\underline{b})$ *is q-convex since* $f(\underline{b}_2) \le f(\underline{b}_1)$ *implies* $(\underline{b}_2 - \underline{b}_1)^T \bigtriangledown f(\underline{b}_1) \le 0$ *using proposition 3. Note* $\bigtriangledown f(\underline{b}_1) = -2X^{'}\underline{y} + 2X^{'}X\underline{b}_1$ *and* $(\underline{b}_2 - \underline{b}_1)^T \bigtriangledown f(\underline{b}_1) \le 0$.

Example 4 Multiple Linear Regression

Show the convexity of a parametric function of squared error loss in the multiple linear regression.

$$f(\underline{b}) = \sum_{i=1}^{n} (y_i - \sum b_i x_{ij})^2 = (\underline{y} - X\underline{b})^{'}(\underline{y} - X\underline{b}) \tag{3.15}$$

Since $\partial f / \partial \underline{b} = -2X'\underline{y} + 2X'X\underline{b}$ *and*

$$\frac{\partial^2 f}{\partial \underline{b}^2} = 2X'X \tag{3.16}$$

is positive semidefinite, the function f *is convex with respect to* $\underline{b}$ *no matter what value of* X *is given.*

[10] We have presented 1)definition of q-convexity on page 39, 2)definition of unimodality(k-convexity) on page 44, and 3)definition of unimodality by Ben-Tal here.

3.3 Convex Envelope

As we have seen in the earlier sections, it is not easy to devise a convex criterion. One of the more extensive recommendations is to find a convex approximation that can be theorically applied to any function $f : R^m \longrightarrow R^1$.

In this section, we introduce the concept of an envelope and show how it can be derived. We note that the goal of this section is to show that the notion of the convex envelope can be used for the optimization of functions in statistics. It will be noted that *the* convex envelope function has the same minimum point as the original function. Therefore if we are able to find the convex envelope, then we use the IGS algorithm to find the optimization point, because the envelope has the same minimum as the original function. The literature [25] [12] [13] shows that, in many cases, finding the convex envelope of a given function is more difficult than the direct optimization of a given function. We start with the mathematical setup for the convex envelope.

Consider a general optimization problem

$$\underline{Min}\ f(x) \tag{3.17}$$

where $x \in \mathcal{D} \subset R^p$. It is known that the original problem in 3.17 is equivalent [11] to the problem with level set[12]

$$\underline{min}\ \{t\ ;\ (x,t) \in \tilde{G}\} \tag{3.19}$$

Equivalently, $\underline{min}\,\varphi_k(D_k)$ is called a **relaxation** for $\varphi_k(x) \leq \varphi_{k+1}(x) \leq f(x), \forall x \in D$ i.e. $\varphi_k(x)$ is a nondecreasing sequence of underestimators of $f(x)$. Usually the relaxed problem can be solved by standard linear or convex programming techniques. Also note that $D_k = D$ possibly. The

[11] Apply an outer approximation after constructing a sequence of nested sets $\tilde{G_1} \supset \tilde{G_2} \supset \ldots \supset \tilde{G_k} \supset \ldots \supset \tilde{G}$ such that each problem

$$\underline{min}\ \{t\ ;\ (x,t) \in \tilde{G_k}\} \tag{3.18}$$

can be solved by available algorithms and note that its optimal solution approaches the optimal solution of the original as k increases.

Consider $\tilde{G_k} = G_k \cap (D_k \times R)$ where $G_k \supset G_{k+1} \supset G$ and $D_k \supset D_{k+1} \supset D$ and G_k is the **epigraph** of some function φ_k defined on some sets $S_k \supset D_k$.

[12] The **level set** of $f(x)$ on R^m is defined by $\tilde{G} = \{(x,t)\ ;\ x \in \mathcal{D},\ f(x) \leq t\} \subset R^{p+1}$.

successive underestimation method consists in constructing a sequence of underestimators $\varphi_k(x)$ such that the sequence $\{x^k \mid x^k \in \{\underline{min}\ \varphi_k(x) : x \in D\}\}$ converges to an optimal solution.

Definition 23 *A* **convex subfunctional** *of f on D is a convex function that never exceeds f on D.*

Definition 24 *A convex subfunctional φ is said to be the* **convex envelope** *of f if no other convex subfunctional exceeds φ at any $x \in D$.*

Therefore the convex envelope is the pointwise supremum of all convex subfunctionals of f. The convex envelope φ is the uniformly best[13] convex approximation of f on D from below. The convex envelope is uniquely determined, if it exists.

Theorem 7 *Let $f : R^p \longrightarrow R$ be continuous on convex and compact set $D \subset R^p$. Let φ be the convex envelope of f on D. Then the optimization problem $\underline{min}\,\varphi$ is equivalent to $\underline{min}\,f$.*

By the above theorem, many non-convex minimization problem can be replaced by a convex minimization problem if the convex envelope is available.

Lemma 2 *Let $\mathcal{D}$ be compact and convex and let $f : R^p \longrightarrow R$ be continuous on D.*
Then $\varphi : R^p \longrightarrow R$ is the convex envelope of f iff

$$\underline{epi}(f)^{14} = \underline{conv}(\underline{epi}(f)) \tag{3.20}$$

or equivalently $\varphi(x) = \underline{inf}\,\{t \mid (x,t) \in \underline{conv}(\underline{epi}(f))\}$.

Corollary 1 *The two functions φ and f coincide at the extreme points.*

Definition 25

$$f^*(t) = \underline{max}\,\{xt - f(x)\} \tag{3.21}$$

is called the **conjugate** *function of f where the max is with respect to $x \in D$.*

---

[13] This means the biggest or closest

[14] **Notation** 1)$\underline{epi}\ (f) = \{(x,t) \in R^p \times R\ ;\ f(x) \leq t\} \in R^{p+1}$, 2)$\underline{conv}(A) = \underline{convexhullof}\ A$.

Note that the domain of f^* is $D(f^*) = \{t; \underline{sup}\{xt - f(x)\} < \infty\}$ where the supremum is with respect to $x \in D$.

Theorem 8

$$f^{**}(x) = \varphi(x) \tag{3.22}$$

Proof can be found in [13]. This is another way to find the convex envelope.

3.4 On finding the Convex Envelope of a composite

Many of the criteria in a statistical estimation problem involves the composite of the fitting and the loss function, e.g. $f(b) = \sum (y_i - x_i b)^2$. Our interests are turned towards finding the convex envelope of a composite. Finding the convex envelope for a general function f is *not* our main goal in this section. We are mainly interested in finding the envelope of a composite in the case where each of the two envelopes are easily found. Also we assume that it is technically *possible* to provide the convex envelope for a *univariate* function.

Consider [15] $G : R^1 \longrightarrow R^1$ and $f : R^p \longrightarrow R^1$ and find the envelope of the composite function

$$(G \circ f)(\underline{x}) \tag{3.23}$$

Assume that there exist an *available*[16] convex envelope function $L(\underline{x}) : R^p \longrightarrow R^1$ and a concave upper-estimating function $U(\underline{x}) : R^p \longrightarrow R^1$, where $L(\underline{x}) \leq f(\underline{x}) \leq U(\underline{x})$ and $a \leq f(\underline{x}) \leq b$ for fixed $a, b \in R^1$. The convex envelope of $G : R^1 \longrightarrow R^1$ in the interval [17] $[a, b]$ is denoted by $E(\cdot)$[18] $: R^1 \longrightarrow R^1$. Now compute a point z_{min} at which each function achieves its minimum on its interval domain where $G(z_{min}) = \underline{inf}\ G(z)$ and the infimum is with respect to $a \leq z \leq b$.

[15] Many criteria usually are involved in a sum and are a function of n. But here in one dimension, we just need show how to derive the convex composite.

[16] It means that we can find $L(\underline{x})$ and $U(\underline{x})$ easily.

[17] since $a \leq f(\underline{x}) \leq b$

[18] Note that we do not need the bounds like in the condition of $f(x)$.

Finding the Envelope of a composite

The lower envelope [25] of $G(f(x))$ for $x \in \mathcal{D} \cap \{x | a \leq G \leq b\}$ is

$$E[\underline{mid}\{L(x), U(x), z_{min}\}] \leq G(f(x)) \tag{3.24}$$

Furthermore if 1)G is continuously differentiable and 2)L and U are also continuously differentiable, then the envelope of $G(f(x))$ is also continuously differentiable.

Sum of two functions

$$\sum_{i=1}^{n} G(f_i(x)) \geq \sum_{i=1}^{n} E_G[\underline{mid}\ \{L_i(x), U_i(x), z_{min}^G\}] \tag{3.25}$$

where E_G is an convex underestimating function of G and $L_i(x) \leq f_i(x) \leq U_i(x)$ are both estimating functions and z_{min}^G is the minimum point of $G : R^1 \longrightarrow R$. [19]

In order to use the method of computing underestimating programs, it must be possible to compute the convex envelope of functions of a single variable. We can only provide a *lower bound* of the sum of two functions. [20]

Example 5 (Application on finding the envelope of a composite) [25]
Consider the composite $(G \circ f)(\underline{x})$ *where* $G(x) = x^2$ *and* $f(x_1, x_2) = \exp(x_1) + x_2$ *on* $\mathcal{D} = \{\,(x_1, x_2)\,|\,0 \leq x_1 \leq 1,\ 20 \leq x_2 \leq 30\,\}$, *where* $a = -1,\ \ b = 2$.

In $[-1, 2]$ *, the convex envelope of* G *is* x^2, *e.g.* $E(x) = x^2$ *and* $x_{min} = 0$. *The convex underestimating function for* g *is* $L(x) = \exp(x_1) + x_2$ *in* $\mathcal{D}$ *and its concave overestimate is* $U(x) = 1 + (1.718)x_1 + x_2$.

Thus from the result in 3.24, a convex underestimating function for $G(f(x)) = (\exp(x_1) + x_2)^2$ *in* $\{x | -1 \leq \exp(x_1) + x_2 \leq 2\}$ *is*

$$[\underline{mid}\ \{1.718x_1 + 1 + x_2, \exp(x_1) + x_2, 0\}]^2 \tag{3.26}$$

[19] It is not true that the envelope of sum is the sum of envelope. It can give just a lower bound.
[20] This implies that the above result is *not* quite useful since many criteria include a summation.

3.5 On some selected Criteria: Least Square, BOS

In this section we review a widely-used criterion, Least Square, and introduce a selected criterion, BOS, to discuss the application to our IGS procedure. The new criterion BOS with the Normal distribution is found to be applicable to the IGS procedure since it exhibits almost unimodality.

Simple examples [22] of a unimodal function are as follows: $1) f(a) = \sum (x_i - a)^2$ is convex, and $a = \overline{x}$ is the unique optimum point; $2) f(a) = \sum |x_i - a|$ if n is odd, then the optimum point is unique; 3) $f(a) = \sum \sqrt{|x_i - a|}$. If $n = 2$, $a = x_1$ and $a = x_2$ are the two minimizers. These raise the question of the form $f(a) = \sum \rho(x_i - a)$. These examples can be used for the application of the statistical criteria since IGS has a guaranteed convergence for the unimodal function.

Ordinary Least Square

Consider the Gauss-Markov Model $\underline{y} = X\underline{b} + \underline{e}$, where $\underline{e}$ has mean vector $\underline{0}_{p\times 1}$ and covariance matrix $I_{p\times p}$. By the Gauss-Markov theorem, $\hat{\underline{b}}_{LSE} = (X'X)^{-1}X'\underline{y}$ is LSE of $\underline{b}$. This is the most widely-used case of the squared error and is also a simple example of the convex criteria. In this case, most optimization methods, including QR decomposition [39], work as well as IGS, as shown in chapter 1.

