A GENERIC AUTOMATIC NUMERICAL STABILITY TESTING METHOD

A GENERIC AUTOMATIC NUMERICAL STABILITY TESTING METHOD

BY

HANG ZHOU, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2010) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE: A GENERIC AUTOMATIC NUMERICAL STABILITY TESTING METHOD

AUTHOR: Hang Zhou

B.Sc., (Electrical Engineering)

University of Science and Technology of China, Hefei, China

SUPERVISOR: Dr. Sanzheng Qiao

NUMBER OF PAGES: ix, 57

To my family

Abstract

In this thesis we develop a new automatic method to test algorithm's numerical stability. The new method is a combination of two stability testing methods introduced by Higham and Kahan, and takes advantage of the both method. We first generate an objective function which can reveal algorithm's stability, then locate the maximum value of the objective function by an optimization method. This method can automatically detect the unstable points and reveal algorithms instability.

To make our method suitable for general problems, we generate the objective function by measuring the difference of the test program's results computed in different rounding modes. We choose a search method suitable for our objective function. To improve the accuracy of our method, the final result is obtained by combining the results from multiple searches. Furthermore, we propose three measurements to measure algorithm's stability. Practical examples are used to test the performance of our method.

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor Dr. Qiao for his expert supervision, insightful discussions throughout the course of the work. His patient and consistent advice is the greatest support for the thesis.

I am very grateful to offer my appreciation to my colleagues and friends for there valuable comments and discussions. They were always there when I needed their support and encouragement.

I also wish to acknowledge my two aunts in Toronto who helped me a lot not only in the academic but also in my daily life in the two years. Their unconditional support encouraged me a lot.

Finally, I would like to thank my parents for their love and support through all my life.

Contents

List of Figures

Chapter 1

Introduction

Numerical stability is a important feature of a numerical algorithm. There are different ways to formalize the concept of stability. How to define stability depends on the context. Normally, numerical stability refers to how a perturbation of input affects the output of an algorithm. In a numerically stable algorithm, errors in the input lessen in significance as the algorithm executes, having little effect on the final output.

Testing the numerical stability is always needed to investigate an algorithm's performance. However, most of the automatic numerical stability testing methods are designed for certain problems. For example, Langlois' [6] stability testing method for liberalization, Miller's [7] stability testing method for algebraic processes, Larson and Sameh's [4] [5] stability testing method for linearized problems. People generate their stability testing method based on the understanding of the specific problem, and those stability testing methods can not be used on other problems.

The task of finding a generic stability testing method is attractive. Rowan [11]

develops a method which measures stability by the difference between the test algorithm's results computed in double precision and single precision. Another method called CESTAC is developed by La Porte and Vignes [14]. It measures stability by making a random perturbation of the last bit of the result of each elementary operation. However, these generic stability testing methods need to deeply change the code of the test program.

In this thesis we will present a simple generic numerical stability testing method. However, as Kahan said, "error-analysis can't be automated in general" [3, p. 58], our stability testing method does not guarantee a complete error-analysis. When the condition number of the problem is known, our method can detect instability. When the condition number of the problem is unknown, our method can give indication of instability.

Chapter 2

Background

Testing stability of a numerical program is nontrivial. Random testing is inadequate, since random data are usually well-behaved. For example, a random matrix is often well-conditioned. In this chapter, first in section 2.1 we present an automatic instability detection method due to Higham [2, p. 471]. This method requires an objective function and applies an optimization method to the objective function to find input data that reveal potential instability. Then in section 2.2 we describe Kahan's method [3, p. 56], which requires proper input data and uses different rounding modes to test stability.

2.1 Higham's Automatic Instability Detection Method

Higham introduced an automatic stability testing method by using direct search optimization [2, p. 471]. His method consists of two steps. First we define an objective function $y = f(x)$ that reflects the stability of algorithm; second, use a direct search method to find the maximum value of the objective function.

2.1.1 Objective Function

The objective function

$$y = f(x), f : \mathbb{R}^n \rightarrow \mathbb{R} \tag{2.1}$$

is a function mapping algorithm's input vector x into a stability measurement value y. The determination of the objective function depends on the problem. For example, we can use the growth factor factor for Gaussian elimination on $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ [2, p. 472],

$$f(x) = \rho_n(A) = \frac{max_{i,j,k}|a_{ij}^{(k)}|}{max_{i,j}|a_{ij}|}, x \in \mathbb{R}^{n^2}, A \in \mathbb{R}^{n \times n}, \tag{2.2}$$

where the $a_{ij}(k)$ are the intermediate values of the entries of A generated during Gaussian elimination. Since the growth factor governs the stability of Gaussian elimination, so we can measure the problem's stability based on the size of $\rho_n(A)$.

2.1.2 Direct Search Method

A direct search method for the problem

$$\max_{x \in R^n} f(x), \mathbb{R}^n \rightarrow \mathbb{R} \tag{2.3}$$

is a numerical method that attempts to find a maximum point of $x \in \mathbb{R}^n$ [2, p. 472]. It does not require derivatives of f and uses function values only so that it is suitable for those functions lack of smoothness or difficult to obtain derivatives when they exist. For problems in which f is smooth and its derivatives are available, direct search methods are less efficient than those optimization methods that use derivatives such as Newton methods [10].

One of the most widely used direct search method is multi-directional search (MDS) method.

The Multi-Directional Method

Suppose the dimension of the problem's input is n ($x \in \mathbb{R}^n$). The MDS method requires $n+1$ points at initial step. These points define a simplex $\{v_0, v_1, v_2\}$ in $\mathbb{R}^n$, and it is assumed that $f(v_0) = \max_i f(v_i)$. One iteration in the case $n = 2$ is illustrated in Figure 2.1.

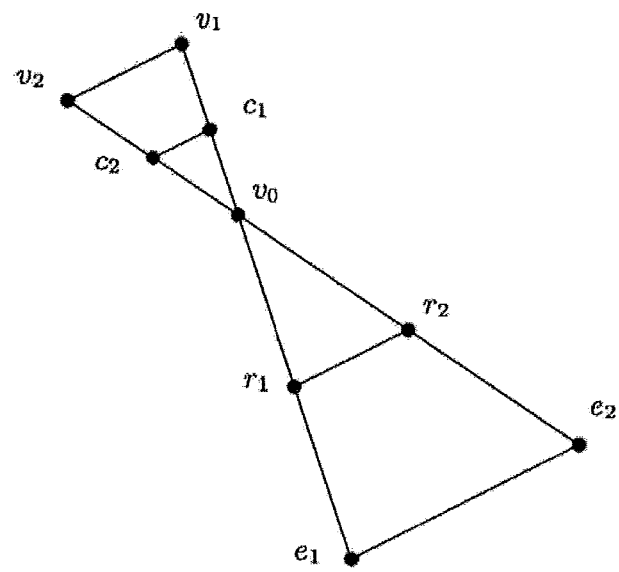

Figure 2.1: The possible steps in one iteration of the MDS method when $n = 2$ [2, p. 475]

In the first step the original simplex is reflected through v_0 to give a new simplex $\{v_0, v_1, v_2\}$. If $\max_i f(r_i) > f(v_0)$,then the reflection step is considered to be successful and the following expansion step takes place. The edges from v_0 to r_i ($i = 1, 2$) are

doubled in length to give a new simplex $\{v_0, e_1, e_2\}$. If $\max_i f(e_i) > \max_i f(r_i)$, the original simplex $\{v_0, v_1, v_2\}$ is replaced by $\{v_0, e_1, e_2\}$, otherwise the original simplex is replaced by $\{v_0, r_1, r_2\}$.

If the reflection step is unsuccessful, the following contraction step takes place. the contracted simplex is given by shrinking the edges from v_0 to v_i $(i = 1, 2)$ of the original simplex to half their lengths. The algorithm will then start the next iteration with the contracted simplex $\{v_0, c_1, c_2\}$.

So, if v_0 is a local maximum point, the simplex keeps shrinking since no other points in those reflection simplices satisfy that $f(v) > f(v_0)$. When the stopping criteria is reached, the search stops.

2.1.3 An Example

Here we give an example of using direct search method to detect the stability of numerical algorithm for finding the roots of a cubic function[2, p. 479].

By dividing through the leading coefficient, a nondegenerate cubic equation can be presented in the form

$$f(x) \equiv x^3 + ax^2 + bx + c = 0. \tag{2.4}$$

Changing the variable

$$x = y - \frac{a}{3}, \tag{2.5}$$

the quadratic term is eliminated

$$y^3 + py + q = 0, p = -\frac{a^2}{3} + b, q = \frac{2}{27}a^3 - \frac{ab}{3} + c. \tag{2.6}$$

With the subsitution $y = \omega - p/(3\omega)$ we get

$$\omega^3 - \frac{p^3}{27\omega^3} + q = 0. \tag{2.7}$$

Putting it in the form

$$(\omega^3)^2 + q\omega^3 - p^3/27 = 0, \tag{2.8}$$

we get

$$\omega^3 = -\frac{q}{2} \pm \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}. \tag{2.9}$$

In this problem, the variables are the coefficients a, b, c. We take the relative error formula as the objective function:

$$\|z - \widehat{z}\|_\infty / \|z\|_\infty. \tag{2.10}$$

To compute the "exact" roots z, we use MATLAB's roots function. Here we take the "+" square in (2.9). The optimization method used is the MDS method. Starting with $[a \quad b \quad c] = [1 \quad 1 \quad 1]$, the vector found by the MDS method is $[a \quad b \quad c] = [1.732 \quad 1 \quad 1.2704]$. The computed and "exact" roots at this point are respectively

$$\widehat{z} = \begin{bmatrix} -1.5999e+0 \\ -6.6066e-2-8.8557e-1i \\ -6.6066e-2+8.8557e-1i \end{bmatrix}, z = \begin{bmatrix} -1.6026e+0 \\ -6.6478e-2-8.8798e-1i. \\ -6.6478e-2+8.8798e-1i \end{bmatrix}$$

The above results show that the algorithm is very unstable, the relative error is about 0.04. Notice that objective function requires the exact solution or an accurate solution, which is usually unavailable.

2.2 Kahan's Stability Testing Method

Kahan introduced a method for testing numerical stability[3, p. 56]. The idea is to run a program in different rounding modes with the same input and check the differences between the results obtained by different rounding modes.

