Automatically Testing Numerical Stability

Sanzheng Qiao*
Department of Computing and Software
McMaster University
Hamilton, Ontario L8S 4L7 Canada

Shikun Wang†
Department of Computing and Software
McMaster University
Hamilton, Ontario L8S 4L7 Canada

1 Introduction

Testing plays an important role in software development. Testing numerical software involves issues such as numerical stability in addition to logic correctness. Traditionally, analyzing the stability of an algorithm requires expertise in numerical analysis to perform theoretical error analysis. The work is often tedious and sometimes difficult. Automatic error analysis uses the computer to help us analyze the stability of a numerical computation. The idea of automatic error analysis goes back to the dawn of scientific computing. Wilkinson introduced running error analysis on the ACE machine. Most of automatic error analysis methods leave an open problem: How should you choose a set of input data to reveal the instability of the algorithm.

In this report, we present an automatic method for testing numerical stability. It automatically selects a set of input data that will reveal possible instability of the numerical software under testing.

This report is organized as follows. Section 2 describes a direct search method for automatic error analysis proposed by Higham [2, Page 473]. The effectiveness of this method depends on the construction of an objective function. Section 3 introduces a debugging method by Kahan [4]. Like most "automatic" error analysis methods, this method requires the user to supply proper input data to reveal numerical instability. Our method, which requires neither the construction of an objective function nor proper input data, is presented in Section 4. Finally, Section 5 gives numerical examples and Section 6 makes concluding remarks.

---

*This work was partly supported by the Natural Sciences and Engineering Research Council of Canada under grant OGP0046301

†This work was supported partly by CITO and partly by NSERC

2 Direct Search Method

Higham [2, Page 473] proposed a direct search method for automatic error analysis. The idea is to use optimization techniques to automatically find a set of input data to reveal possible instability of an algorithm. An algorithm can be viewed as a function

$$y = g(x)$$

mapping the input vector x into the output vector y. The idea of the direct search method is to find an input vector x that will maximize the computational errors. Since the derivative of the function g is normally unavailable, direct search methods are used. However, a question remains: How is the objective function determined? Obviously, the arguments of the objective function are the input data. If we know the exact results, then the value of the objective function can be the difference $|y - \widehat{y}|$ between the exact output y and the computed output $\widehat{y}$, assuming that all variables are equally important.

Example 1 *Solving a Cubic [2, Page 483].*

Consider the problem of solving a cubic

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

Changing the variable

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

we eliminate the quadratic term

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

Vieta's substitution $y = w - p/(3w)$ gives

$$w^3 - \frac{p^3}{27w^3} + q = 0$$

leading to $(w^3)^2 + qw^3 - p^3/27 = 0$, which is solved by

$$w^3 = -\frac{q}{2} \pm \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}. \tag{1}$$

Higham applied the direct search method to investigate the stability of the above algorithm. The arguments of the objective function are the coefficients a, b, and c. The value of the objective function is

$$\|z - \widehat{z}\|_\infty / \|z\|_\infty. \tag{2}$$

where $\widehat{z} \in C^3$ is the root vector computed by the above algorithm and z contains the roots computed by an accurate method such as MATLAB's `roots` function. We consider z

as “exact” since `roots` function uses the QR algorithm to compute the eigenvalues of the companion matrix [1]. The optimization method used is the multidirectional search method (MDS) of Dennis and Torczon [6]. Starting with $[a\ b\ c] = [1\ 1\ 1]$, the MDS maximizer rapidly found

$$[a\ b\ c]^{\mathrm{T}} = [1.732\ 1\ 1.2704],$$

which caused inaccurate computed results comparing with the “exact” results:

$$\widehat{z} = \begin{bmatrix} -1.5999e+0 \\ -6.6066e-2-8.8557e-1\,i \\ -6.6066e-2+8.8557e-1\,i \end{bmatrix}, \quad z = \begin{bmatrix} -1.6026e+0 \\ -6.4678e-2-8.8798e-1\,i \\ -6.4678e-2+8.8798e-1\,i \end{bmatrix}.$$

In the above algorithm, we arbitrarily take the “+” sign for the square root in (1). To avoid possible catastrophic cancellation, we take

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

With the use of (3), Higham was unable to locate instability using the objective function (2). Higham defined another objective function

$$\max_i \frac{|f(z_i)|}{\max(|a|,\ |b|,\ |c|,\ 1)\sum_{j=0}^{3}|\widehat{z}_i^{j}|}, \tag{4}$$

and obtained another set of input data

$$[a\ b\ c]^{\mathrm{T}} = [-589\ 315\ \ -13.6]$$

with starting vector $[1\ 1\ 1]^{\mathrm{T}}$. The computed roots are

$$\widehat{z}^{\mathrm{T}} = [0.0475\ 0.487\ 589].$$

The value of the objective function (4), which measures the error, corresponding to the “exact” roots is of order 10^{-16} whereas the value corresponding to $\widehat{z}$ is of order 1.2×10^{-11}.