Weighted Least Square

Furthermore if $\underline{e}$ has covariance matrix $W_{p\times p}$, then $f(\underline{b}) = (\underline{y} - X\underline{b})^T W^{-1}(\underline{y} - X\underline{b})$ where $W_{p\times p}$ is positive definite.

- If $W = I$, then f is convex in b and equivalent to the previous OLS.
- If $W \neq I$, then

$$\frac{\partial^2}{\partial b^2} f = X^T (W^{-1})^T X \tag{3.27}$$

So if W is positive definite, then $\partial^2 f/\partial b^2$ is positive definite by the Cholesky decomposition [39], which implies the convexity of f. Therefore we can choose any kind of positive definite W.

Best Order Statistic

Suppose a random variable X has a distribution function F, then we know that the transformed random variable $F(X)$ has a distribution $F(X) \sim \underline{U}(0,1)$. In order to introduce the BOS we need to show[21] $EFX_{[i]} = i/(n+1)$, and note that $[F(X)]_{[i]} = F(X_{[i]})$ since the order statistic is invariant under a monotone mapping (transformation), and $F(X_{[i]})$ can be considered as an approximation to the i$\underline{th}$ order statistic from the *Uniform* distribution. Also we can find the distribution of $F(X_{[i]})$ for $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$. The *exact* distribution of the k$\underline{th}$ order statistics $X_{(k)}$ is easily derived as a cumulative distribution function.

$$F_{X_{(k)}}(x) = P\{X_{(k)} \leq x\} = \sum_{i=k}^{n} F(x)^i (1 - F(x))^{n-i} \tag{3.30}$$

The motivation behind proposing a new criterion (BOS) comes from the fact that

$$E\, F\, X_{[i]} = \frac{i}{n+1} \tag{3.31}$$

---

[21] Show that the expected value of the F of the $i\underline{\text{th}}$ order statistics among n samples $X_1, \ldots, X_n$ is

$$\underline{E}FX_{[i]} = \frac{i}{n+1} \tag{3.28}$$

From the definition $F_X(x) = \underline{Pr}\{X \leq x\}$, we have $F_X(x_{[i]}) = \int_{-\infty}^{x_{[i]}} f_X(x)\, dx$. And note that $f_{X_{[i]}}(x) = (n, i-1, n-i) F_X(x)^{i-1} f_X(x)[1 - F_X(x)]^{n-i}$ where $(n, i-1, n-i) = n!/(i-1)!(n-i)!$. Therefore

$$\begin{aligned}
\underline{E}FX_{[i]} &= \int_{-\infty}^{\infty} F(x) \underline{Pr}\{X_{[i]} = x\}\, dx \\
&= (n, i-1, n-i) \int_{-\infty}^{\infty} F(x)^i [1 - F(x)]^{n-i} f(x)\, dx \\
&= (n, i-1, n-i) \int_0^1 t^i (1-t)^{n-i}\, dt \\
&= (n, i-1, n-i)\, B(i+1, n-i+1) \\
&= \frac{i}{n+1}
\end{aligned} \tag{3.29}$$

substituted by $t = F(x)$.

For an ideal distribution

$$(FX_{[i]} - \frac{i}{n+1})^2 = 0 \tag{3.32}$$

But in most cases this will not be zero.

The BOS estimate $\hat{\theta}_{BOS}$ of θ, using the new criterion BOS, is minimizing

$$f(\theta) = \sum_{i=1}^{n} (F[X_{[i]}|\theta] - \frac{i}{n+1})^2 \tag{3.33}$$

Since the object function $f(\theta)$ displays almost unimodality in Figure 4.3, we can use the IGS procedure to find the approximate estimate.

Chapter 4
Application Study

In this chapter we will use the proposed IGS program to show that it works for unimodal and for almost unimodal functions. The IGS program will find the true (reasonable) solution given reasonable bounds on the region of interest.

4.1 Main Code

The main code IGS, as shown in appendices 5.7, is a subroutine used to find a minimum point of a unimodal function `ff`: $R^m \longrightarrow R^1$ given `xmax`, `xmin` $\in R^m$ representing an initial rectangle, and it returns a vector $\in R^m$ of the optimum point for a unimodal function or an estimated optimum point for an almost unimodal function. Note that if the function `ff` is almost unimodal, then it provides a rough estimate.

Remarks on the main program `ft1`

- Give the initial rectangular region that is represented by the two points `xmin`, `xmax` $\in R^m$.
- Before running `ft1`, define an outside function `ff` that will be used in `ft1`. Remember that `ff` is not a local variable. Obviously the dimension of parameters `xmin`, `xmax` should be equal to the dimension of the domain of the function `ff`. The function `ff` has a form like `f(z)=z[1] ^ 2 + z[2] ^ 2` where `z=(z[1],z[2],...)` is a vector of length `p`.
- Adjust the value of `maxiter` so that, in case of a divergence condition, the loop can be ended.
- Adjust the α value. The possible value of α can be 1)$\alpha = 1/3$ (equi-dist grid) or 2)$\alpha = 0.618$ for better efficiency (as shown in chapter 2).

- Given the dimension `p` of the parameter, the index matrix `mt` will be a matrix [1] of size p by 2^p such that all of 2^p combinations of 0 and 1 occur(s). From the initial setting of `mt`$= (0, 1)$, the first loop gives an index matrix for the use of the main loop (the `while` loop). This matrix `mt` is designed to locate the index for the sub-region in which the value of the function gives the minimum.

4.2 Multiple Linear Regression

Result of `ft2`

For the MLR Model, $\hat{\underline{y}} = X\hat{\underline{b}}$, where $X_{n\times p}$, $\underline{b}_{p\times 1}$, $\underline{y}_{n\times 1}$ and $n = 5$, $p = 3$, find the LSE of $\underline{b}$ using our main code `ft1` and compare it with the result using the built-in `lsfit` subroutine in *Splus 3.1* to check that the main code `ft1` is working. The reader should note that the `lsfit` function employs the QR decomposition. Due to the convexity (even quadratic), the two methods give virtually the same answer, as is predicted mathematically. The computational error is due to the limit of the storage in digital computers. Therefore we conclude that, under the assumption of a good condition number and with a convex criterion, the two methods, using the QR decomposition and the IGS algorithm, give the same answer.

```
xmin=-10 -10 -10
xmax=10 10 10
grid search  1.12636086385887 -0.627212043673474 0.738778264942685
ls fit  1.12776741365863 -0.626757419680911 0.737262775300298
```

[1] For example, if `p`$= 2$, then the index matrix `mt` is

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}_{2\times 2^2} \tag{4.1}$$

and if `p`$= 3$, then the index matrix `mt` is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}_{3\times 2^3} \tag{4.2}$$

Result of `f3`

In the multiple linear regression case, we consider the possible difficulties for the inverse $(X^TX)^{-1}$ in case of the *rank deficiency* and present a simple example where the IGS procedure is definitely needed. The difficulty occurs when the rank of the design matrix X is *almost* 1. It is possible to prove that the rank of the matrix X^TX is the same as the rank of the matrix X.

Consider the case of the example in section 1.1 in which the usual method of least square fit $(X^TX)^TX^T\underline{y}$ did not work but the IGS algorithm did solve the least square equation. We consider the condition number and the inverse as shown in appendices 5.3.

4.3 MLE in the Cauchy distribution

Finding MLE of the location parameter in the Cauchy distribution using the codes `p1` and `p2`

This section intends to show the validity of what was presented in section 1.2.

Given the likelihood function to maximize,

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{1 + (x_i - \theta)^2} \tag{4.3}$$

we can minimize, equivalently, the unimodal function $f(\theta) = -\log L(\theta)$, which is the object function in this section. As noted in chapter 1, the graph of $f(\theta)$ shows, *amazingly*, unimodality. But *unfortunately* no algebraic method can be used to verify this unimodality. Even though it is not proved or verified that f is *always* unimodal, we assume in this simulation study it is unimodal.

$$l(\theta) \equiv \sum_{i=1}^{n} \log[1 + (x_i - \theta)^2] \tag{4.4}$$

To use the Newton method [2] introduced in formula 1.13, we need to find a zero of the equation, $\partial l/\partial\theta = 0$, in formula 4.5. The answer using the Newton method will be compared with the answer

[2] Also in this case we can use the **Bisection Method** since the shape of f_1 shows negative on the left side of the zero and positive on the right side of the zero, as shown in Figure 4.2.

using the IGS method.

$$f(\theta) \equiv \frac{\partial l}{\partial \theta} = \sum_{i=1}^{n} \frac{-2(x_i - \theta)}{1 + (x_i - \theta)^2} = 0 \tag{4.5}$$

Its derivative is

$$f_2(\theta) \equiv \frac{\partial f}{\partial \theta} = \sum_{i=1}^{n} \frac{2[1 + 3(x_i - \theta)^2]}{[1 + (x_i - \theta)^2]^2} \tag{4.6}$$

Remarks on program `p1`

- Set the parameter $\theta = 2$, for example, and generate a random sample from the Cauchy distribution using the known formula:

$$\tan[\pi(X - \frac{1}{2})] + \theta \ \sim \ Cauchy(\theta) \tag{4.7}$$

  where X has the Uniform(0,1) distribution. The value of $\theta = 2$ that was pre-set is irrelevant to this particular program since it is location equivariant.

- In Figure 4.1, we show that the object function f is unimodal provided with a range of $-10 \leq x \leq 10$.

- `tt`= $(-49/5, -48/5, \ldots, 50/5)$

- `z3`= $(f(-49/5), f(-48/5), \ldots, f(50/5))$

Result of program `p1`

- In Figure 4.1, it shows the 2-dimensional unimodal graph of f with respect to `tt` that has minimum value around 2, as it should be from the presetting of $\theta = 2$.

Remarks on program `p2`

- The program is designed so that, for many cases of random samples ($n = 1, 2, \ldots, 20$), we can see the shape of f and f_2 defined in formulas 4.5 and 4.6.

- The shapes of f_1 and f_2 with range $(\theta - 5, \theta + 5)$ are shown.

- The Newton method and the Secant method for finding the zeros of the equation $f(\theta) = 0$ are used.
- Note that for small integer n, the graph of f may not show unimodality. As n increases, the plot of $L(\theta)$ tends toward a unimodality and a smoother shape.

Result of program `p2`

- See Figure 4.2 and appendices 5.5.
- Obviously the Newton method of finding the zero of the equation $f_1 = 0$ does show divergence.

Remarks on program `p21`

- This program is similar to the previous program `p2`. Program `p21` is designed to show the result with the replication after choosing θ and n.

Conclusion

As we can see in appendices 5.5, neither the Newton method nor the Secant method is recommended. Note that the Newton method is not reliable since it *sometimes* diverges to the choice of the initial value. Also it is easy to find other examples in which the Newton method does not converge. But the IGS method can be used whenever the graph shows unimodality.