2.2.1 Rounding Modes

As we know, IEEE Standard 754 specifies four rounding modes for binary floating-point arithmetic [1]. The default rounding mode is rounding to the nearest, which is what all the programmers use in most situations. Other three rounding modes are: rounding towards zero (truncate), rounding up (towards $+\infty$), and rounding down (towards $-\infty$). With a rounding modes setting command, we can change a program's entire rounding mode environment, influencing every floating-point operation in that program.

We run a program in all four rounding modes and compare their results. If the problem is well-conditioned and the difference between the results computed in different rounding modes is large, then we can say that the algorithm is numerically unstable.

2.2.2 An Example

Here we give an example of the application of Kahan's stability testing method[3, p. 59].

Let a, b, c be the three side-lengths of a triangle, and assume that $a > b > c$.

	Heron's formula	Improved formula
To nearest	12345680.0	6249012.0
Upward	12345680.0	6249013.0
Downward	0	6249011.0
Toward zero	0	6249011.0

Table 2.1: Sensitivity to rounding of two triangle area calculating algorithms

A classical formula due to Heron is

$$\Delta = \sqrt{s(s-a)(s-b)(s-c)}, s = (a+b+c)/2. \tag{2.11}$$

An improved formula of computing the area of a triangle is

$$\Delta = \frac{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}{4}. \tag{2.12}$$

Compare both formulas on a needle-like triangle

$$a = 12345679, b = 12345679, c = 1.01233995.$$

Compute this problem with both formulas in four different rounding modes. The result is shown in Table 2.1.

Table 2.1 shows that when applying the first formula, the results computed in four different rounding modes change dramatically. We can get the conclusion that the first formula is numerically unstable.

Notice that in this method, the selection of input data is critical. For example, if we change the lengths of three sides to $a = 12345, b = 1234, c = 123456$, for both algorithms, all the results computed in four different rounding modes are 3460216.2 - no difference is shown.

2.3 Summary and Discussion

Highanm's automatic stability testing method requires a proper objective function. The performance of his idea largely depends on the quality of the objective function. The construction of a good objective function is not always an easy task. Moreover, a disadvantage of this idea is that the objective function should be constructed specifically for a particular problem. Of course, the relative forward error formula $\frac{\|\hat{y}-y\|}{\|y\|}$ could be used as a simple and universal objective function. But this objective function requires an accurate result of the problem, which is usually unavailable.

Kahan's stability testing method is a simple method. Unfortunately, it is not automatic. To show a problem's instability, it requires appropriate input data. For most problems, finding a appropriate input data to show instability is a challenge if we don't have a highly clear understanding of the problem. Is there any method to construct appropriate input data automatically, and make the method an automatic stability testing method?

Combining Higham's and Kahan's stability testing methods, we can get a new automatic stability testing method. It does not require an objective function from the user, and can automatically find proper input data.

Chapter 3

Basic Idea of the New Automatic Numerical Stability Testing Method

In last chapter we discussed the disadvantages of the Higham's and Kahan's stability testing methods. In this chapter we describe our new stability testing method, which is a combination of the Higham's and Kahan's methods. In section 3.1 we describe the goal of our method. The main step of our method is presented in section 3.2. Then in section 3.3 we discuss the construction of a general objective function. In section 3.4 we give a discussion about the justification of our method.

3.1 Our Goal

Our goal is to develop an automatic numerical stability testing method with the following features:

- It provides an objective function suitable for general problems.
- The objective function is simple, easy to understand.
- The objective function can reflect problem's stability.
- Appropriate input data for the objective function which shows problem's instability can be found automatically.

We propose a new automatic stability testing method with all the properties above. The Higham's stability method requires an objective function, and the Kahan's method requires appropriate input data. The objective function and the appropriate input data are both hard to obtain in practice. Compared with Kahan's method, our method uses direct search method to find appropriate input data. Compared with Higham's method, our method uses different rounding modes to construct an objective function.

3.2 Main Steps of Our Method

To make the new stability testing method an automatic method, we take Highan's automatic method as a framework. Like Higham's method, the new stability testing method contains two steps:

- Obtain an objective function using the Kahan's method.
- Use direct search method for a maximizer of the objective function.

The core of the new automatic stability testing method is the objective function part. The construction of objective function largely affects the performance of the new method. Section 3.3 will present the details of how to construct objective function.

After an objective function is constructed, there are several questions remain to be answered. Can the constructed objective function reveal problem's stability? How to

measure problem's stability by the maximum value of the objective function found by the direct search method? We attempt to answer these questions by a combination of a further discussion and practical experiments. The further discussion are presented in section 3.4, and the practical experiments are shown in Chapter 5.

The second step of the new method - finding the proper input which shows instability seems to be easy, we just need to apply our direct search method on the objective function, and locate the point where the objective function reaches a maximum value. In practice, the situation turns out to be complicated. Since the direct search methods are far from being perfect, it often fails to find the most unstable point of algorithm. In this situation, we have to try to find those points with "large engough" value which can show algorithm's instability. Details about the implementation of instability searching will be presented in Chapter 4.

3.3 Construction of the General Objective Function

In this section we will solve two problems: why we need a general objective function and how to construct a general objective function.

3.3.1 Why We Need a General Objective Function

As mentioned before, in order to phrase the stability testing method as an optimization problem and apply the direct search method, the objective function should be constructed in this form:

$$y = f(x), y \in \mathbb{R}, x \in \mathbb{R}^n. \tag{3.1}$$

The first step of our method is like the first step of Higham's stability testing method, which is defining an objective function $y = f(x)$ that reflects the stability of algorithm. As mentioned in section 2.3, in Higham's method the construction of the objective function is problem dependent.

For example, for Strassen's matrix inversion method [2, p. 478], the stability measure is

$$f(x) = \frac{min\{\|A\widehat{X} - I\|_\infty, \|\widehat{X}A - I\|_\infty\}}{\|A\|_\infty \|\widehat{X}\|_\infty}, \tag{3.2}$$

where $\widehat{X}$ is the inverse of the matrix A computed using Strassen's inversion method.

The disadvantage of constructing the objective function specifically for problems is that we have to find a stability measure formula for the objective function for every single problem to be tested. Often a good measure formula is hard to obtain.

The problem can be solved by defining a general objective function. We propose a general objective function which can measure every problem's numerical stability.

In section 3.2.2 we will describe how to construct a general objective function by applying Kahan's stability testing method.

3.3.2 Constructing a General Objective Function

In section 2.2.2 a simple objective function introduced by Higham is taken as an example. It's the relative forward error formula

$$E_{rel}(x) = \frac{\|\widehat{y} - y\|}{\|y\|}, \tag{3.3}$$

where $\widehat{y}$ is the result computed by the program and y is the exact or accurate result. This objective function is used to measure the numerical stability of solving a cubic

equation. Higham took the results computed by MATLAB's roots function as the accurate results y. This objective function is not suitable for our general function, because the accurate result is usually unavailable. We can't always use MATLAB or other tools to get the accurate result. Here we construct a general objective function which is similar to the form of formula (3.3).

Suppose y_1 and y_2 are the results of the program being tested computed in two different rounding modes, we propose the formula

$$f(x) = \frac{\|y_1 - y_2\|}{\|y_1\| + \|y_2\|} \tag{3.4}$$

as a form of our general objective function. Here $\|y_1\| + \|y_2\|$ is the scaling factor.

Since there are four rounding modes specified by the IEEE 754 standard, which two rounding modes are used to compute the results y_1 and y_2? In the example in section 2.2, the difference between the results computed in rounding up and rounding to the nearest is zero. If we take the results computed in rounding up and rounding to nearest as y_1 and y_2, no instability will be revealed. The same thing happens if we pick the results computed in rounding down and rounding to zero as y_1 and y_2, since their results are both 0. If we take the results computed in rounding up and rounding down as y_1 and y_2, the measurement's value is

$$\frac{\|12345680.0 - 0.0\|}{\|12345680.0\| + \|0.0\|} = 1. \tag{3.5}$$

Huge instability is detected.

This example shows the importance of picking the right rounding modes. We should always pick the rounding modes that can reveal the largest differences. We

propose

$$f(x) = \max_{y_1, y_2 \in \{y_{near}, y_{up}, y_{down}, y_{zero}\}} \frac{\|y_1 - y_2\|}{\|y_1\| + \|y_2\|}, \tag{3.6}$$

where $y_{near}, y_{up}, y_{down}, y_{zero}$ are the results computed in four different rounding modes, as our objective function.

3.4 A Justification of the Scheme

As described previously, our method for testing stability is based on Kahan's testing method. Regarding his method, Kahan commented [3, p. 58]: "Of course this scheme can't be foolproof since error-analysis can't be automated in general."

Thus, in theory, our method is not a stability analysis. In practice, however, it often reveals stability in the following sense. Changing rounding modes can be viewed as introducing small perturbation into the data. By measuring the maximal difference between the results computed in different rounding modes, we know the sensitivity of the program to the perturbation of data. When the measurement (3.6) is close to 1, the program is very sensitive to the perturbation of data, when the measurement (3.6) is close to 0, the program is insensitive to the perturbation of data.

However, we must point out that the measurement (3.6) includes both the sensitivity of the underlying problem and the backward stability of the algorithm. A program solves a problem and implements an algorithm. A program solves a problem and implements an algorithm. The measurement (3.6) is about the behavior of the program, which involves both the problem and the algorithm. The measurement (3.6) is close to 1 when either the problem is ill-conditioned or the algorithm is unstable on both. The measurement (3.6) is close to 0 when the problem is well-conditioned

and the algorithm is stable. Thus a small value of the measurement (3.6) gives us confidence of the program. A large value of the measurement (3.6) gives us a warning. Further analysis is required.

3.5 Summary and Discussing

In this chapter we have described some fundamental aspects of our new numerical stability testing method, including the goal, the main step, why we need a general objective function and how to obtain it. We also gave a justification of our method.

At the beginning of this chapter we presented the goal of our method we expect to reach. It contains the following features:

- It provides an objective function suitable for general problems.
- The objective function is simple, easy to understand.
- The objective function can reflect problem's stability.
- Appropriate input data for the objective function which shows problem's instability can be found automatically.

We can say that the first two features are reached with no doubt. The third and forth features are worth to be discussed here.

As mentioned in section 3.4, a precise stability analysis can't be automated in general, our method is no exception. However, as explained in section 3.3, our scheme reveals the stability behavior of the program being tested. In Chapter 5, we will present our experimental results to demonstrate that our method offers a practical way of testing the stability of a program.

The forth feature of our goal, which is its objective function's maximizer can be successfully located by direct search methods, is the main topic of the next chapter.

In the next chapter we will describe all the efforts we can do to get a better result.

Chapter 4

Implementation Details

In Chapter 3 we gave a general description of our stability testing method. However, there are several implementation details remain to be discussed. In section 4.1 we discuss some details of the direct search method, like the choosing of the direct search method, the initial simplex and stopping criteria. In section 4.2, we present the sample construction details.

4.1 Details of the Direct Search Method

Our objective function is lack of smoothness and we can't obtain its derivative, so we choose the direct search methods. There are numerous direct search methods. Higham suggested three direct search methods for his stability testing method. The three direct search methods Higham suggested are the alternating directions (AD) method, the multi-directional search (MDS) method and the Nelder-Mead direct search method. AD is the simplest direct search method, its performance is worse than the other two methods [2, p. 475]. Since the MDS method is an improved

method over the Nelder-Mead method, we consider to use the MDS method in our implementation. The MDS method is described in section 2.1.

Although we have decided to use the MDS method as our direct search method, there are several details remain to be decided, such as the initial simplex and stopping criteria. According to years of experience with direct search methods in history, there is a variety of choices of these factors. Here we present our choices, which is used in the practical experiment in Chapter 5.

4.1.1 The Initial Simplex

Refer to section 2.1.2, in the first step of the MDS method, we generate an initial simplex from a given single starting point. The initial simplex is very important since it affects all the following simplexes generated in every iteration. Shape and size of the initial simplex is the main concern here.

Shape

The shape of the initial simplex should be decided first. Many people have deep research on the initial simplex for direct search methods, for example Spendley, Hext, and Himsworth [15], Kowalik and Pizzo [12] [9], There is only one requirement needs to be satisfied for the initial simplex generation: the generated simplex should be nondegenrate. Here we choose to generate a right-angled initial simplex with the same length of each edge. Which means, the n point other than the initial point is determined by formula (4.1).

$$x_i = x_0 + \alpha e_i, \tag{4.1}$$

where x_0 is the initial guess point, α is the initial edge length, and e_i is the unit basis vectors for each dimension.

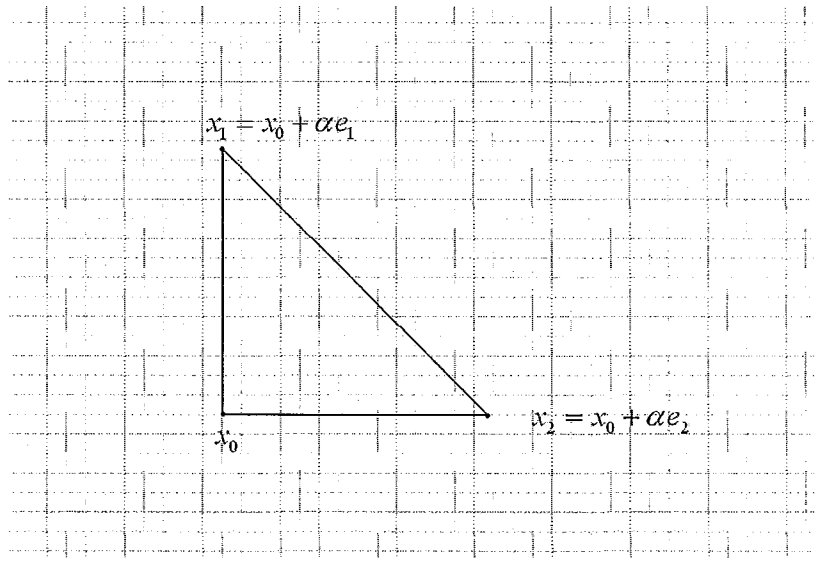

Figure 4.1: The right-angled initial simplex with the same length of each edge with dimension two

Size

The size of the initial simplex is another issue to be determined. Note that if the initial size is too small, the extension of the search process will be very slow, which leads to the situation that the search algorithm can only search a small local area. And if the initial size is too large, it takes much more time for the algorithm to shrink the simplex to a reasonable size [13]. In other words, the search method may spend too many iterations to contract the simplex before it can make any real progress. Since $2n$ function evaluations needed for contracting the simplex in each iteration, a large initial simplex makes the searching process costly.

But comparing both the disadvantage of small size and large size, the disadvantage of large size is less important due to the improvement of the computing speed of the computer by now. So we prefer to choose large size of the initial simplex in our implementation.

4.1.2 The Stopping Criteria

Here we give a discussion of how to terminate the searching process. Torczon introduced three stopping criteria in her paper [13].

The first stopping criteria is suggested by Nelder and Mead [8]. Searching stops when the standard deviation of the function values in the simplex falls below a preset value. The stopping test is as below:

$$\sqrt{\Sigma \frac{(f(v_i^k - \overline{f}))^2}{n}} < \varepsilon, \tag{4.2}$$

where v_i^k is the function value of the ith vertex in the kth iteration, $\overline{f}$ is the mean of the function values at the $n+1$ vertices, ε is the preset tolerance.

The second stopping criteria is suggested by Parkinson and Hutchinson [9]. Searching stops when both the range in f and the corrections to v_i for all i fall below a preset value. That means the following two tests should be both satisfied:

$$f(v_n^k) - f(v_0^k) < \varepsilon, \tag{4.3}$$

$$\frac{1}{n}\Sigma \|v_i^{k+1} - v_i^k\|^2 < \varepsilon, \tag{4.4}$$

where v_n^k is the vertex with the smallest function value in the simplex in the kth

iteration.

This scheme is not suitable for our objective function since it uses the function value as one of the stopping criterion, and the function value of our general objective function changes dramatically in unstable areas.

The third stopping criteria is suggested by Woods [16]. He slightly changed the second stopping criteria and made it to a stopping criterion measuring the relative size of the simplex by considering the length of the longest edge adjacent to v_0^k:

$$\frac{1}{\Delta} \max_{1 \leq i \leq n} \|v_i^k - v_0^k\| \leq \varepsilon, \tag{4.5}$$

where $\Delta = max(1, \|v_0^k\|)$.

For most optimization methods, people often choose function value as the measure of stopping criteria. But for simplex search methods, we have another measurement, which is the edges' length. It's easy to notice that the measurement of the first stopping criteria is the function value. The measurement of the third stopping criteria is the edges' length. The second stopping criteria's measurement is both the function value and the edges' length.

Normally, choosing function value as the stopping criteria should be the first choice to be considered, since this kind of stopping criteria is obvious. One disadvantage of choosing function value as the stopping criteria is that it may lead to premature termination when the simplex becomes too small relative to the curvature of the surface before the final minimum is reached [13].

For our objective function, most of the points are stable points. This means that the values of the objective function at those stable points are small and smooth. So it leads to premature termination.

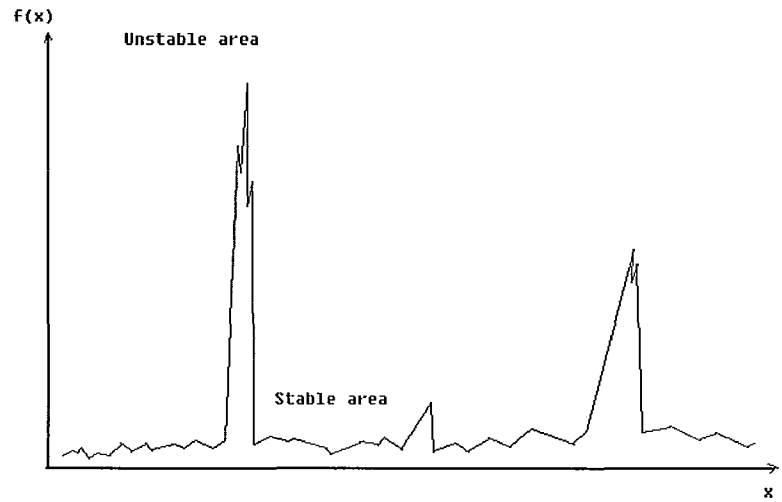

Figure 4.2: The general behavior of our objective function with dimension one

Even if the searching process reaches those unstable area where the function values are large, the stopping criteria by checking function value still has its disadvantage. The function value changes dramatically due to the unstable behavior of the program to be tested at those unstable area. So if the preset ε is small, the searching is hard to converge.

In summary, if the preset ε is large, it leads to premature termination of the search algorithm at those stable area. If the preset ε is small, the search algorithm is hard to converge due to the large differences between the values in neighborhood points at the unstable area.

So the stopping criteria measure using function value is not suitable for our objective function. Since the first stopping criteria and the second stopping criteria both use function value as the measure, they are not suitable for our objective function. Therefore, we choose formula (4.5) as our stopping criteria.

Now we have not determined the value of ε yet. The value of ε affects the number

of function evaluations, and we have to determine it in practical experiment. So we leave the decision of the value of ε to Chapter 5.

4.1.3 Other Factors

Other than the initial simplex and stopping criteria, there are several factors remain to be discussed.

The Orientation of the Initial Simplex

The orientation of the initial simplex determines the search direction, which also has effects on the search process. But since the effects of the orientation is hard to know or test, we just use the default orientation here.

The Scaling Factors

The scaling factors include the expansion factor μ, which determines the expansion speed, and the shrinking factor θ, which determines the shrinking speed. The effects of those two factors are also hard to know or test. So we just set $\mu = 2$ and $\theta = \frac{1}{2}$, which are the common choices for the direct search methods [13].

Maximum Function Evaluations

To ensure the termination of the searching process, a maximum function evaluations is necessary for our program. We never know how many function evaluations needed to reach the preset stopping criteria for a specific problem before practicing. Moreover, due to the unpredictable behavior of the objective function in the neighborhood of

converging point, some times the amount of function evaluations needed is far more than expected. So we set a maximum function evaluations as a extra stopping criteria. The searching stops while the amount of function evaluations reaches the maximum function evaluations, or formula (4.5) is satisfied. Choosing the maximum function evaluations would be a personal choice, it depends on how much time the user's computer's calculation costs for one function call and how long the user can bear for a searching process.