4.4 BOS in the Normal distribution (example of almost unimodality)

BOS to estimate the mean of the 1-par Normal distribution using `ft4` and `ft4a`

This section intends to show the almost unimodality using the BOS of one parameter `mu` from the Normal distribution `N(mu,sigma)` as shown in Figure 4.3. Set a parameter `mu`=1 from N(`mu`,2) with *fixed* variance 2 and generate the random samples. This is an example in which the IGS algorithm is definitely needed since the criterion function demonstrates almost unimodality. We

fix the variance and generate the random samples with mean `mu=` 1. We then investigate the 1-dimensional surface of the object function of `mu` (μ) and want an estimate $\hat{\mu}$ even though we know the sample was generated from the distribution with $\mu = 1$. Obviously $\hat{\mu}$ should have a value close to 1. See Figure 4.3. [3]

Remarks on program `ft4` and `ft4a`

- Note that `xmin` and `xmax` were randomly selected from the Uniform distribution to provide the two fixed parameters $-1 \leq$ `xmin` $\leq 1 \leq$ `xmax` ≤ 3.
- The built-in subroutine `rnorm(3,mean=0,sd=1)` in *Splus 3.1* gives 3 random samples of N(0,1).
- The built-in subroutine `pnorm(q,0,1)`$=F(q) \equiv \int_{-\infty}^{q} f(x)dx$ where $-\infty < q < \infty$ and $f(x)$ is the density of $N(0,1)$.
- The built-in *Splus 3.1* subroutine `qnorm(x)`$=\{y | \int_{-\infty}^{y} f(x)dx = x\}$
- Note that the function `ff` represent the BOS criterion:

$$f(\theta) = \sum_{i=1}^{n} (F[X_{[i]}|\theta] - \frac{i}{n+1})^2 \tag{4.8}$$

- $l = [ff(1/50), ff(2/50), \ldots, ff(100/50)]^T$
- $ll = [ff(-49/5), ff(-48/5), \ldots, ff(50/5)]^T$
- `z1` is the estimated minimum point of `ff` using the IGS algorithm.
- `z2` is the estimated minimum point of `ff` using the built-in subroutine `nlmin` in *Splus 3.1*.

[3] Why 20 observations make more than 20 bumps? And how about changing n for more smoothing? They are left to the Further Study.

Result

- In Figure 4.3, the graph with x range $-10 \leq x \leq 10$ is given. Almost unimodality is obvious from the figure(s).[4] With a smaller x range of $0 \leq x \leq 2$, the graph of `ff` is more unstable. Even though the graph seems quite erratic(not exactly unimodal), we shall still use the IGS to find the estimate that gives the minimum value for the BOS criterion in formula 4.8.

- From the output shown in appendices 5.7, the subroutine `nlmin` finds the local minimum which is *highly dependent* on the initial point of the q-Newton algorithm. Here in this project we set the range of `xmin` and `xmax` for an initial rectangle. And the subroutine `nlmin` shows quite biased results depending on the selection of the initial point in the q-Newton method. On the other hand, IGS gives a stable answer around 1 as expected.

[4] Of course it is almost unimodal with minimum around $x = 1$ as it is supposed to be.

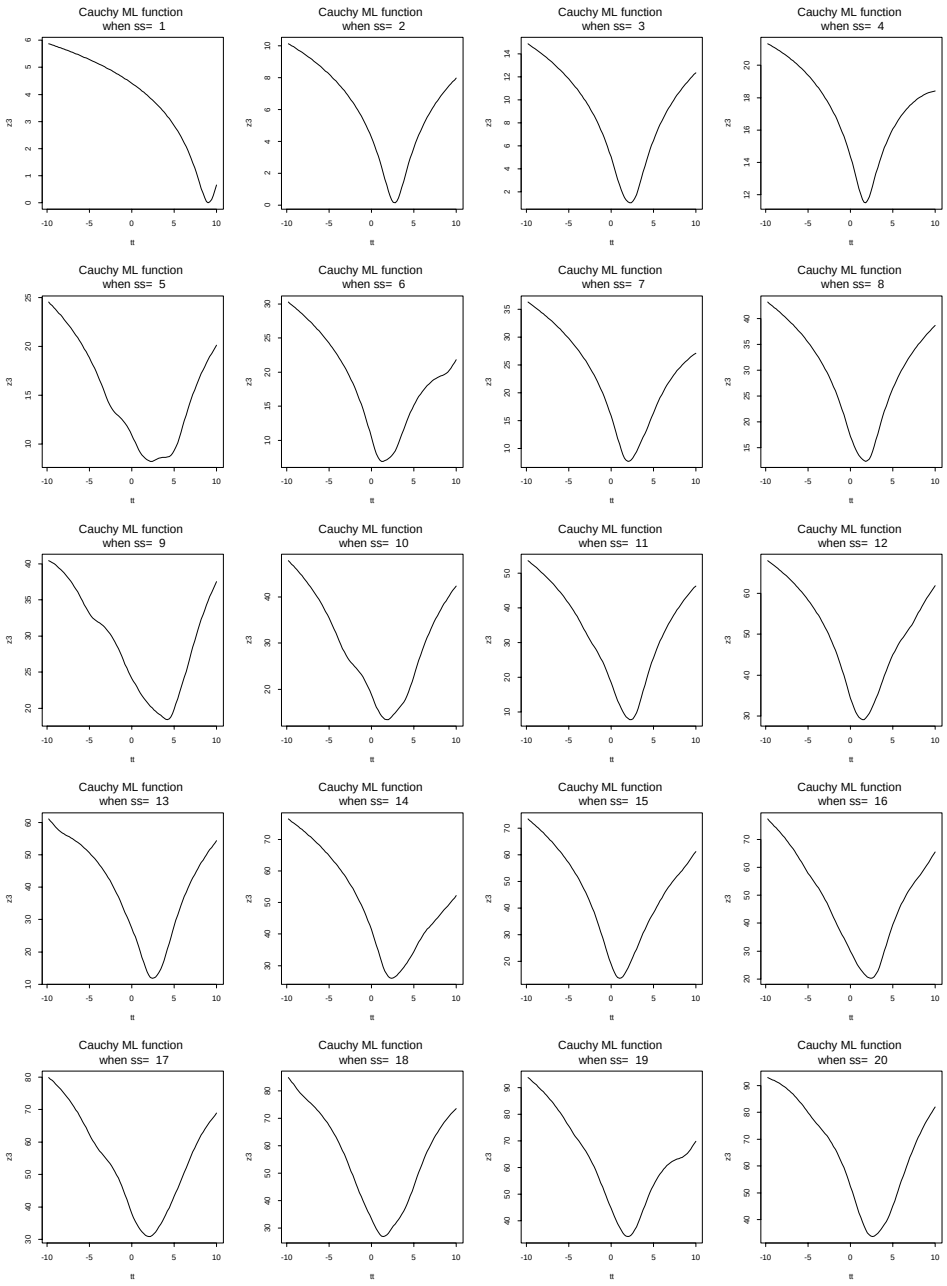

Figure 4.1: Cauchy MLE function (Plot of $\sum_{i=1}^{n} \log[1 + (x_i - \theta)^2]$, $\theta = 2$ fixed, $X_i \sim Cauchy(\theta)$) with different sample size `ss`