4.2 Sample Construction

Can the search method guarantee to find the most unstable point in the entire range space? If no, how to improve our strategy to gurantee a convincing result? In this section we will discribe how to improve the accuracy of our method by sample construction.

4.2.1 Limitations of the MDS Method

The direct search methods have those main disadvantages below:

• There are infinity points inside the input range. Since the direct searching can only cover a finite number of points, we can only say that the "maximum" point found by the direct search method is the maximum point among the points the searching tried, not the real maximum point of the function.

• The search direction can be sensitive to the initial point and initial simplex. Even though two searching process start at two neighboring points, the search results, both the location of maximum point and the function value at the maximum point could

be very different.

- Although the MDS method assures its vertices span the full space, the direct search method can only find a local maximum point near the initial point. That means the "most unstable point" found by the direct search method is usually just a most unstable point in a small area near the initial point, not the most unstable point in the full space.

The above limitations of direct search method showes above indicate that we can't rely on one searching process. We have to repeat the searching process for several times, and get a final result by analyzing the whole sample.

4.2.2 Details of Sample Construction

A simple way to generate initial input data is using random data. The size of the sample depends on the time cost for one searching process and the whole time the user can bear. Due to the low converging speed for the MDS method, one searching process usually needs to compute the algorithm thousands of times, which is really costly. So large size of sample may not be suitable.

Just using random initial data also has its disadvantage. Since we can't control the generation of the random initial data, sometimes most random initial data concentrated to a small area, and makes just few initial data in other place. This kind of distribution of initial data obviously fails to solve the problem of "local maximizer", and makes our results less accurate.

To avoid this disadvantage of random initial data, it's reasonable to use another sample construction strategy, which is dividing the full space into many small areas, and generate initial data in each of the small areas. This strategy can avoid the

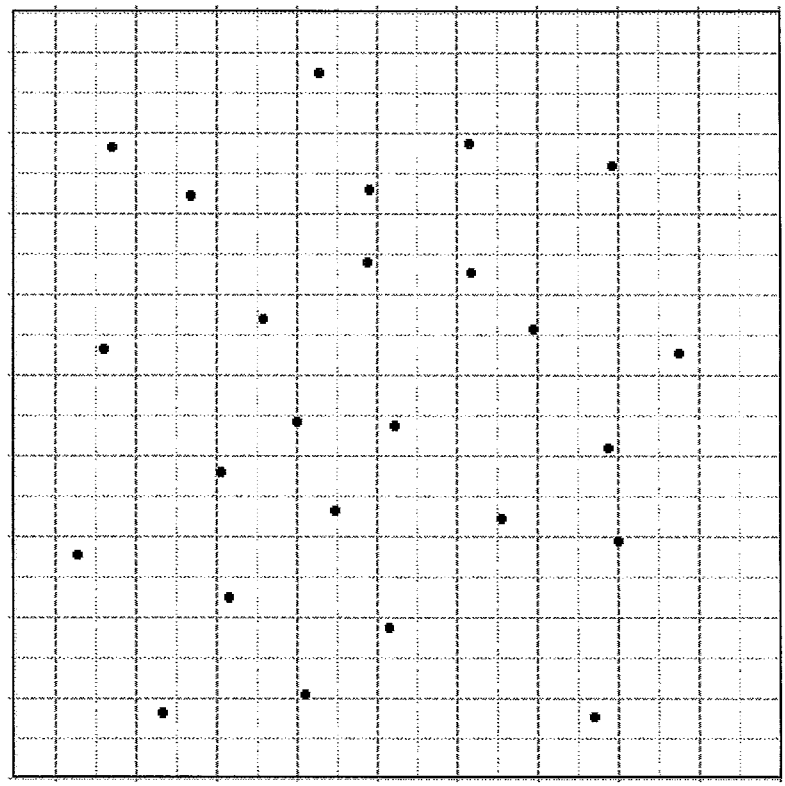

Figure 4.3: Good distribution of the input random data

disadvantage that the initial points are unevenly distributed. But this strategy also has its own disadvantage. Comparing to random initial data, generate initial data in this way makes our implementation much more complicated, especially when the dimension is large.

Actually, there is a simple method to reduce the effects of "local maximizer" for random inputs. If we set the initial step length to a large size, the effects of the location of initial points could be reduced. For example, if we set the initial size to 1/4 of the full space's size, the points could be largely span over the full space after one iteration.

As mentioned before, if the initial size is too large, the searching algorithm may

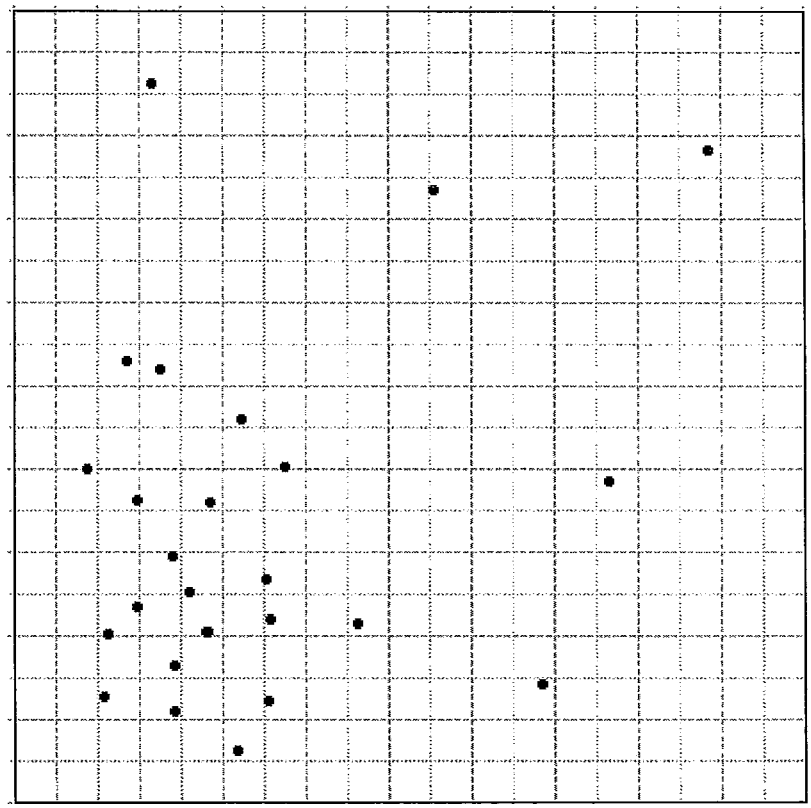

Figure 4.4: bad distribution of the input random data

spend more time to converge, since more iterations needed to contract the step length to a reasonable size. But since the time cost is not the first concern, here we consider using random initial data with large size of initial simplex as our strategy in our implementation.

4.3 Summary and Discussion

As mentioned in Chapter 3, our stability testing method contains two steps:

- Obtain an objective function using the Kahan's method.
- Use direct search method to find a maximizer of the objective function.

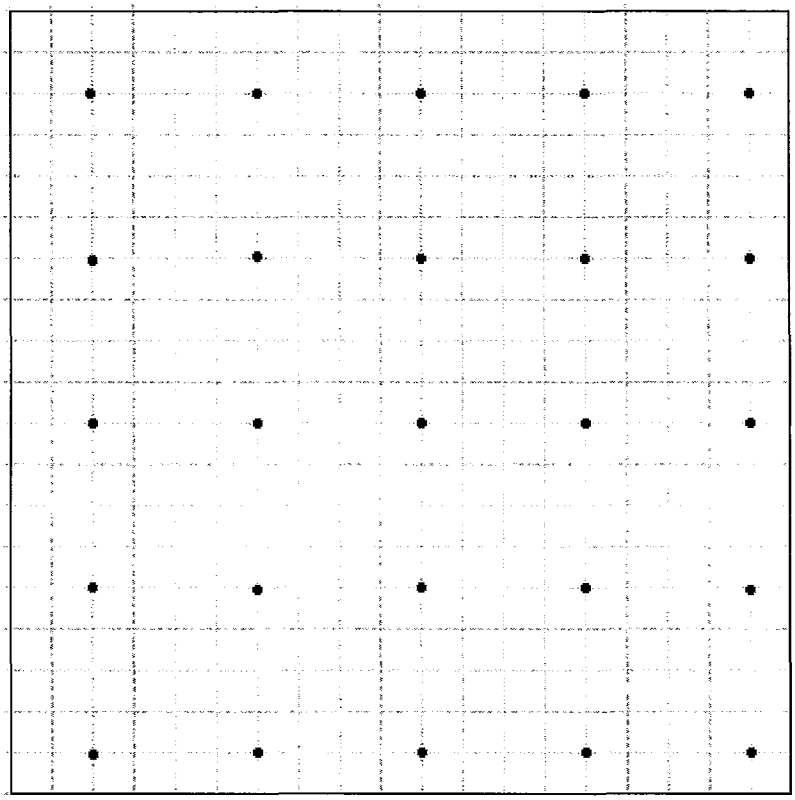

Figure 4.5: A improved strategy to generate input data

In Chapter 3 we concentrated on the construction of the objective function, which is the first step. In this chapter we concentrated on the second step, discussed the details of applying the direct search method. Comparing with Chapter 3, the content of this chapter is more practical and complicated.

In our implementation we have chosen the MDS method. There is no restriction for the choosing of the direct search method. Choosing the MDS method here is just because it's one of a mature direct search method with nice performance, and it's suitable for the situation of our method. Of course, there are many good direct search methods to apply. In this chapter we have discussed the implementation details of applying the MDS method, to do our best to make the searching process

more accurate.

Although the direct search methods have been developed for several decades, their performance are still far from being perfect. Since there is no derivative information we can get from our objective function, we have to choose the direct search method for our implementation. We can image that due to the behavior of our objective function, it's nearly impossible for the direct search method to find the real maximum point of the objective function. To guarantee the accuracy of our stability testing method, we have improved our strategy by combining the results from multiple searches instead of one searching process. The new strategy can make our method more convincing.

Chapter 5

Performance

In this chapter we test the performance of our method through some practical experiments. We first present some preliminaries we need in section 5.1. These preliminaries include the measurement of stability, the purpose of our tests and how to choose the test problem. In section 5.2 we present the test results with comments. The conclusion is shown in section 5.3.

5.1 Preliminaries

Before we start our tests, there are several topics remain to be discussed here.

5.1.1 The Measurement of stability

As mentioned before, the objective function value f_{obj} is affected by both the instability of the test algorithm and the condition number of the problem. It's reasonable

to measure algorithm's stability with the formula

$$\frac{f_{obj}(x)}{cond(x)}, \tag{5.1}$$

where $cond(x)$ is the condition number of the problem at the point of x.

The measurement f_{obj} is not as accurate as $\frac{f_{obj}(x)}{cond(x)}$, but it has two advantages which the measurement $\frac{f_{obj}(x)}{cond(x)}$ does not have. Firstly, the measurement $\frac{f_{obj}(x)}{cond(x)}$ is not suitable for the problem whose condition number is unavailable. In this case, we can only use f_{obj} to measure stability. Secondly, we can get a quantitative impression of stability from the measurement f_{obj}, since the range of f_{obj} is $[0, 1]$. The quantitative impression is unavailable from the measurement $\frac{f_{obj}(x)}{cond(x)}$, because there is no upper bound for $\frac{f_{obj}(x)}{cond(x)}$. So f_{obj} is worth to be considered as one measurement even when the condition number is known. When f_{obj} is small, we can conclude that the algorithm is stable. But since f_{obj} is affected by both algorithm's stability and problem's condition number, when f_{obj} is large, we can't conclude the algorithm is unstable.

Here we introduce another measurement. We can measure stability by checking how many well-conditioned points with large f_{obj} can be found by the search method in a sample. If the search method can't find any or can only find few, it gives us confidence about the stability of the algorithm in well-conditioned case.

Above all, we use three measurements in our tests to measure algorithm's stability. These three measurements are:

- $\frac{f_{obj}(x)}{cond(x)}$:

This value is the best measurement of stability because it is a mixture of both the objective function value and the condition number of the problem.

- f_{obj}:

Due to lacking the consideration of the condition number, this measurement is not as accurate as the first one. But this measurement can give an quantitative impression of the stability since the range of f_{obj} is $[0, 1]$.

- The number of unstable well-conditioned points detected:

The measurement can give an impression about the algorithm's stability in well-conditioned case.

5.1.2 Purpose of the Tests

We will apply our stability testing method to measure the stability of some given algorithms whose stability is already known. These algorithms include both numerically stable algorithms and numerically unstable algorithms. The task of our tests is to check whether our measurements can accurately measure the given algorithm's stability.

5.1.3 The Test algorithms

We intend to choose our test algorithms with three concerns. First, the test algorithms should be the classic algorithms. Second, the test algorithms' numerical stability should be already known. Third, For a problem it's better to test more than one algorithms, so that we can compare their behaviors. Here we choose two sets of algorithms for our tests. They are:

- Gaussian elimination
- Cubic root finding algorithms

Here we give a simple introduction of the test algorithms.

5.1.4 Gaussian Elimination

The first set of algorithms being tested is Gaussian elimination. They are classic algorithms of scientific computing. We test the three kinds of algorithms:

- Gaussian elimination without pivoting. This algorithm is the most unstable Gaussian elimination.
- Gaussian elimination with partial pivoting. This algorithm is more stable than Gaussian elimination without pivoting.
- Gaussian elimination with complete pivoting. This algorithm is the most stable algorithm among the three.

In general, GE without pivoting is an unstable algorithm. GE with partial pivoting is also an unstable algorithm, but it is stable in practical. GE with complete pivoting is a stable algorithm.

5.1.5 Cubic Root Finding Algorithms

In section 2.1 we introduced one cubic root finding algorithm. To compare with the original algorithm, we pick a better cubic root finding algorithm [2, p. 479]. It modifies the formula

$$\omega^3 = -\frac{q}{2} \pm \sqrt{\frac{q^2}{4} + \frac{p^3}{27}} \tag{5.2}$$

to

$$\omega^3 = -\frac{q}{2} - sign(q)\sqrt{\frac{q^2}{4} + \frac{p^3}{27}}. \tag{5.3}$$

The new algorithm can avoid possible catastrophic cancellation while $\frac{p^3}{27}$ is much smaller than $\frac{q^2}{4}$. But although the second algorithm is better than the first one, both the two algorithms are unstable algorithms [2, p. 479].

5.1.6 Set ε in the Stopping Criteria

Now we have to determine the stopping criteria ε. In the last chapter we have decided to use the stopping criterion which measures the relative size of the simplex by considering the length of the longest edge adjacent to v_0^k:

$$\frac{1}{\Delta} \max_{1 \leq i \leq n} \|v_i^k - v_0^k\| \leq \varepsilon, \tag{5.4}$$

where $\Delta = max(1, \|v_0^k\|)$.

The value of ε remains to be decided. Since the value of ε affects the total number of function evaluation, to limit the time cost for each searching process, we choose to find a suitable value of ε which makes the number of function evaluation be less than 1000 in most case. Because according to our previous experience, for our objective function, the results of the direct search method does not change too much when the function evaluation is more than 1000. After testing, we found that $\varepsilon = 10^{-4}$ is suitable for our choice. So we choose $\varepsilon = 10^{-4}$ in our tests.

5.2 Results

In this section we present the results of our method for each algorithm. For every algorithm we obtain ten points found by the search method. First we give a discussion for each set of algorithms separately. Then we give a conclusion by considering all the results for all the algorithms in the next section.

5.2.1 Gaussian Elimination

In the tests, the range of the inputs is from -10000 to 10000, all the test problem is in dimension four, and the initial simplex's size is 5000. Since we only consider the condition number of the matrix A, we set the vector b to $[1\ 1\ 1\ 1]^T$. The program is computed in double precision. We expect the result will show that the GE without pivoting is the most unstable algorithm, while GE with complete pivoting is the most stable one.

The ten points (matrix A) found by the search method for GE without pivoting are:

$$A_1 = \begin{pmatrix} 1.43999 & -9848.01 & -6431.8 & 19.3328 \\ -4995.1 & -8486.75 & 8.35652 & -7.39266 \\ -1006.13 & -1.01958 & 1018.65 & -2342.12 \\ -1.80602 & 297.735 & -90.8728 & -6815.39 \end{pmatrix}, A_2 = \begin{pmatrix} 1.95875 & 7526.61 & 541.21 & -4792.73 \\ -9226.96 & 6.45597 & 2680.29 & 1054.78 \\ 395.481 & -21.0101 & -27.8136 & 9.56359 \\ -5312.92 & -3567.13 & -3999.7 & 399.727 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} -9925.36 & -4651.89 & 54.9596 & -1.98713 \\ -38.1049 & -17.8595 & -4819.22 & 9987.84 \\ -76.4282 & 9885.65 & 6245.47 & -7499.98 \\ -3305.77 & -9566.81 & -2800.58 & -550.717 \end{pmatrix}, A_4 = \begin{pmatrix} 9.08664 & -1083.41 & -14.6202 & 8473.08 \\ 7646.02 & -983.76 & -2829.83 & -1444.81 \\ -168.939 & -7627.09 & -17.0357 & -49.3958 \\ 2595.83 & -73.9318 & -5674.05 & -34.8452 \end{pmatrix},$$

$$A_5 = \begin{pmatrix} 0.599255 & -2370.55 & 1.42625 & -2630.88 \\ -8688.97 & -3688.86 & 13.6895 & 4.50765 \\ 1.51462 & -3639.33 & 8438.54 & -45.4359 \\ -7503.5 & -91.5346 & 624.187 & 3728.13 \end{pmatrix}, A_6 = \begin{pmatrix} -1.47678 & -7283.16 & 8918.5 & -573.311 \\ -8752.93 & 217.936 & -16.177 & -17.8294 \\ 618.277 & 1.87534 & 1.17806 & 5960.86 \\ -5030.52 & 115.853 & 1.34672 & -254.801 \end{pmatrix},$$

$$A_7 = \begin{pmatrix} -1447.5 & -9997.91 & -154.841 & 9983.1 \\ -7501.24 & 3221.38 & 4965.52 & -4998.99 \\ -704.078 & 24.9819 & 368.644 & -35.3597 \\ 9680.19 & -3511.16 & 2.73312 & -7938.65 \end{pmatrix}, A_8 = \begin{pmatrix} -18.4722 & -3.8317 & 4168.19 & 1151.46 \\ -18.2657 & -5.79542 & 4.76832 & -10.9389 \\ -91.5908 & 62.1426 & 1922.71 & -397.71 \\ 5.24062 & -351.442 & -8367.68 & 1042.4 \end{pmatrix},$$

Point	Condition Number	f_{obj}	$\frac{f_{obj}(x)}{cond(x)}$	Condition
A_1	9.2669e+007	1.44064e-005	1.5546e-013	ill-conditioned
A_2	5.2626e+007	0.00340149	6.4635e-011	ill-conditioned
A_3	32.6749	1.17527e-007	3.5969e-009	well-conditioned
A_4	2.6293	4.62064e-007	1.7574e-007	well-conditioned
A_5	1.1943e+007	0.000121288	1.0156e-011	ill-conditioned
A_6	1.4646e+008	0.475207	3.2446e-009	ill-conditioned
A_7	1.7005e+007	4.78212e-007	2.8122e-014	ill-conditioned
A_8	2.6905e+008	2.83584e-006	1.0540e-014	ill-conditioned
A_9	1.1766e+007	0.000187658	1.5949e-011	ill-conditioned
A_{10}	8.7517e+009	9.56064e-006	1.0924e-015	ill-conditioned

Table 5.1: Results for GE without pivoting

$$A_9 = \begin{pmatrix} 807.423 & -9929.16 & -9977.92 & -6222.87 \\ -5.48785 & 116.375 & 9143.78 & 8949.95 \\ 8980.77 & 4707.62 & 15.2584 & 5008.83 \\ -7622.55 & -4854.95 & 8081.05 & 3966.22 \end{pmatrix}, A_{10} = \begin{pmatrix} 3.14731 & -5.39911 & -9753.51 & -6.30177 \\ -1623.4 & 1.99721 & 9170.67 & 20.5083 \\ 3194.33 & 2.92222 & -6604.18 & -254.229 \\ 407.212 & -10.2771 & 562.324 & 4906.66 \end{pmatrix}.$$