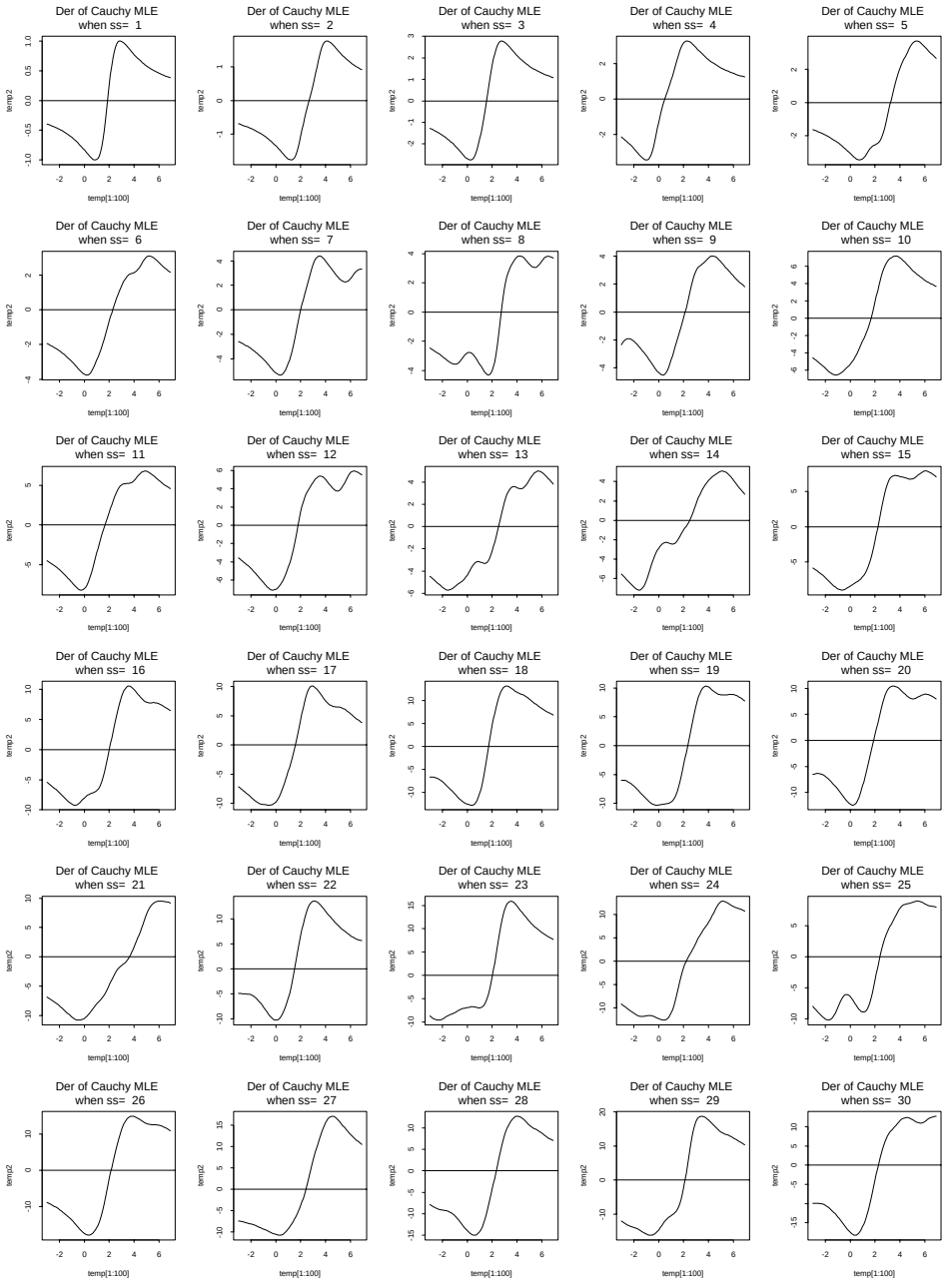

Figure 4.2: Derivative of Cauchy MLE function, with different sample size

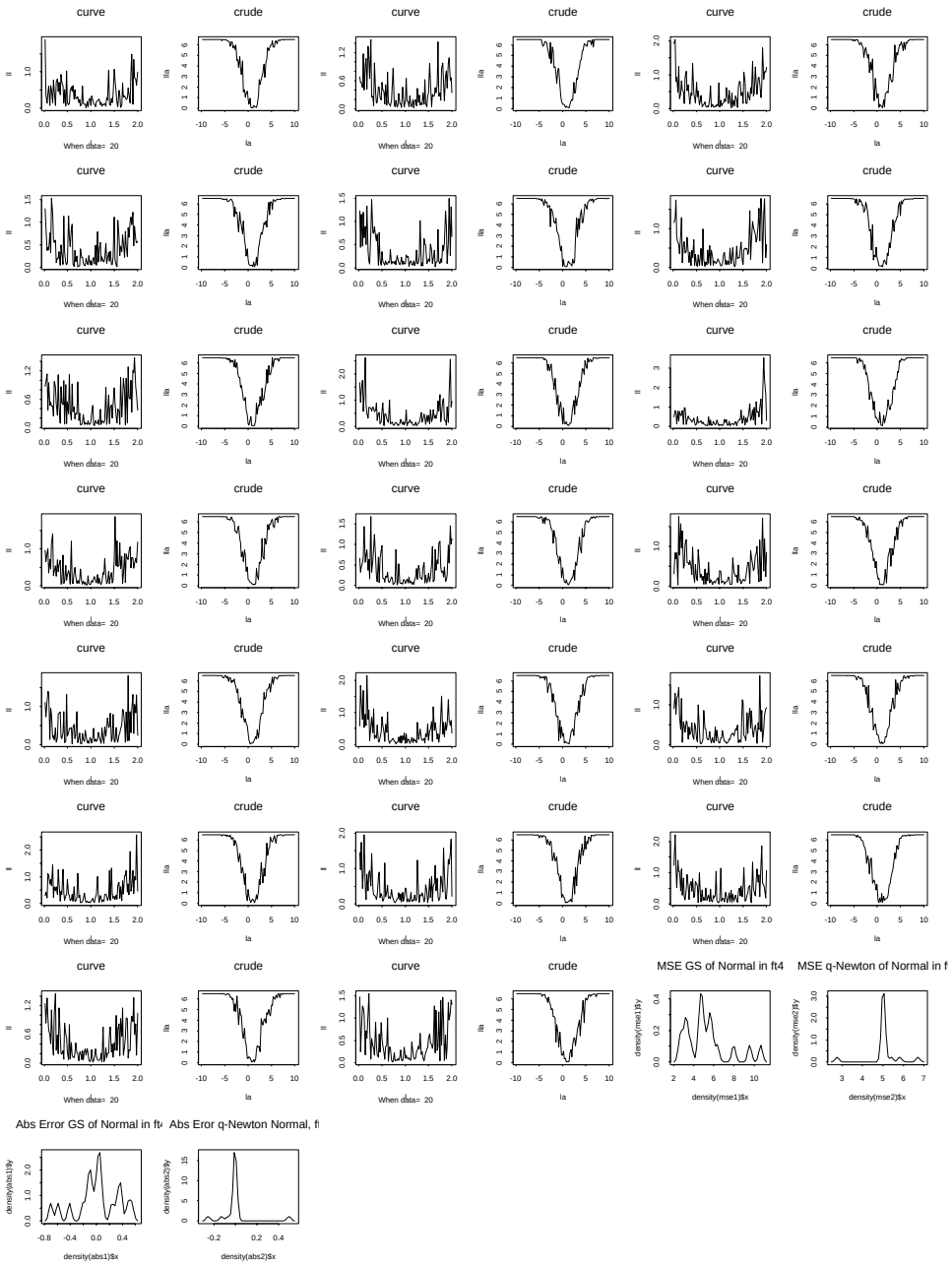

Figure 4.3: An example of almost unimodality from BOS of the one parameter Normal distribution: to see the almost unimodality

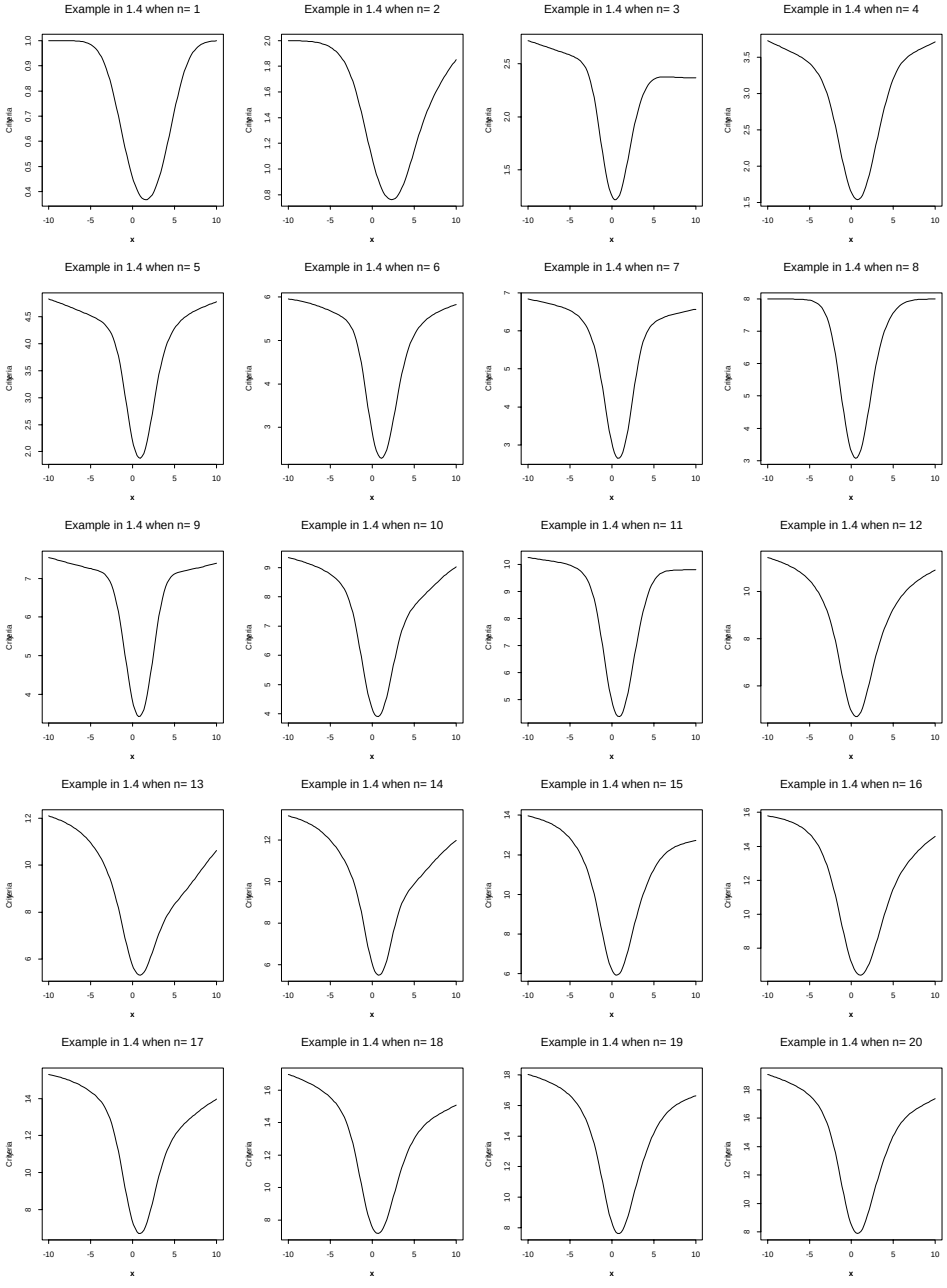

Figure 4.4: A simple example of unimodal criteria presented in section 1.4

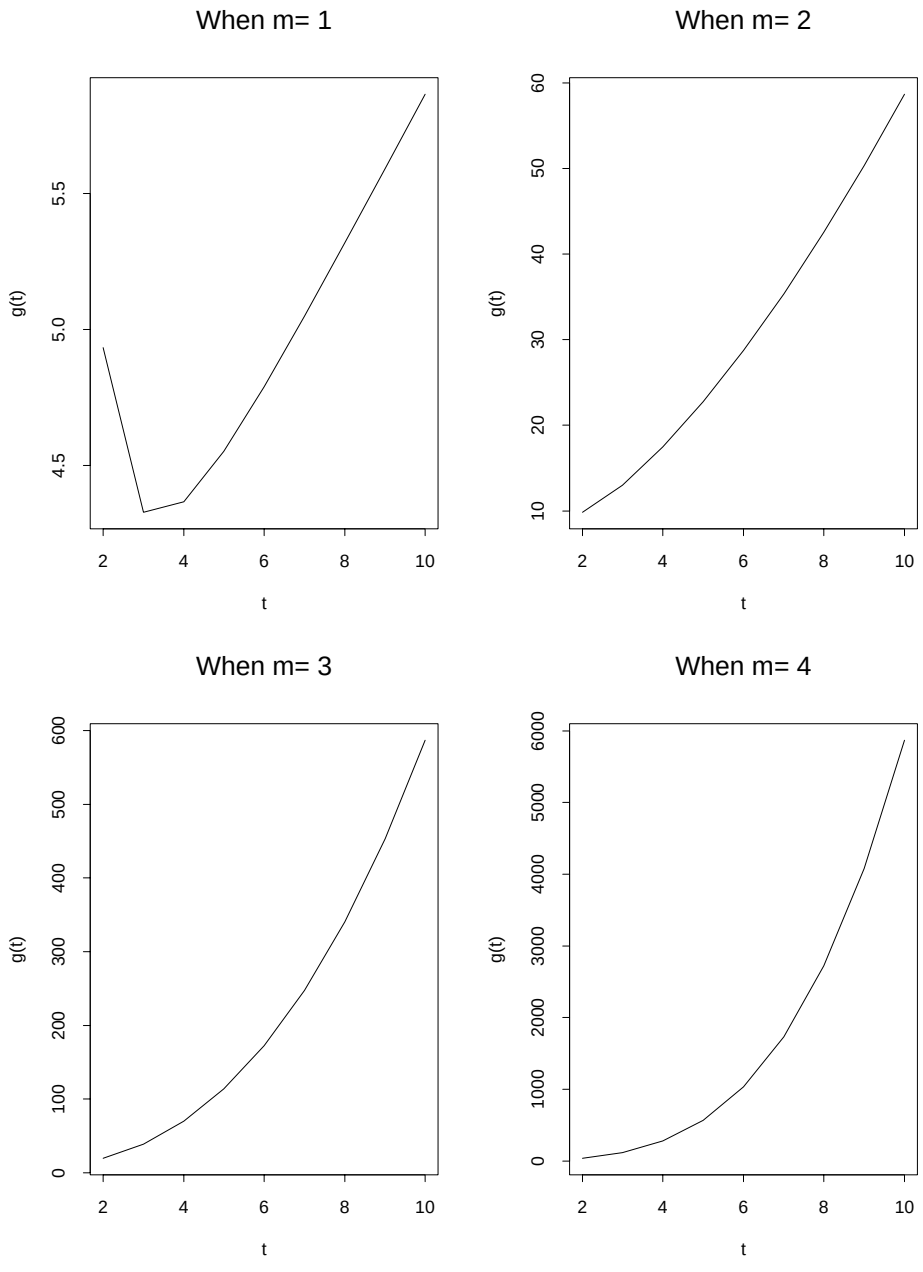

Figure 4.5: Minimization of $g(t)$

Chapter 5

Appendices

5.1 The ff function

```
 What?>>ff
function(b)
{
#Note n=5, p=3, \vec{b} is 3 by1 vector.
# x <- matrix(runif(15), nrow = 5, ncol = 3)
# y <- matrix(runif(5), nrow = 5, ncol = 1)
# tt <- as.matrix(rbind(b[1], b[2], b[3]))
#tt <- b
x <- matrix(c(1, 2, 1.0001, 2, 1.0002, 2), nrow = 3, ncol = 2, byrow = T)
y <- matrix(c(2, 3, 4), nrow = 3, ncol = 1)
return(t(y - x %*% matrix(b, nrow = 2, ncol = 1)) %*% (y - x %*%
matrix(b, nrow = 2, ncol = 1)))
}
```

5.2 Minimization of $g(t)$

```
 What?>>g
function(m, t)
{
return((t^m)/(log(t + 1) - log(2)))
}
```

```
 What?>>g(1,c(1:5))
[1]      Inf 4.932607 4.328085 4.365427 4.551196
 What?>>g(1,1:5)
[1]      Inf 4.932607 4.328085 4.365427 4.551196
 What?>>g(2,1:5)
[1]       Inf  9.865214 12.984255 17.461707 22.755981
 What?>>g(3,1:5)
[1]       Inf  19.73043  38.95277  69.84683 113.77990
```