The ten points (matrix A) found by the search method for GE with partial pivoting are:

$$A_1 = \begin{pmatrix} -99.7053 & 1.51037 & -146.294 & -9.62291 \\ -1323.73 & 1793.47 & 9600.51 & -1392.87 \\ 367.816 & -10.4284 & -2.53052 & -5620.15 \\ -29.9225 & -1.54993 & -620.423 & -6246.63 \end{pmatrix}, A_2 = \begin{pmatrix} -938.124 & 6570.85 & -9600.72 & -9994.56 \\ 304.335 & 93.6211 & -173.95 & -51.1581 \\ -892.087 & -2.59536 & -1.17608 & -131.548 \\ -145.249 & 308.99 & 7501.35 & -9272.59 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} 4229.87 & -4033.67 & 109.766 & -2406.75 \\ -11.6105 & -1.93314 & 3.55366 & 83.9949 \\ -114.043 & 123.104 & -6.69624 & 8.62863 \\ 6915.74 & -2595.49 & -583.097 & -3901.81 \end{pmatrix}, A_4 = \begin{pmatrix} -9960.63 & 9998.38 & 8260.38 & 141.204 \\ 1.56306 & -79.0914 & 1.60491 & -27.2871 \\ 10.5345 & 1685.17 & 7865.32 & 1573.09 \\ -2032.45 & 1906.91 & -9997.83 & -1456.48 \end{pmatrix},$$

$$A_5 = \begin{pmatrix} -9619.18 & 1.64325 & -41.5976 & -2.1938 \\ -490.854 & -825.496 & 1353.83 & -5370.23 \\ -1247.14 & 20.3704 & -133.936 & -126.614 \\ -3.93872 & 89.1523 & -204.752 & 422.605 \end{pmatrix}, A_6 = \begin{pmatrix} -11.1751 & 1.35889 & -2580.63 & 5050.82 \\ -26.2784 & 175.719 & -1565.95 & 9.07643 \\ -2.01186 & 62.81 & -2794.29 & 636.715 \\ 19.1812 & -8.93468 & -3753.36 & -65.5517 \end{pmatrix},$$

Point	Condition Number	f_{obj}	$\frac{f_{obj}(x)}{cond(x)}$	Condition
A_1	3.6436e+008	9.50351e-005	2.6083e-013	ill-conditioned
A_2	1.4004e+009	3.01207e-007	2.1509e-016	ill-conditioned
A_3	5.6930e+008	1.62221e-007	2.8495e-016	ill-conditioned
A_4	8.7583e+007	8.43754e-009	9.6338e-017	ill-conditioned
A_5	1.0902e+008	2.47832e-006	2.2734e-014	ill-conditioned
A_6	1.7771e+008	6.59396e-010	3.7105e-018	ill-conditioned
A_7	3.0708e+008	6.59396e-010	4.6648e-015	ill-conditioned
A_8	4.5766e+007	6.59396e-010	1.1264e-014	ill-conditioned
A_9	5.1927e+008	5.45116e-007	1.0498e-015	ill-conditioned
A_{10}	8.3726e+008	5.70857e-007	6.8182e-016	ill-conditioned

Table 5.2: Results for GE with partial pivoting

$$A_7 = \begin{pmatrix} -22.5396 & -142.72 & -4.27564 & -16.5263 \\ -4544.75 & 3185.07 & -4.91253 & -2.24621 \\ -18.0614 & 73.1052 & 11.2191 & 3.65291 \\ 464.202 & -13.728 & -4815.66 & 1354.99 \end{pmatrix}, A_8 = \begin{pmatrix} -9975.67 & -140.485 & 371.477 & 6253.51 \\ -2520.69 & -1005.97 & -766.026 & -858.549 \\ -3166.04 & 371.147 & -538.774 & 149.546 \\ -1.43424 & 7541.52 & -68.6056 & -14.7109 \end{pmatrix},$$

$$A_9 = \begin{pmatrix} 146.048 & -9998.94 & -8216.81 & -4903.87 \\ -8836.65 & -2496.01 & -99.7053 & -644.884 \\ -7501.52 & -863.469 & 943.827 & -4.17821 \\ 4532.48 & 699.738 & -324.113 & -8451.87 \end{pmatrix}, A_{10} = \begin{pmatrix} -2458.61 & 6718.19 & 608.335 & -3.77823 \\ 2.69195 & 808.961 & -1.41422 & 3.60193 \\ -135.602 & 7225.94 & 17.9501 & 40.5813 \\ -4917.22 & -1317.27 & -7526.97 & 9865.12 \end{pmatrix}.$$

The ten points (matrix A) found by the search method for GE with complete pivoting are:

$$A_1 = \begin{pmatrix} -609.704 & -5.74352 & 3239.54 & 312.3 \\ -244.419 & -257.103 & 11.1124 & 1115.31 \\ -3886.75 & -5057.1 & 7.48887 & -273.239 \\ -11.1062 & -41.1788 & 1614.97 & -7960.14 \end{pmatrix}, A_2 = \begin{pmatrix} 524.725 & 7.05961 & -23.3653 & 9956.76 \\ -8568.68 & -7.56504 & -527.683 & -374.492 \\ 5074.01 & -5588.46 & -66.2184 & 2.57212 \\ 7.1918 & -9993.97 & -678.434 & -2.0985 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} -9995.3 & 31.071 & -7.97749 & 621.229 \\ -8973.17 & -510.261 & 3345.03 & 4.7182 \\ -107.93 & -72.085 & -24.2077 & -1.23226 \\ 99.0242 & -284.162 & 8.48909 & -51.6784 \end{pmatrix}, A_4 = \begin{pmatrix} -10.231 & 4.67542 & 79.3586 & -9688.74 \\ -9607.44 & -3578.99 & -1120.18 & -3.22524 \\ -1188.29 & -117.36 & 37.4676 & -7129.26 \\ 3043.61 & -11.0067 & -88.9021 & 3112.39 \end{pmatrix},$$

Point	Condition Number	f_{obj}	$\frac{f_{obj}(x)}{cond(x)}$	Condition
A_1	2.4421e+007	5.46753e-008	2.2389e-015	ill-conditioned
A_2	9.8396e+007	7.49831e-008	7.6206e-016	ill-conditioned
A_3	9.0631e+008	6.04709e-008	6.6722e-017	ill-conditioned
A_4	3.3833e+008	1.34554e-007	3.9770e-016	ill-conditioned
A_5	6.6716e+007	5.43878e-009	8.1521e-017	ill-conditioned
A_6	1.3873e+007	1.33657e-007	9.6346e-015	ill-conditioned
A_7	6.8271e+007	8.822e-010	1.2922e-017	ill-conditioned
A_8	7.3204e+007	4.02706e-008	5.5012e-016	ill-conditioned
A_9	4.9773e+008	2.67991e-007	5.3843e-016	ill-conditioned
A_{10}	3.3261e+009	3.60004e-008	1.0824e-017	ill-conditioned

Table 5.3: Results for GE with complete pivoting

	GEWP	GEPP	GECP
Average $\frac{f_{obj}(x)}{cond(x)}$	9.13303e-8	1.415445e-13	7.10985e-15
Average f_{obj}	0.0478945	1.0104962e-5	8.08922e-8
Well-Conditioned Points	2	0	0

Table 5.4: The Three Measurements For the Three GE Algorithm

$$A_5 = \begin{pmatrix} 7.42736 & 8830.95 & 8.38162 & -3481.45 \\ 2759.43 & 7511.51 & 307.776 & -9220.07 \\ -1643.69 & -76.3867 & 2931.15 & 131.252 \\ -2.74544 & -1.34747 & 167.614 & -189.468 \end{pmatrix}, A_6 = \begin{pmatrix} -9996.37 & 9996.33 & 2867.5 & -3.7761 \\ 30.8274 & -8306.69 & -1886.31 & -3097.07 \\ 1215.32 & 21.3915 & 24.7443 & 457.21 \\ -2006.79 & 3269.52 & 7813.02 & 17.8194 \end{pmatrix},$$

$$A_7 = \begin{pmatrix} -1254.18 & -3.90126 & 39.7459 & 4205.02 \\ -26.7554 & -58.6136 & -9.73171 & -271.513 \\ 522.37 & 214.17 & -176.71 & 204.522 \\ 7231.58 & -28.0965 & -7522.6 & -926.962 \end{pmatrix}, A_8 = \begin{pmatrix} -8816.58 & -1916.93 & -75.2316 & -1.13579 \\ -2826.31 & 3139.84 & -610.733 & 5136.7 \\ -2556.8 & -8017.7 & -443.788 & 4307.91 \\ 371.348 & -2371.27 & 18.8392 & 4.67279 \end{pmatrix},$$

$$A_9 = \begin{pmatrix} -2181.21 & 387.818 & 17.1703 & 10.9567 \\ -2542.3 & -9979.78 & -1569.55 & 1863.06 \\ 2.91074 & 40.5584 & 3.15085 & 92.4184 \\ -1.4685 & -423.795 & 30.1589 & -2987.62 \end{pmatrix}, A_{10} = \begin{pmatrix} -41.9053 & -50.7 & -257.827 & 1.40867 \\ -7487.42 & 45.6664 & 90.8217 & 300.763 \\ -9701.46 & 7.31001 & -35.6792 & -1457.13 \\ 914.943 & -2.64544 & 1.45127 & 2.5291 \end{pmatrix}.$$

Table 5.4 shows the results of the three measurements for the three kinds of Gaussian eliminations. The average value of $\frac{f_{obj}(x)}{cond(x)}$ for the ten points for each algorithm

shows that GE without pivoting has the largest $\frac{f_{obj}(x)}{cond(x)}$ and GE with complete pivoting has the smallest. It leads to the conclusion that GE without pivoting is the most unstable algorithm among the three, and GE with complete pivoting is the most stable one. This conclusion matches the expectation.

Like the result of $\frac{f_{obj}(x)}{cond(x)}$, the average value of f_{obj} for the ten points for each algorithm also shows that the GE without pivoting is the most unstable algorithm, and GE with complete pivoting is the most stable one. The value of f_{obj} for GE without pivoting is 0.0478945, it is very large and close to 1, the upper bound of f_{obj}. It means that GE without pivoting is an very unstable algorithm. On the other hand, the value of f_{obj} for the other two algorithms are 1.0104962e-5 and 8.08922e-8, they are much lower than the value of f_{obj} for GE without pivoting.