5.3 Result of QR decomposition using *Splus*

```
 When eps=  0.1
x=     [,1] [,2]
[1,]  1.0    2
[2,]  1.1    2
[3,]  1.2    2
x'x=     [,1] [,2]
[1,] 3.65  6.6
[2,] 6.60 12.0

 The cond of x'x is 0.000981826938262349
 The inverse is      [,1]      [,2]
[1,]  50.0 -27.50000
[2,] -27.5  15.20833

 When eps=  0.01x=     [,1] [,2]
[1,] 1.00    2
[2,] 1.01    2
[3,] 1.02    2
x'x=       [,1]  [,2]
[1,] 3.0605  6.06
[2,] 6.0600 12.00

 The cond of x'x is 0.0000105813639327538
 The inverse is      [,1]      [,2]
[1,]  5000 -2525.000
[2,] -2525  1275.208
```

```
 When eps=  0.001x=      [,1] [,2]
[1,] 1.000    2
[2,] 1.001    2
[3,] 1.002    2
x'x=         [,1]   [,2]
[1,] 3.006005  6.006
[2,] 6.006000 12.000

 The cond of x'x is 1.06581336261621e-007
 The inverse is        [,1]      [,2]
[1,]  500000 -250250.0
[2,] -250250  125250.2

 When eps=  0.0001x=       [,1] [,2]
[1,] 1.0000    2
[2,] 1.0001    2
[3,] 1.0002    2
x'x=       [,1]    [,2]
[1,] 3.0006  6.0006
[2,] 6.0006 12.0000

 The cond of x'x is 1.06658132609369e-009
 The inverse is
Error in solve.qr(a): apparently singular matrix
Dumped
```

5.4 Program and result using *SAS*

```
data htwt; input x0 1-10 x1 11-12 y 13-14;
 cards;
1          2 2
1.001      2 3
1.002      2 4
; run;
proc print data=htwt; run;
proc reg data=htwt; model y=x0 x1; run;

data htwt; input x0 1-10 x1 11-12 y 13-14;
 cards;
1          2 2
1.0001     2 3
1.0002     2 4
; run;
proc print data=htwt; run;
proc reg data=htwt; model y=x0 x1; run;

data htwt; input x0 1-10 x1 11-12 y 13-14;
 cards;
1          2 2
1.00001    2 3
1.00002    2 4
; run;
proc print data=htwt; run;
proc reg data=htwt; model y=x0 x1; run;
```

```
data htwt; input x0 1-10 x1 11-12 y 13-14;
 cards;
1          2 2
1.000001   2 3
1.000002   2 4
; run;
proc print data=htwt; run;
proc reg data=htwt; model y=x0 x1; run;
```


OBS	X0	X1	Y
1	1.000	2	2
2	1.001	2	3
3	1.002	2	4

Model: MODEL1
Dependent Variable: Y

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	1	2.00000	2.00000	7773936923.6	0.0020

Error	1	2.572699E-10	2.572699E-10
C Total	2	2.00000	

Root MSE	0.00002	R-square	1.0000
Dep Mean	3.00000	Adj R-sq	1.0000
C.V.	0.00053		

NOTE: Model is not full rank. Least-squares solutions for the parameters are not unique. Some statistics will be misleading. A reported DF of 0 or B means that the estimate is biased.
The following parameters have been set to 0, since the variables are a linear combination of other variables as shown.

X1 = +2.0000 * INTERCEP

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	B	-998.000000	0.01135308	-87905.657	0.0001
X0	1	1000.000000	0.01134173	88169.932	0.0001
X1	0	0	.	.	.

OBS	X0	X1	Y
1	1.0000	2	2
2	1.0001	2	3
3	1.0002	2	4

Model: MODEL1

Dependent Variable: Y

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	1	2.00000	2.00000	19451064.123	0.0020
Error	1	1.0282213E-7	1.0282213E-7		
C Total	2	2.00000			

Root MSE	0.00032	R-square	1.0000
Dep Mean	3.00000	Adj R-sq	1.0000
C.V.	0.01069		

NOTE: Model is not full rank. Least-squares solutions for the parameters are not unique. Some statistics will be misleading. A reported DF of 0 or B means that the estimate is biased.
The following parameters have been set to 0, since the variables are a linear combination of other variables as shown.

X1 = +2.0000 * INTERCEP

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	B	-9997.999486	2.26762754	-4409.013	0.0001
X0	1	9999.999486	2.26740080	4410.336	0.0001
X1	0	0	.	.	.

OBS	X0	X1	Y
1	1.00000	2	2
2	1.00001	2	3
3	1.00002	2	4

Model: MODEL1

Dependent Variable: Y

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	0	0.00000	.	.	.
Error	2	2.00000	1.00000		
C Total	2	2.00000			

Root MSE	1.00000	R-square	0.0000
Dep Mean	3.00000	Adj R-sq	0.0000
C.V.	33.33333		

NOTE: Model is not full rank. Least-squares solutions for the parameters are not unique. Some statistics will be misleading. A reported DF of 0 or B means that the estimate is biased.
The following parameters have been set to 0, since the variables are a linear combination of other variables as shown.

X0 = +1.0000 * INTERCEP
X1 = +2.0000 * INTERCEP

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	B	3.000000	0.57735027	5.196	0.0351
X0	0	0	.	.	.
X1	0	0	.	.	.

OBS	X0	X1	Y
1	1.00000	2	2
2	1.00000	2	3
3	1.00000	2	4

Model: MODEL1

Dependent Variable: Y

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	0	0.00000	.	.	.
Error	2	2.00000	1.00000		

C Total 2 2.00000

Root MSE	1.00000	R-square	0.0000
Dep Mean	3.00000	Adj R-sq	0.0000
C.V.	33.33333		

NOTE: Model is not full rank. Least-squares solutions for the parameters are not unique. Some statistics will be misleading. A reported DF of 0 or B means that the estimate is biased.
The following parameters have been set to 0, since the variables are a linear combination of other variables as shown.

X0 = +1.0000 * INTERCEP
X1 = +2.0000 * INTERCEP

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	B	3.000000	0.57735027	5.196	0.0351
X0	0	0	.	.	.
X1	0	0	.	.	.

5.5 Result of p2 (MLE in the Cauchy distribution)

```
What?>>p2()

n=  1  the simulated rv are [1] 3.174869

Mean of Simulated was  3.17486910220851
After  1  iter, Newton is  3.17486910220851
After  2  iter, Secant is  3.17486910220851

n=  2  the simulated rv are [1] 1.650393 1.388411

Mean of Simulated was  1.51940173127223
After  1  iter, Newton is  1.51940173127223
After  2  iter, Secant is  1.51940173127223

n=  3  the simulated rv are [1] -0.8958592  6.6350467  2.2761273

Mean of Simulated was  2.67177157602522
After  15  iter, Newton is  2.1974039859748
After  6  iter, Secant is  2.19740393157568

n=  4  the simulated rv are [1]  3.070262  4.365707 55.482076  2.034943

Mean of Simulated was  16.2382467708589
After  41  iter, Newton is  3.07372825101987
After  8  iter, Secant is  42.4578121033605
```

```
 n=  5  the simulated rv are [1] 2.6113811 1.2483863 0.6923784 3.9299325 2.46855
85

 Mean of Simulated was  2.19012736837757
 After  39  iter, Newton is  2.26002053625478
 After  5  iter, Secant is  2.26002034768051

 n=  6  the simulated rv are [1]   1.653661   3.003110  20.040370   4.245578   2
.657971  -2.901410

 Mean of Simulated was  4.78321345329752
 After  25  iter, Newton is  2.74804416898015
 After  735  iter, Secant is  -1.54290815596155e+154

 n=  7  the simulated rv are [1] 2.562862 1.485409 1.491514 2.132490 3.243093 2.
464808 3.101601

 Mean of Simulated was  2.35453967629533
 After  24  iter, Newton is  2.38115816041208
 After  3  iter, Secant is  2.38115802717894

 n=  8  the simulated rv are [1]   3.7782344 -63.6883914   1.8148006   0.9789462
  -0.2013178   2.3818271
[7]   1.0139688   0.9857537

 Mean of Simulated was  -6.6170222927845
 After  28  iter, Newton is  1.27371216733664
 After  11  iter, Secant is  -55.6449383572386
```

```
 n=  9  the simulated rv are [1]  -1.06620616  -0.01912366  23.02628910   1.3936
9147  10.04470602
[6]  -1.49536939   3.93615048   3.00539635   2.34686784

 Mean of Simulated was  4.57471133888298
 After  94  iter, Newton is  1.97920351515055
 After  7  iter, Secant is  21.3282908340751

 n=  10  the simulated rv are  [1] 5.9538103 2.5187891 1.4226256 4.5624658 0.807
9517 3.1555267 2.4670325
 [8] 3.2653687 1.6713703 2.3004808

 Mean of Simulated was  2.81254214516313
 After  23  iter, Newton is  2.48931242766442
 After  5  iter, Secant is  2.48931229336596

 n=  11  the simulated rv are  [1]  2.89414320  2.39514899  5.22627169  2.554597
37  1.99312042  2.45311062
 [7]  0.75693806  7.38839188 -0.04449007  1.43160548  9.43514525

 Mean of Simulated was  3.31672571813768
 After  24  iter, Newton is  2.2892883334291
 After  737  iter, Secant is  -1.86681702438255e+154

 n=  12  the simulated rv are  [1]  1.6623009  2.2893162  5.2426546  0.9645044 -
1.5617721  2.1258625
 [7]  2.8924539  0.7284072  3.3716038  2.7159320  3.6245148  2.2454655
```

```
 Mean of Simulated was  2.19177030934135
 After  30  iter, Newton is  2.35833353599842
 After  4  iter, Secant is  2.35833366086779

 n=  13  the simulated rv are  [1] 5.0034277 2.1649090 3.8616224 3.1866334 0.580
6087 1.3828695 1.9936324
 [8] 2.6490341 3.4514877 2.1110871 3.3811219 4.1679605 0.3790329

 Mean of Simulated was  2.6394944006508
 After  55  iter, Newton is  2.72488243884937
 After  4  iter, Secant is  2.72488276169981

 n=  14  the simulated rv are  [1]  1.6395601 -2.1287934  2.4862003  3.5620221
2.2968234  0.8285473
 [7]  1.2744164  2.8775734  1.3821320  1.3665802  2.5853245  2.3122296
[13]  1.6336613  1.5426686

 Mean of Simulated was  1.68992471207917
 After  29  iter, Newton is  1.84726039628635
 After  5  iter, Secant is  1.84726052928798

 n=  15  the simulated rv are  [1]  4.4506414  3.9742761  2.1487162  3.3047954
3.9717322  2.4026986
 [7]  1.0427722  0.9848508  2.2116035  3.0066292 20.2862485 10.4047435
[13]  1.6422844  3.3294821  2.8196527

 Mean of Simulated was  4.39874178564007
```

```
 After  41  iter, Newton is  2.81890171647155
 After  7  iter, Secant is  2.81890150543074

 n=  16  the simulated rv are  [1]   3.81720467   2.52303388   3.03137328   0.17
564430  -0.07134503
 [6]  49.36006241   2.97505759   4.03906411   2.57412770   0.97546199
[11]   2.43157839   3.52512086  10.11197126   3.59470738   1.80652422
[16]   1.05367521

 Mean of Simulated was  5.74520388855307
 After  33  iter, Newton is  2.72354535586967
 After  743  iter, Secant is  1.64644973101685e+154

 n=  17  the simulated rv are  [1]   3.68540697 301.68098565   2.15570290   0.40
468040   2.60635959
 [6]   0.09736907   3.92254377   1.06436931   4.50178882   2.91828074
[11]   1.41458334   2.36970326   2.50160514   1.92409901   2.31078684
[16]   0.06767422   3.47186192

 Mean of Simulated was  19.8292824086107
 After  28  iter, Newton is  2.33702970587432
 After  735  iter, Secant is  2.04936845889798e+154

 n=  18  the simulated rv are  [1] -4.9872734  0.5917118  3.1871973  2.4958853
2.0326804  4.3831660
 [7]  2.3271195  3.3334976  0.7692857  2.9799792  6.9568104 -3.0199501
[13]  1.3417098  4.0275165  1.3798935  1.3450783  3.6568165  1.6631121
```

```
 Mean of Simulated was  1.91467980407321
 After  52  iter, Newton is  2.21743997339335
 After  5  iter, Secant is  2.21744035462036

 n=  19  the simulated rv are  [1]   1.322116  -9.000537  35.273097   2.565107
 2.956734   3.543565
 [7]   1.920276   1.305627   1.258581   1.605596   3.071987   2.537025
[13]   2.810091   7.107145   1.992018   1.747365   2.901513   1.473952
[19]   3.203856

 Mean of Simulated was  3.66290072339511
 After  41  iter, Newton is  2.27859942770442
 After  744  iter, Secant is  1.84329264190247e+154

 n=  20  the simulated rv are  [1]   1.74187758   3.06363817  -2.15126518   2.71
862154  21.16855575
 [6]   1.76079646   1.76805016   0.90760972  19.94080976  63.64099563
[11]   3.25832771   0.58670063  -0.09637437   1.88055972   2.42636240
[16]   6.59183716  79.64506088   2.30691818   1.92225636   2.90201703

 Mean of Simulated was  10.799167764946
 After  29  iter, Newton is  2.08515473272233
 After  842  iter, Secant is  -2.09396526559989e+154


 The following columns are
 1)sample size 2)mean of
simulated rv 3)median of simulated rv
```

```
 4)newton method 5)secant method         [,1]    [,2]   [,3]   [,4]         [,5]

 [1,] "n= 1"  "3.17"  "3.17" "3.17" "3.17"
 [2,] "n= 2"  "1.52"  "1.52" "1.52" "1.52"
 [3,] "n= 3"  "2.67"  "2.28" "2.2"  "2.2"
 [4,] "n= 4"  "16.2"  "3.72" "3.07" "42.5"
 [5,] "n= 5"  "2.19"  "2.47" "2.26" "2.26"
 [6,] "n= 6"  "4.78"  "2.83" "2.75" "-1.54e+154"
 [7,] "n= 7"  "2.35"  "2.46" "2.38" "2.38"
 [8,] "n= 8"  "-6.62" "1"    "1.27" "-55.6"
 [9,] "n= 9"  "4.57"  "2.35" "1.98" "21.3"
[10,] "n= 10" "2.81"  "2.49" "2.49" "2.49"
[11,] "n= 11" "3.32"  "2.45" "2.29" "-1.87e+154"
[12,] "n= 12" "2.19"  "2.27" "2.36" "2.36"
[13,] "n= 13" "2.64"  "2.65" "2.72" "2.72"
[14,] "n= 14" "1.69"  "1.64" "1.85" "1.85"
[15,] "n= 15" "4.4"   "3.01" "2.82" "2.82"
[16,] "n= 16" "5.75"  "2.77" "2.72" "1.65e+154"
[17,] "n= 17" "19.8"  "2.37" "2.34" "2.05e+154"
         [,1]   [,2]   [,3]   [,4]         [,5]
[18,] "n= 18" "1.91" "2.18" "2.22" "2.22"
[19,] "n= 19" "3.66" "2.54" "2.28" "1.84e+154"
[20,] "n= 20" "10.8" "2.37" "2.09" "-2.09e+154"
```

5.6 Result of p3 (Replication of the Cauchy MLE)

see p3.ps and p3.tex.

Change before derivative.

```
****When theta= 2  n= 1

Mean of Simulated was  3.48972004962021
After  6  iter, Newton is  3.48972004962021
After  737  iter, Secant is  -2.08905924043462e+154

****When theta= 2  n= 2

Mean of Simulated was  -6.27137399028563
After  7  iter, Newton is  1.43852951618277
After  6  iter, Secant is  1.4385295160216

****When theta= 2  n= 3

Mean of Simulated was  1.2817305870415
After  33  iter, Newton is  1.47504759392051
After  8  iter, Secant is  1.47504744507323

****When theta= 2  n= 4

Mean of Simulated was  1.00983667487776
After  65  iter, Newton is  1.76276655051927
After  6  iter, Secant is  1.76276614488547
```

```
****When theta= 2  n= 5

Mean of Simulated was  0.429260951812849
After  17  iter, Newton is  1.53880264318729
After  6  iter, Secant is  1.53880258580142

****When theta= 2  n= 6

Mean of Simulated was  2.50164414255658

****When theta= 2  n= 1

Mean of Simulated was  3.48972004962021
After  6  iter, Newton is  3.48972004962021
After  737  iter, Secant is  -2.08905924043462e+154

****When theta= 2  n= 2

Mean of Simulated was  -6.27137399028563
After  7  iter, Newton is  1.43852951618277
After  6  iter, Secant is  1.4385295160216

****When theta= 2  n= 3

Mean of Simulated was  1.2817305870415
After  33  iter, Newton is  1.47504759392051
After  8  iter, Secant is  1.47504744507323
```

```
****When theta= 2  n= 4

Mean of Simulated was  1.00983667487776
After  65  iter, Newton is  1.76276655051927
After  6  iter, Secant is  1.76276614488547

****When theta= 2  n= 5

Mean of Simulated was  0.429260951812849
After  17  iter, Newton is  1.53880264318729
After  6  iter, Secant is  1.53880258580142

****When theta= 2  n= 6

Mean of Simulated was  2.50164414255658
After  26  iter, Newton is  2.7967021477128
After  8  iter, Secant is  2.79670221597714

****When theta= 2  n= 7

Mean of Simulated was  9.48362316685796
After  42  iter, Newton is  2.17889716432378
After  5  iter, Secant is  2.1788974094939

****When theta= 2  n= 8

Mean of Simulated was  0.110776756327258
After  49  iter, Newton is  0.507515387422301
```

```
After  9  iter, Secant is  0.507515164513904

****When theta= 2  n= 9

Mean of Simulated was  -1.00028178748055
After  53  iter, Newton is  2.36001839146472
After  6  iter, Secant is  2.36001873149084

****When theta= 2  n= 10

Mean of Simulated was  3.50459128673816
After  40  iter, Newton is  3.70374432869444
After  7  iter, Secant is  1.43681007804469

****When theta= 2  n= 11

Mean of Simulated was  1.63013395505937
After  30  iter, Newton is  1.93650738394577
After  4  iter, Secant is  1.9365072590229

****When theta= 2  n= 12

Mean of Simulated was  1.22982858564425
After  32  iter, Newton is  1.51817224768362
After  5  iter, Secant is  1.51817210786492

****When theta= 2  n= 13
```

```
Mean of Simulated was  -1.60834913316914
```

5.7 Selected program codes

In this section we have a main program of IGS in R^m to find the optimum point of the unimodal function.

Main code

```
function(xmin, xmax, ff)
{
#This might be working fine
#This is a subroutine to find min point for unimodal function ff
#If ff is not unimodal or \delta-unimodal, then force to consider it is unimodal
# xmin <- c(-10, -10, -10, -10)
# xmax <- c(10, 10, 10, 10)
#Define ff outside(not local variable)
#with given rectangular region
#xmin and xmax vector of length p
#ff has a form like f(z)=z[1]^2 + z[2]^2
#if ff has a mountain-type unimodal, then use -ff
#
p <- length(xmin)
a <- xmin
b <- xmax
maxiter <- 1000
epsilon <- 0.01
alpha <- (1/3)
mt <- matrix(c(0, 1), nrow = 1, ncol = 2)
if(p >= 2) {
for(i in 2:p) {
```

```
a0 <- matrix(rep(0, 2^(i - 1)), nrow = 1, ncol = 2^(i - 1))
a1 <- matrix(rep(1, 2^(i - 1)), nrow = 1, ncol = 2^(i - 1))
mt <- cbind(rbind(a0, mt), rbind(a1, mt))
}
}
i <- 1
while(min(abs(a - b)) > epsilon && i <= maxiter) {
z <- NULL
for(k in 1:2^p) {
x2 <- matrix(0, nrow = 4, ncol = 1)
for(j in 1:p) {
temp1 <- (1/3) * cbind((2 - mt[j, k]), (1 + mt[j, k]))
temp2 <- matrix(c(a[j], b[j]), nrow = 2, ncol = 1)
x2[j] <- temp1 %*% temp2
}
x2 <- as.matrix(x2)
z <- cbind(z, ff(x2))
}
for(k in 1:p) {
if(abs(a[k] - b[k]) >= epsilon) {
if(mt[k, order(z)[1]] == 0)
  b[k] <- (1/3) * (a[k] + 2 * b[k])
else a[k] <- (1/3) * (2 * a[k] + b[k])
}
}
i <- i + 1
}
return((a + b)/2)
```

```
}
 Bye bye
```

5.8 Source code of `lsfit` and `qr` written in *Splus*

```
 What?>>lsfit
function(x, y, wt, intercept = T, tolerance = 1e-007, yname = NULL)
{
if(!is.matrix(x))
x <- array(x, c(length(x), 1), list(names(x), "X"))
else x <- as.matrix(x)
specials.x <- !is.finite(x %*% rep(1, ncol(x)))
if(is.matrix(y)) {
specials.y <- !is.finite(y %*% rep(1, ncol(y)))
}
else {
specials.y <- !is.finite(y)
}
if(missing(wt)) {
specials.wt <- F
}
else {
specials.wt <- !is.finite(wt)
}
ok <- !(specials.x | specials.y | specials.wt)
resid <- y * NA
if(is.matrix(y)) {
y <- as.matrix(y)
y <- y[ok,  , drop = F]
y.matrix <- TRUE
```

```
dy <- dim(y)
q <- dy[2]
yn <- dimnames(y)[[2]]
if(length(yn) == 0) {
yn <- if(is.null(yname)) {
paste("Y", 1:q, sep = "")
}
else {
yname
}
length(yn) <- q
}
}
else {
y <- y[ok]
if(y.matrix <- !is.null(yname)) {
yn <- yname[1]
y <- matrix(y, ncol = 1)
resid <- matrix(resid, ncol = 1)
}
dy <- c(length(y), 1)
q <- 1
}
storage.mode(y) <- "double"
x <- x[ok,  , drop = F]
storage.mode(x) <- "double"
dx <- dim(x)
n <- dx[1]
```

```
dn <- dimnames(x)
xn <- dn[[2]]
if(length(xn) == 0)
xn <- paste("X", 1:dx[2], sep = "")
if(n != dy[1])
stop("Number of observations in x and y not equal")
if(intercept) {
dx <- dx + c(0, 1)
xn <- c("Intercept", xn)
x <- array(c(rep(1, n), x), dx)
}
dimnames(x) <- list(dn[[1]], xn)
if(!missing(wt)) {
wt <- as.double(wt[ok])
if(length(wt) != n)
stop("Weight vector has wrong number of observations")
if(any(wt < 0))
stop("Weights must be non-negative")
wt.factor <- wt^0.5
wt.zero <- wt.factor == 0
x0 <- x[wt.zero,  , drop = F]
y0 <- if(y.matrix) y[wt.zero,  , drop = F] else y[wt.zero]
x <- x * wt.factor
y <- y * wt.factor
inv.wt.factor <- 1/ifelse(wt.zero, 1, wt.factor)
}
if((bad.obs <- sum(!ok)) > 0)
warning(paste(bad.obs,
```

```
"observations with NA/NaN/Inf in x, y, or wt removed."))
p <- dx[2]
z <- .Fortran("dqrls",
qr = x,
as.integer(dx),
pivot = as.integer(1:p),
qraux = double(p),
y,
as.integer(dy),
coef = double(p * q),
residuals = y,
qt = y,
tol = as.double(tolerance),
double(2 * p),
rank = as.integer(p))
if(z$rank < p) {
xn <- xn[z$pivot]
dimnames(z$qr)[[2]] <- xn
warning(paste("x-matrix collinear; coefficient estimates set to zer
o for x-variables:",
paste(xn[z$qraux == 0], collapse = ", "), sep = "\n"))
}
if(y.matrix) {
b <- matrix(z$coef, p, q)
dimnames(b) <- list(xn, yn)
z$coef <- b
}
else {
```

```
names(z$coef) <- xn
}
if(!missing(wt)) {
z$residuals <- z$residuals * inv.wt.factor
z$wt <- rep(NA, length(specials.wt))
z$wt[ok] <- wt
if(any(wt.zero)) {
if(y.matrix)
z$residuals[wt.zero,  ] <- y0 - x0 %*% z$coef
else z$residuals[wt.zero] <- y0 - x0 %*% z$coef
}
}
if(y.matrix) {
resid[ok,  ] <- z$residuals
dimnames(resid) <- list(dimnames(resid)[[1]], yn)
}
else {
resid[ok] <- z$residuals
}
z$residuals <- resid
qr <- z[c("qt", "qr", "qraux", "rank", "pivot", "tol")]
z <- z[match(c("coef", "residuals", "wt"), names(z), 0)]
z$intercept <- intercept
z$qr <- qr
z
}
 What?>>qr
function(x, tol = 1e-007)
```

```
{
x <- as.matrix(x)
cmplx <- mode(x) == "complex"
if(!(is.numeric(x) || cmplx))
stop("x must be numeric or complex")
if(!cmplx)
storage.mode(x) <- "double"
dx <- dim(x)
dn <- dimnames(x)
p <- dx[2]
z <- .Fortran(if(!cmplx) "dqr" else "zqr",
qr = x,
as.integer(dx),
pivot = as.integer(1:p),
qraux = if(!cmplx) double(p) else complex(p),
as.double(tol),
if(!cmplx) double(2 * p) else complex(2 * p),
rank = as.integer(p))[c("qr", "qraux", "rank", "pivot")]
if(z$rank < p && length(dn[[2]])) {
pivot <- z$pivot
qr <- z$qr
dn[[2]] <- dn[[2]][pivot]
dimnames(qr) <- dn
z$qr <- qr
}
z
}
```