The last row in table 5.4 shows how many points are well-conditioned out of the ten points for each algorithm. The GE without pivoting has two points which are well-conditioned, it means that even the input data is well-conditioned, GE without pivoting still has chance to be unstable. There is no well-conditioned point out of the ten unstable points found by the search method for the other two algorithms. It means that while the input data is well-conditioned, these two algorithms are unlikely to be unstable.

Actually, there exist some well-conditioned points reveal unstable behaviors for GE with partial pivoting. For example, consider the matrix

$$A^{n\times n} = \begin{pmatrix} 1 & 0 & 0 & ... & 1 \\ -1 & 1 & 0 & ... & 1 \\ -1 & -1 & 1 & ... & 1 \\ ... & ... & ... & ... & ... \\ -1 & -1 & -1 & ... & 1 \end{pmatrix},$$

where $a_{ij} = 1$ when $i = j$ or $j = n$, $a_{ij} = -1$ when $i < j$, $a_{ij} = 0$ when $i > j$ and $j \neq n$.

This is a well-conditioned matrix which made to reveal the unstable behavior of GE with partial pivoting [2, p. 166]. The condition number of matrix $A^{20\times 20}$ is 8.8343 and the value of f_{obj} is 1.90615e-011. The condition number of matrix $A^{20\times 20}$ is rather small and f_{obj} is large, comparing to 10e-16 (double precision).

5.2.2 Cubic Root Finding Algorithms

In the tests, the range of the inputs is from -10000 to 10000. The initial simplex size is 5000. The program is computed in double precision. We expect the result will show that the improved algorithm is more stable than the original one, while both the two algorithms are unstable algorithms.

The ten points found by the search method for the original algorithm are:

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \begin{pmatrix} 15.7708 \\ -665.522 \\ -9749.30 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = \begin{pmatrix} 9999.95 \\ 174.385 \\ 0.760298 \end{pmatrix}, \begin{pmatrix} a_3 \\ b_3 \\ c_3 \end{pmatrix} = \begin{pmatrix} 13.6932 \\ -692.481 \\ -9999.85 \end{pmatrix}, \begin{pmatrix} a_4 \\ b_4 \\ c_4 \end{pmatrix} = \begin{pmatrix} 15.7604 \\ -673.917 \\ -9967.47 \end{pmatrix},$$

Point	f_{obj}	Roots (Computed by MATLAB)	Condition
1	0.0366355	[25.4442 − 27.0501 − 14.1649]	well-conditioned
2	1.0000000	[−9999.93 − 0.008719 − 0.008719]	ill-conditioned
3	0.0364738	[26.5582 − 25.4661 − 14.7853]	well-conditioned
4	0.0366649	[25.6540 − 27.0520 − 14.3624]	well-conditioned
5	0.027029	[−9.89349 0.000368 0.000368]	ill-conditioned
6	0.0296989	[43.3486 − 37.5949 − 6.13610]	well-conditioned
7	1.0000000	[−9999.64 0.0000002 0.0000002]	ill-conditioned
8	1.0000000	[−1.00000 0.0000035 0.0000035]	ill-conditioned
9	0.0369007	[25.3543 − 25.1323 − 16.1521]	well-conditioned
10	0.0373315	[24.4961 − 25.1323 − 16.1521]	well-conditioned

Table 5.5: Results for original cubic root finding algorithm

$$\begin{pmatrix} a_5 \\ b_5 \\ c_5 \end{pmatrix} = \begin{pmatrix} 9892.76 \\ -7291.66 \\ 1343.56 \end{pmatrix}, \begin{pmatrix} a_6 \\ b_6 \\ c_6 \end{pmatrix} = \begin{pmatrix} 0.382359 \\ -1664.99 \\ -9999.96 \end{pmatrix}, \begin{pmatrix} a_7 \\ b_7 \\ c_7 \end{pmatrix} = \begin{pmatrix} 9999.64 \\ -3.36823 \\ 0.000308915 \end{pmatrix}, \begin{pmatrix} a_8 \\ b_8 \\ c_8 \end{pmatrix} = \begin{pmatrix} 9999.96 \\ -692.495 \\ 11.9887 \end{pmatrix},$$

$$\begin{pmatrix} a_9 \\ b_9 \\ c_9 \end{pmatrix} = \begin{pmatrix} 15.5405 \\ -648.277 \\ -9852.45 \end{pmatrix}, \begin{pmatrix} a_{10} \\ b_{10} \\ c_{10} \end{pmatrix} = \begin{pmatrix} 16.7882 \\ -605.371 \\ -9944.01 \end{pmatrix}$$

The ten points found by the search method for the improved algorithm are:

$$\begin{pmatrix} a_1 \\ 2b_1 \\ c_1 \end{pmatrix} = \begin{pmatrix} 17.8584 \\ -549.602 \\ -9324.17 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = \begin{pmatrix} -6.88691 \\ -9987.12 \\ -9997.13 \end{pmatrix}, \begin{pmatrix} a_3 \\ b_3 \\ c_3 \end{pmatrix} = \begin{pmatrix} -6.14627 \\ -9968.88 \\ -9889.08 \end{pmatrix}, \begin{pmatrix} a_4 \\ b_4 \\ c_4 \end{pmatrix} = \begin{pmatrix} 6764.23 \\ -52.0384 \\ 0.0522849 \end{pmatrix},$$

$$\begin{pmatrix} a_5 \\ b_5 \\ c_5 \end{pmatrix} = \begin{pmatrix} 15.4116 \\ -544.820 \\ -8052.69 \end{pmatrix}, \begin{pmatrix} a_6 \\ b_6 \\ c_6 \end{pmatrix} = \begin{pmatrix} 15.5904 \\ -608.472 \\ -9251.93 \end{pmatrix}, \begin{pmatrix} a_7 \\ b_7 \\ c_7 \end{pmatrix} = \begin{pmatrix} 17.9942 \\ -586.059 \\ -9999.62 \end{pmatrix}, \begin{pmatrix} a_8 \\ b_8 \\ c_8 \end{pmatrix} = \begin{pmatrix} -9.90136 \\ -9996.23 \\ 9836.97 \end{pmatrix},$$

Point	f_{obj}	Roots (Computed by MATLAB)	Condition
1	0.0376142	[23.1871 −24.8883 −16.1572]	well-conditioned
2	0.0262475	[103.918 −96.0295 −1.00179]	well-conditioned
3	0.0262442	[103.442 −96.3031 −0.992700]	well-conditioned
4	0.00321615	[−6764.24 0.006504 0.001188]	ill-conditioned
5	0.0371456	[23.1495 −24.1680 −14.3931]	well-conditioned
6	0.0370372	[24.5485 −25.1589 −14.9800]	well-conditioned
7	0.0375311	[23.9381 −25.6412 −16.2912]	well-conditioned
8	0.0256946	[104.583 −95.6652 0.983205]	well-conditioned
9	0.0373503	[24.4138 −25.7199 −15.9254]	well-conditioned
10	0.0262483	[103.322 −95.4345 −0.999019]	well-conditioned

Table 5.6: Results for original cubic root finding algorithm

	Original Algorithm	Improved Algorithm
Average f_{obj}	0.324073	0.02993292
Well-Conditioned Points	6	9

Table 5.7: The two measurements for the cubic root finding algorithms

$$\begin{pmatrix} a_9 \\ b_9 \\ c_9 \end{pmatrix} = \begin{pmatrix} 17.2316 \\ -607.120 \\ -9999.95 \end{pmatrix}, \begin{pmatrix} a_{10} \\ b_{10} \\ c_{10} \end{pmatrix} = \begin{pmatrix} -6.88919 \\ -9868.43 \\ -9850.88 \end{pmatrix}$$

In the problem of cubic function, the condition number is unavailable. The problem is ill-conditioned if the function has multiple-roots or close roots. So the measurement $\frac{f_{obj}(x)}{cond(x)}$ can't be used in this problem.

According to table 5.7, the average value of $\frac{f_{obj}(x)}{cond(x)}$ is 0.324073 for the original algorithm, while it is 0.02993292 for the improved algorithm. From this result we can get the conclusion that the improved algorithm is more stable than the original algorithm, which matches the expectation. Both the two values are large, it leads to the conclusion that both the algorithms are unstable.

The number of unstable well-conditioned points found by the search method is 6

for the original algorithm, and 9 for the improved algorithm. Both the two numbers are very large, comparing to the number in the GE algorithms. It shows that these algorithms are both unstable while well-conditioned. The measurement does not show that the improved algorithm is more stable than the original one. One reason is that the stability of the two algorithms are very close. Another reason is that the sample is not large enough.

5.3 Conclusion

The goal of the tests is to show that whether our stability testing method can successfully reveal the stability of some algorithms which's stability is already known. Now we have tested two sets of algorithms, which shows that our method's performance have achieved the expectation. For numerically stable algorithms like GE with complete pivoting and GE with partial pivoting (which is stable in practical), our method can indicate that they are stable. For numerically unstable algorithms like GE without pivoting and the two cubic root finding algorithms, our method can indicate that they are unstable algorithms.

We have developed three measurements for our method to test algorithm's stability. Each measurement has both advantages and disadvantages. Considering all the three measurements together can make our method more convincing.

The performance of the third measurement, which measures the number of unstable well-conditioned points found by the search method, is not as accurate as the first two measurements. A large size of sample can increase the accuracy of this measurement. So we propose to use this measurement as only a suggestion.

Chapter 6

User Guide

In this chapter we present the user guide. Section 6.1 introduces the modules, including the module decomposition and the module hierarchy. User interface is explained in section 6.2. Section 6.3 gives an introduction of the important steps of the software for the users.

6.1 Module Guide

Five head files and five resource files are contained in the project. They are decomposed into four modules: the master control module, the search module, the objective function calculating module and the pipe module.

6.1.1 Module Introduction

A brief introduction of each module is shown below:

- The master control module

Module service: This module controls the execution of input, data initial, search process, and output.

Module secret: User interface.

Files: AutoStabilityTest.h, AutoStabilityTest.cpp

- The search module

Module service: This module performs the direct search method.

Module secret: The data structure and algorithms in the direct search method.

Files: Matrix.h, SearchMethod.h, MultiDirectional.cpp, NelderMead.cpp (Matrix.h declares the data structure of Matrix, which is used to apply the data of simplex in the direct search method)

- The objective function module

Module service: It calculates the value of the objective function.

Module secret: The algorithm to calculate the value of the objective function.

Files: Objective.h, Objective.cpp

- The pipe module

Module service: This module is the pipe between the software and the test problems.

Module secret: Pipe functions to write to and read from the test problems.

Files: Pipe.h, Pipe.cpp

6.1.2 Hierarchy

The relation between the four modules is quite simple. Their hierarchy is one dimensional. Each module uses and only uses the next module. The inherency of these modules are shown in figure 6.1:

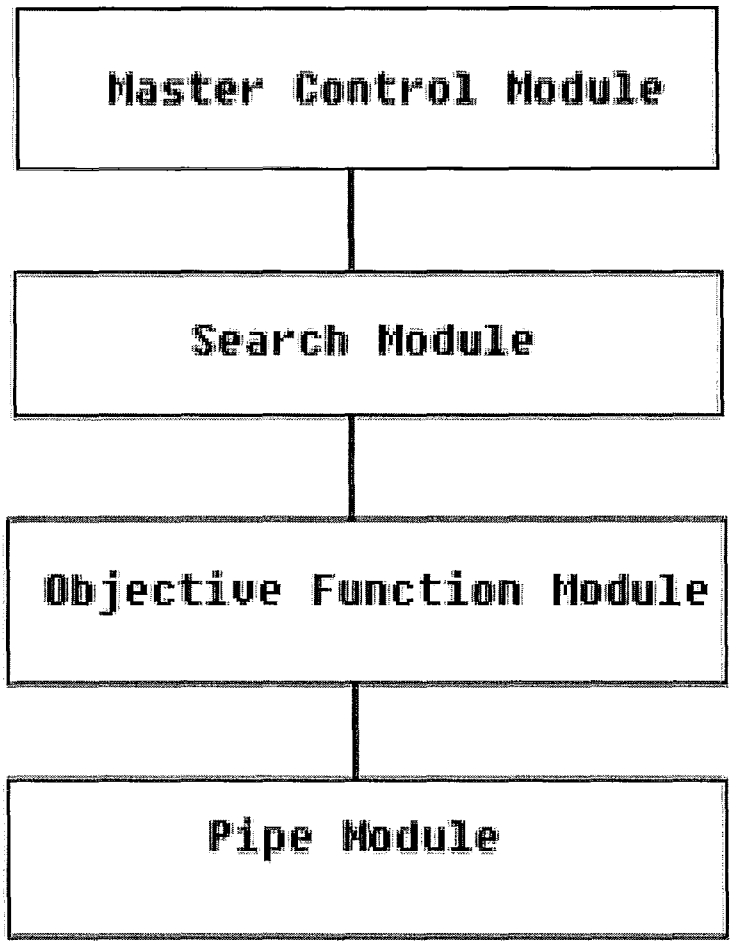

Figure 6.1: The module hierachy

6.2 User Interface

There are two output functions, they output the results to both screen and a txt file called "results" in the current directory. The txt file will be automatically generated if no such file exists.

The input of the project contains necessary parameters to apply the direct search method. Note that the stoppingStepLength is preset to 10^{-5} in SearchMethod.h and

does not need to be set by users. Users can change the size of stoppingStepLength through the head file. Another set of inputs is the command lines of the test problem. Four command lines are needed, signs to the test program in four different rounding modes. In Linux version the command lines are in the form of "./a.out" (for example), the default directory is the directory of the project.

The inputs and outputs are listed as following:

Inputs

- cmdLine1/2/3/4

  Type: string

  The command lines for the test programs in four rounding modes.

  - SampleSize

  type: int

  The size of the sample.

  - Dim

  Type: int

  The dimension of the problem.

  - RetDim

  Type: int

  The dimension of the outputs of the test program.

  - MaxCalls

  Type: long int

  The maximum function evaluations, which is introduced in section 4.1.3.

6.2.1 Outputs

- MaxTotal

  Type: double float

  The maximum value of $f_{obj}(x)$ found among the whole sample.

  - MaxTotalPt

  Type: vector(double)

  The point x in the maximum value of $f_{obj}(x)$.

  - MaxEachTrial

  Type: vector(vector(double))

  This vector stores the maximum $f_{obj}(x)$ found in each trial.

  - Avg

  Type: double float

  The average value of $f_{obj}(x)$ found among the whole sample.

6.3 Introduction of the Important Steps

Choose the Compiler

Before the test, please make sure that your compiler supports the C99 standard (like gcc, intel C++ compiler), since only the C99 standard defines the changing rounding modes functions.

Change Test Program's Rounding Mode

Since the father process can't change the child process's rounding mode, we have to create four executable files as the test program. Each executable file is with one

rounding mode out of the four.

The function fesetround (int rounding_mode) is used to set rounding modes. The value of "rounding_mode" could be FE_DOWNWARD, FE_TONEAREST, FE_UPWARD, FE_TOWARDZERO. They are the four different rounding modes - round downward, round to nearest, round toward zero and round upward. Four executable files need to be generated as the test executable files. To create the test executable files, add the function fesettround (FE_UPWARD)/fesetround (FE_DOWNWARD)/fesetround (FE_TONEAREST)/fesetround (FE_TOWARDZERO) to the top line of the test problem, then compile them separately to obtain four executable files. Remember to include the head file fenv.h. This head file declares the fesetround() function and defines those rounding mode control instants.

Modify the Interface of the Test Program

The interface of the test program needs to be modified to match the project. Since the read/write pipe functions can't distinguish useful and useless inputs/outputs, please be sure no useless inputs/outputs are contained in the program being tested. For example, if the test program's outputs is "$x_1 = 1; x_2 = 2; x_3 = 3$", it should be modified to "1 2 3". For the outputs of the test problem, the output precision needs to be modified to show the difference between the results with different rounding modes, because in most case the difference is tiny. Precision of 16 digits is suitable for double precision and precision of 8 digits is suitable for single precision.

Run the Program

Now we can run our program to do the stability tests. Remember to type the right directory of program being tested in the cmdLine. Choose proper sample size and set maximum function evaluations considering both the time and the accuracy we want. If the problem's condition number is available, obtaining the condition number at those unstable points is strongly advised.

Chapter 7

Conclusions

Automatic stability testing method is an attractive technique in scientific computing. In this thesis we present a new generic stability testing method, which combines an automatic stability testing method introduced in Higham's literature [2, p. 479] and a generic stability testing method introduced by Kahan [3, p. 56], and takes both of their advantages. For normal stability testing, Obtaining proper input data to reveal stability is a tough task. One advantage of our method is that it does not require proper input data, since it uses optimization method to automatically detect those unstable points and reveal the instability there. For those automatic stability testing methods, objective function whose maximal value can be located by optimization method is created for specify problems. Our method's objective function is generic since the construction of our objective function does not related to the problem being tested. Refer to Chapter 1, normal generic automatic stability methods are quite complicated to implement and use, while our method is quite simple.

A big problem for automatic stability testing methods is that their accuracy can not be guaranteed. We propose to use a sample instead of one searching result to

measure algorithm's stability. Moreover, we present three measurements to measure algorithm's stability. Considering all the three measurements makes our scheme more accurate.

However, although we don't have to know the information of the problem being tested while constructing the objective function, knowledge of the condition of the problem is required if we want to get a convincing conclusion. Our method still can test those problems which's condition number is unavailable since one of our measurement does not require the information condition number, but for those problems the accuracy of our method can not be guaranteed.

As error analysis cannot be automated in general, in Chapter 5 we use practical experiments to test the performance of our method. Although the tests shows that our method works well, we can not fully guarantee the accuracy for other problems. It's better to consider the result of our method as a beginning of error analysis, further analysis for the problem is strongly suggested.

Bibliography

[1] IEEE Standard for Binary Floating-Point Arithmetic. *SIGPLAN Notices 22:2*, 37:9–25, 1985.

[2] Nicholqs J. Higham. *Accuracy and Stability of Numerical Algorithms, Second Edition.* SIAM, Philadelphia, 2002.

[3] W. Kahan and Joseph D. Darcy. How Java's Floating-Point Hurts Everyone Everywhere. *ACM 1998 Workshop on Java for High-Performance Network Computing*, 1998.

[4] John l. Larson and Ahmed h. Sameh. Efficient calculation of the effects of roundoff errors. *ACM Trans. Math. Software*, pages 228–236, 1979.

[5] John l. Larson and Ahmed h. Sameh. Algorithms for roundoff error analysis - A relative error approach. *Computing*, pages 275–297, 1980.

[6] Philippe Langlois. Automatic linear correction of rounding errors. *BIT*, pages 415–539, 2001.

[7] Webb Miller. Software for roundoff analysis. *ACM Trans. Math. Software*, pages 108–128, 1975.

[8] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, pages 308–313, 1965.

[9] J. M. Parkinson and D. Hutchinson. An investigation into the dfficiency of variants on the simplex method. *Numerical Methods for Non-linear Optimization*, 37:115–135, 1972.

[10] M. J. D. Powell. A suirvey of numerical methods for unconstrained optimization. *SIAM Rev.*, 37:79–97, 1970.

[11] Thomas Harvey Rowan. Functional Stability Analysis of Numerical Algorithms. *PhD thesis, University of Texas at Austin*, 1990.

[12] J. S. Kowalik S. L. S. Jacoby and J. T. Pizzo. Iterative Methods for Nonlinear Optimization Problems. *Prentice Hall, Inc., Englewood Cliffs, New Jersey*, 37, 1972.

[13] Virginia Joanne Torczon. Multi-Directional Search: A Direct Search Algorithm for Parallel Machines. *Ph.D thesis, Rice University*, 1989.

[14] R. Vichnevetsky and J. Vignes. Marie-Christine Brunet and Francoise Chatelin. CESTAC, a tool for a stochastic round-off error analysis in scientific computing. *Numerical Mathematics and Applications*, pages 11–20, 1986.

[15] G. R. Hext W. Spendley and F. R. himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 37:441–461, 1962.

[16] Daniel J. Woods. An Interactive Approach for Solving Multi-Objective Optimization Problems. *Ph.D thesis, Rice University*, 1985.