5.9 Program code of ft4 and ft4a

```
 What?>>ft4
function()
{
#This is ft4
#One parameter mu=1 from N(mu,2) using BOS
# sink("ft4.tex")
cat("\n This is output from ft4 \n")
cat("\n Normal mu=1") # ps("ft4.ps")
win.graph()
cat("\n Note that we want answer is 1")
par(mfrow = c(8, 6))
for(i in 1:1) {
xmin <- c(1 - 2 * runif(1))
xmax <- c(1 + 2 * runif(1))
cat("\n i=", i, "\n xmin=", xmin, " xmax=", xmax)
cat("\n nlmin1=", (xmax + xmin)/2, " nlmin2=", 2)
cat(" \n")
ans <- NULL
mse1 <- NULL
mse2 <- NULL
abs1 <- NULL
abs2 <- NULL
IGS <- NULL
nlmin <- NULL
rep <- 30
for(k in 11:rep) {
```

```
n <- 20 #x <- sort(rnorm(n, mean = 1, sd = sqrt(2)))
ff <- function(z)
{
n <- 20 #x <- sort(rnorm(n, mean = 1, sd = sqrt(2)))
#zz <- mean(x) #n <- k
x <- sort(rnorm(n, mean = 1, sd = sqrt(2)))
x0 <- c(1:n)/(n + 1)
zz <- pnorm(x, mean = z, sd = sqrt(2)) - x0
zz <- sum(zz^2)
return(zz)
}
l <- (1/50) * c(1:100)
ll <- NULL
for(j in 1:100) {
ll <- cbind(ll, ff(l[j]))
}
plot(l, ll, type = "l")
title(main = "curve", sub = paste("When data= ", n))
la <- (1/5) * c(1:100) - 10 #cat("\n", la, "\n")
lla <- NULL
for(j in 1:100)
lla <- cbind(lla, ff(la[j])) #cat("\n", lla, "\n")
plot(la, lla, type = "l")
title(main = "crude")
z1 <- ft1(xmin, xmax, ff)
z2 <- nlmin(ff, (xmin + xmax)/2)$x
z3 <- nlmin(ff, 2)$x
cat("\n When k= ", k, " IGS=", z1, " nlmin1=", z2, " nlmin2=", z3)
```

```
#   fg1(ff, xmin, xmax, 100)
abs1 <- cbind(abs1, z1 - 1)
abs2 <- cbind(abs2, z2 - 1)
ans <- cbind(ans, rbind(as.matrix(z1), as.matrix(z2)))
mse1 <- cbind(mse1, sum((z1 - c(1, 2, 3))^2))
mse2 <- cbind(mse2, sum((z2 - c(1, 2, 3))^2))
IGS <- cbind(IGS, z1)
nlmin <- cbind(nlmin, z2)
}
#print(ans)
cat("\n", IGS)
cat("\n", nlmin)
cat("\n The mean of IGS is ", mean(IGS))
cat("\n The mean of nlmin is ", mean(nlmin))
mse1 <- sort(as.vector(mse1))
mse2 <- sort(as.vector(mse2))
abs1 <- sort(as.vector(abs1))
abs2 <- sort(as.vector(abs2))
cat("\n The CI of MSE of grid search of above ", rep, " rep is \n [", mse1[
floor(rep/4)], " , ", mse1[ceiling((3 * rep)/4)], "] \n")
cat("\n The CI of MSE of q-Newton of above ", rep, " rep is \n [ ", mse2[
floor(rep/4)], " , ", mse2[ceiling((3 * rep)/4)], " ] \n")
plot(density(mse1), type = "l")
title("MSE GS of Normal in ft4")
plot(density(mse2), type = "l")
title("MSE q-Newton of Normal in ft4")
plot(density(abs1), type = "l")
title("Abs Error GS of Normal in ft4")
```

```
plot(density(abs2), type = "l")
title("Abs Eror q-Newton Normal, ft4")
}
sink() # graphics.off()
}


 What?>>ft4a
function()
{
#This is ft4a
#One parameter mu from N(mu,2)
cat("\n This is output from ft4a \n")
ps("graph1.ps")
par(mfrow = c(6, 5))
for(i in 1:1) {
#cat("\n When i=", i, "\n")
xmin <- c(1 - 2 * runif(1))
xmax <- c(1 + 2 * runif(1))
cat("######### \n") #cat("\n When xmin=", xmin, " xmax=", xmax, "\n")
ans <- NULL
mse1 <- NULL
mse2 <- NULL
rep <- 30
for(k in 1:rep) {
n <- k #cat("\n When i=", i, " when k= ", k)
ff <- function(z)
{
```

```
n <- 20
x <- sort(rnorm(n, mean = 1, sd = sqrt(2)))
x0 <- c(1:n)/(n + 1)
zz <- pnorm(x, mean = z[1], sd = sqrt(2)) - x0
zz <- sum(zz^2)
return(zz)
}
l <- (1/5) * c(1:100) - 10
cat("\n", l, "\n")
ll <- NULL
for(j in 1:100)
ll <- cbind(ll, ff(l[j]))
cat("\n", ll, "\n")
plot(l, ll, type = "l")
}
#print(ans)
mse1 <- sort(as.vector(mse1))
mse2 <- sort(as.vector(mse2))
cat("\n The CI of MSE of grid search of above ", rep, " rep is \n [", mse1[
floor(rep/4)], " , ", mse1[ceiling((3 * rep)/4)], "] \n")
cat("\n The CI of MSE of q-Newton of above ", rep, " rep is \n [ ", mse2[
floor(rep/4)], " , ", mse2[ceiling((3 * rep)/4)], " ] \n")
plot(density(mse1), type = "l")
title("Grid Search of Normal in ft4a")
plot(density(mse2), type = "l")
title("q-Newton of Normal in ft4a")
}
}
```

5.10 Result from ft4 (BOS in the Normal distribution)

```
 What?>>ft4()

 This is output from ft4

 Normal mu=1
 Note that we want answer is 1
 i= 1
 xmin= 0.392746951431036  xmax= 1.48250934202224
 nlmin1= 0.937628146726638  nlmin2= 2

 k= 11  IGS= 0.821822655378693  nlmin1= 0.937628088479663  nlmin2= 2.0
3124999874515
 k= 12  IGS= 1.47411016553442  nlmin1= 0.937631961422736  nlmin2= 2.00
003051625124
 k= 13  IGS= 0.77235445870094  nlmin1= 0.941534396054292  nlmin2= 1.99
998474121094
 k= 14  IGS= 1.07942679265455  nlmin1= 0.937628143001359  nlmin2= 1.99
999988079071
 k= 15  IGS= 1.16684302646207  nlmin1= 0.937620536287558  nlmin2= 1
 k= 16  IGS= 0.981817124686647  nlmin1= 0.937872287322389  nlmin2= 1.9
9999999254942
 k= 17  IGS= 0.68518019335669  nlmin1= 0.93762813182548  nlmin2= 2.000
01525881984
 k= 18  IGS= 0.783378377841198  nlmin1= 0.953253146311266  nlmin2= 2.0
0024414035548
 k= 19  IGS= 1.05941312992968  nlmin1= 0.938604708160165  nlmin2= 2.00
```

```
012207028505
 k= 20  IGS= 0.931471279684482  nlmin1= 1.43762806153177  nlmin2= 2.06
249999878953
 k= 21  IGS= 0.477919850502798  nlmin1= 0.937624332029744  nlmin2= 1
 k= 22  IGS= 0.766383169166632  nlmin1= 0.93811642797152  nlmin2= 2.00
006103515355
 k= 23  IGS= 1.24501934590092  nlmin1= 0.937689182008675  nlmin2= 2.00
012207591321
 k= 24  IGS= 1.20005012217587  nlmin1= 0.953253146711993  nlmin2= 1
 k= 25  IGS= 1.02285538617363  nlmin1= 0.937628116924448  nlmin2= 2.00
012207002594
 k= 26  IGS= 0.504134467744068  nlmin1= 0.937567116611005  nlmin2= 1.9
9996948242188
 k= 27  IGS= 1.05893329416353  nlmin1= 0.938116427902701  nlmin2= 2.00
781249693615
 k= 28  IGS= 1.22037957425111  nlmin1= 0.937628385130752  nlmin2= 2.00
000001490116
 k= 29  IGS= 0.973682473044272  nlmin1= 0.937872287343199  nlmin2= 1.5
 k= 30  IGS= 0.58254865448579  nlmin1= 0.937658664300733  nlmin2= 1.96
875
 The mean of IGS is  0.9403861770919
 The mean of nlmin is  0.964509177366572
 The CI of MSE of grid search of above  30  rep is
 [ 4.65411098044605  ,  NA ]

 The CI of MSE of q-Newton of above  30  rep is
 [  5.38279016207036  ,  NA  ]
```

5.11 Program codes p1 in the Cauchy problem

```
 What?>>p1
function()
{
win.graph() # ps("p1.ps")
par(mfrow = c(5, 4))
theta <- 2
xmin <- -10
xmax <- 10
for(n in 1:20) {
#for(n in 1:1) {
nn <- n
cat("\n Doing n=", nn, "\n") #        n <- 3
z1 <- runif(nn)
z2 <- tan(pi * (z1 - 0.5)) + theta
x <- seq(-20, 20, 0.1)
z3 <- NULL
tt <- NULL
for(i in 1:100) {
t <- (i - 50)/5
tt <- cbind(tt, t)
# z3 <- cbind(z3, sum((z2 - t)/(1 + (z2 - t)^2)))
z3 <- cbind(z3, sum(log(1 + (z2 - t)^2)))
}
#    cat("\n The   z3 is", z3, "\n")
plot(tt, z3, type = "l")
zzz <- paste("Cauchy ML function \n when ss= ", n)
```

```
#title(main = zzz, sub = paste("min point=", ft1(xmin, xmax, tt)))
title(main = zzz)
}
#graphics.off()
}


 What?>>p2
function()
{
# Using simulated cauchy variate, Find the MLE using 1)Newton , 2)Secant
theta <- 2 #       n <- 4
# Initial setting
# ps("p2.ps")
win.graph()
par(mfrow = c(7, 6)) # sink("p2.tex")
maxiter <- 10000
int <- 0.01
epsilon <- 1e-007
M <- matrix(0, nrow = 20, ncol = 5)
for(n in 1:20) {
# Generate Cauchy Variaite
z1 <- runif(n)
z2 <- tan(pi * (z1 - 0.5)) + theta
M[n, 1] <- paste("n=", n)
M[n, 2] <- signif(mean(z2), 3)
M[n, 3] <- signif(median(z2), 3)
cat("\n n= ", n, " the simulated rv are ")
```

```
print(z2)
cat("\n Mean of Simulated was ", mean(z2), "\n")
# Define function \frac{\partial}{\partial theta} f
f <- function(z, t)
{
sum((-2 * (z - t))/(1 + (z - t)^2))
}
z <- z2
fl <- function(z, t)
{
prod(1/(1 + (z - t)^2))
}
# Plotting of f wrt theta
temp <- seq(theta - 5, theta + 5, 0.1)
temp2 <- NULL
for(i in 1:100) {
temp2 <- cbind(temp2, f(z2, temp[i]))
}
plot(temp[1:100], temp2, type = "l")
abline(0, 0)
zzz <- paste("plot of f \n when ss=", n)
title(main = zzz) # Find Derivative
deriv <- function(z, t)
{
sum((2 * (1 + 3 * (z - t)^2))/(1 + (z - t)^2)^2)
}
temp2 <- NULL
for(i in 1:100) {
```

```
temp2 <- cbind(temp2, fl(z, temp[i]))
}
plot(temp[1:100], temp2, type = "l") # abline(0
, 0)
zzz <- paste("plot of fl \n when ss=", n)
title(main = zzz) # Newton Method
#told <- theta # + 3 * runif(1)
told <- median(z2)
tnew <- told + int
t1 <- told
t2 <- tnew
i <- 1
while(abs(t1 - t2) >= epsilon && i <= maxiter) {
t1 <- told
tnew <- told - f(z2, told)/deriv(z2, told)
t2 <- tnew
told <- tnew
i <- i + 1
}
if(i == maxiter + 1)
cat("The Newton does not converge \n ")
else cat(" After ", i - 1, " iter, Newton is ", tnew, "\n")
M[n, 4] <- signif(tnew, 3) # Secant method
told <- mean(z2)
thalf <- told + int
i <- 1
while(abs(told - thalf) >= epsilon && i <= maxiter) {
tnew <- thalf - (f(z2, thalf) * (thalf - told))/(f(z2,
```

```
thalf) - f(z2, told))
told <- thalf
thalf <- tnew
i <- i + 1
}
if(i == maxiter + 1)
cat(" The Secant does not converge \n")
else cat(" After ", i - 1, " iter, Secant is ", tnew, "\n")
M[n, 5] <- signif(tnew, 3) # xmin <- -10
# xmax <- 10
# cat(" The GS is ", ft1(xmin, xmax, fl), "\n ")
}
# sink("p2out.tex")
cat("\n \n The following columns are \n 1)sample size 2)mean of\nsimulat
ed rv 3)median of simulated rv \n 4)newton method 5)secant method"
)
return(M) #Grid Search
# sink()
# sink()
# graphics.off()
}
```

5.12 Example of roundoff error

```
 What?>>version
Version 3.2 Release 1 for MS Windows 3.1 : 1994
 What?>>print((10^-1)^2,digits=20)
```

```
[1] 0.0100000000000000016
 What?>>print((10^2)^-1,digits=20)
[1] 0.01
 What?>>print(10^-2,digits=20)
[1] 0.0100000000000000016
 What?>>print(1.0e-2,digits=20)
[1] 0.01
 What?>>10^-2==1.0e-2
[1] F
```

Chapter 6

Summary

This dissertation discusses a family of procedures, based on the iterated grid search algorithm, which does not require conditions such as regularity or the existence of derivatives. One of the basic requirements of this project is to find an algorithm that is *completely stable*, implying that we can *always* find the optimum point of a parametric function under given conditions. The purpose of this dissertation is to present an algorithm for parameter estimation on unimodal functions or almost unimodal functions. This research includes the mathematical extension of the domain of available functions for the Iterated Grid Search algorithm.

There are many ways in past literature, including the widely-used Newton-type methods, to estimate the parameter in the Least Square criterion. Previous research provides ways for solving nonlinear Least Square problems using derivative methods but not many using derivative-free methods. In this project, we present a relatively simple algorithm *without derivatives*, not only for Least Square but also for other proposed criteria. In many cases, the Newton-type methods work more efficiently than the derivative-free methods; however, in some cases, the Newton-type methods cannot be used for certain parametric functions, due to chaotic behavior, even though the quadratic information makes it converge faster. In this research we are interested in avoiding the danger of encountering chaotic behavior of object functions. Some examples are given in chapter 1 to show why the proposed Iterated Grid Search algorithm for statistical optimization is recommended and should be used in these cases.

Statistical packages, such as *Splus*, *SAS* use a class of the Newton-type methods to find the optimum point of a parameter in object function (parametric function or likelihood function). In this project, for restricted cases, we suggest a relatively simple iterated grid search method. This new iterated grid search has a *guaranteed convergence* property because of its simple structure, and

furthermore we can even predict how long it takes. The IGS algorithm is suitable for unimodal functions but it is applied at first to convex functions, since the notion of convexity is characterized to application and unimodality is not yet in past literature. We found that there is no support for the grid search module in *SAS*, *Splus*, and *IMSL*.

REFERENCES

[1] Avriel, M. (1976), *Nonlinear Programming:Analysis and methods*. Prentice-Hall.

[2] Bard. Y. (1974), *Nonlinear parameter estimation*. Academic Press.

[3] Bates, D. and Watts, D. (1988), *Nonlinear regression analysis and its applications*. Wiley.

[4] Barzilai, J. and Dempster, M. (1993), "Measuring rates of convergence," *Journal of Optimization Theory and Application* Vol78, No1

[5] Bazarra, M. S., Sherali, H. D. and Shetty, C. M. (1993), *Nonlinear Programming: Thoery and Algorithms*. 2nd edition. John Wiley & Sons.

[6] Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988), *The New S Language*. Wadsworth & Brooks Cole, Computer Science Series.

[7] Berman, G. (1966), "Minimization by successive approximation," *SIAM Journal on Numerical Analysis*. 3:123-133

[8] Brent, R. P. (1973), *Algorithms for minimization without derivatives*. Prentice Hall.

[9] Breuer, S. and Zwas, G. (1993), *Numerical Mathematics: a laboratory approach*. Cambridge Press.

[10] Chambers, J. M. (1977), *Computational Methods for data analysis*. Wiley.

[11] Diewert, W. E. (1993), *Mathematical modelling in economics*. Springer-Verlag.

[12] Falk, J. E. and Hoffman, K. R. (1976), "A successive underestimating method for concave minimization problems," *Mathematics of Operations Research* 1:251-259.

[13] Falk, J. E. (1969), "Lagrange multiplier and nonconvex programs," *SIAM Journal of Control* Vol. 7, No. 4

[14] Fiacco, A. V. and McCormick, G. P. (1968), *Nonlinear Programming:Sequential Unconstrained Minimization Techniques*. Wiley.

[15] Flemming, W. H. (1965), *Functions of several variables*. Addison-Wesley.

[16] Forsyth, G. E. and Malcolm, M. A. (1977), *Computer methods for mathematical computations*. Prentice Hall.

[17] Fletcher, R. (1969), *Optimization*. Academic Press.

[18] Huber, P. J. (1981), *Robust Statistics*. Wiley.

[19] Kennedy, W. J. and Gentle, J. E. (1980), *Statistical Computing*. Marcel-Dekker.

[20] Kiefer, J. (1953), "Sequetial minimax search for a maximum," *Proceedings of AMS*. 4:502-506.

[21] Kiefer, J. (1957), "Optimal sequential seach and approximation methods under minimum regularity assumptions," *Journal of the Society for Industrial and Applied Math*. 5:105-136.

[22] Lehmann, E. L. (1983), *Theory of Point Estimation*. Wiley.

[23] Lamport, L. (1986), LaTeX*: A document preparation system*. Addison-Wesley.

[24] Lewis, P. A. W. and Orav, E. J. (1996), *Simulation methodology for statisticians*. Vol1 Kluer Academie Publishers.

[25] McCormick, G. P. (1976), "Computability of global solutions to factorable nonconvex programs," *Mathematical programming* 10:147-175

[26] More, J. J. and Wright, S. J. (1993), *Optimization Software Guide*. Society for industrial and applied mathematics.

[27] Ortega, J. M. and Rheinbolt, W. C. (1970), *Iterative solution of Nonlinear Equations in several variables*. Academic Press.

[28] Reeds, J. A. (1985), "The number of Cauchy roots," *The Annals of Statistics* 13:775-784.

[29] Rockafellar, R. T. (1970), *Convex Analysis*. Princeton University Press.

[30] Rudin. *Real Analysis*

[31] SAS Institute, Inc. (1985), *SAS User's guide: Statistics version 5 edition*.

[32] SAS Institute, Inc. (1993), *SAS companion for the Microsoft Windows Environment*.

[33] Statistical Science, Inc. (1993), *S-PLUS Programmer's Manual*.

[34] Statistical Science, Inc. (1993), *S-PLUS for Windows: User's Manual*.

[35] Searle, S. R. (1971), *Linear Models*. John Wiley and Sons.

[36] Spath, H. (1992), *Mathematical algorithms for linear regression*. Academic Press.

[37] Thisted, R. A. (1988), *Elements of Statistical Computing*. Chapman and Hall.

[38] Thisted, R. A. (1989), "Improving on Fibonacci Search: On minimization by successive approximation," Technical Report #190, Department of Statistics, The University of Chicago.

[39] Van Loan, C. F. and Golub, G. H. (1989, 1996), *Matrix Computation*. Johns Hopkins.

[40] Wilde, D. J. (1964), *Optimum Seeking Methods*. Prentice-Hall.

[41] Wolfram, S. (1991), *Mathematica: A system for doing mathematics by computer*. 2nd Ed, Addison Wesley.

[42] Zanakis, S. H. and Rustagi, J. S. (1982), *Optimization in Statistics*. North-Holland.

[43] Zangwill, W. I. (1969), *Nonlinear Programming: A Unified Approach*. Prentice-Hall.

VITA

Jinhyo Kim was born in Seoul, Korea on July 21, 1963. He received a BA degree in Computer Science and Statistics from Seoul National University in February 1986.

From the fall of 1986, he was enrolled in master's program of Statistics in NC State University at Raleigh, NC. Upon completing the first program, from the fall of 1988 he studied in the applied mathematics master program of Mathematics department in NC State University.

After graduation, he served in the Korean Army, and the author began his PhD coursework studies in the University of Illinois at Urbana-Champaign. He continued his studies in the Department of Statistics at Virginia Tech and, with two year's leave, completed the requirement for a PhD degree in Statistics in June, 1997.

The author is currently a member of the Association of Korean Scientists in America, Mu Sigma Rho, and the American Statistical Association.