This example shows that the effectiveness of this direct search method depends on the construction of the objective function, which can be a difficult task. The objective function (2) is simple, but it failed to detect the instability when (3) was used. On the other hand, the objective function (4) detected instability, but it was not so obvious. Higham derived the objective function (4) using backward error analysis.

3 Instability Detection

The direct search method described in the previous section requires the construction of an objective function. In Example 1, the objective function (2) uses the “exact” results, which are unknown in most applications. Kahan [4] proposed a method for detecting instability

Rounding mode	Heron's Formula	Better Formula
to nearest	12345680.0	6249012.0
to $+\infty$	12345680.0	6249013.0
to $-\infty$	0.0	6249011.0
to 0	0.0	6249011.0

Table 1: Area of the triangle with side-lengths $a = 12345679$, $b = 12345679$, and $c = 1.01233995$ computed in different rounding modes.

without the knowledge of "exact" results. The idea is to compute the results in different rounding modes specified by IEEE floating-point standards [3]. If the difference between the results obtained by different rounding modes is large, then the program is numerically unstable provided that the problem is well-conditioned. The following example [4, Page 58] illustrates Kahan's method.

Example 2 *Computing the area of a needle-like triangle.*

Let a, b, and c be the three side-lengths of a triangle, a classical formula of triangle area due to Heron is

$$\triangle = \sqrt{s(s-a)(s-b)(s-c)}, \qquad s = (a+b+c)/2. \tag{5}$$

A better but less-known method is as follows.

Algorithm 1 *A better method for computing the area of a triangle.*

1. sort a, b, c, so that $a \geq b \geq c$;
2. if $c - (a - b) < 0$ then the data are not side-lengths of a real triangle;
   else
   $$\triangle = \frac{(a + (b + c))(c - (a - b))(c + (a - b))(a + (b - c))}{4}.$$

Kahan computed, in float (single precision), the area of a needle-like triangle with the following side-lengths:

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

This triangle is well-conditioned (condition number 2). So, a stable method should be insensitive to the change of round mode. Table 1 [4, Page 59] shows that Heron's formula is numerically unstable.

Kahan's method does not require an objective function, however, it requires the construction of proper input data to reveal the instability. For non needle-like triangles, Heron's formula is accurate.

4 Automatic Testing Method

In this section, we present our method for debugging numerical instability. Our method does not require the user to construct an objective function. Moreover, like the direct search method, it automatically finds a set of input data to reveal possible instability. The objective function is determined as follows. The arguments are the input data. The value of the objective function is the maximal difference between the results computed in different rounding modes. Then, as in the direct search method, an optimizer is used to find a set of input data maximizing the value of the objective function. If we view a program as a function $y = g(x)$, mapping input vector x into output vector y, then the computed result $\widehat{y}$ can be viewed as the exact output of the function g with a perturbed input $\widehat{x} = x + \delta x$, i.e.,

$$\widehat{y} = g(x + \delta x).$$

Therefore, results computed in different rounding modes can be viewed as exact results of g with differently perturbed inputs. Suppose we compute $\widehat{y}_1$ and $\widehat{y}_2$ in two rounding modes, then we have

$$\widehat{y}_1 = g(x + \delta x_1) \quad \text{and} \quad \widehat{y}_2 = g(x + \delta x_2).$$

As we know, the forward error

$$|\widehat{y} - y|/|y| = \frac{|\widehat{y} - y|/|y|}{|\delta x|/|x|} |\delta x|/|x|$$

where

$$\text{cond} = \frac{|\widehat{y} - y|/|y|}{|\delta x|/|x|}$$

is the condition number of the problem, which measures the sensitivity of the problem. This means that if the problem is well-conditioned then large forward error implies large backward error $||\delta x|/|x|$, i.e., the method is (backward) unstable. Since the exact result y is usually unknown, we use $|\widehat{y}_1 - \widehat{y}_2|$ as an approximation.

Although this method does not require the user to construct an objective function and input data, its effectiveness depends heavily on the optimizer used. Also, after a set of input data is obtained, we must decide whether this set of input data determines an ill-conditioned problem. If the problem corresponding to the input data is well-conditioned, then the program is numerically unstable.

5 Examples

We programed our method in C and ran all examples on a SUN SPARC V running Solaris 2.7. Libraries `-lm` and `-lsunmath` were linked.

Example 3 *Debugging cubic.*

Rounding Mode	always "+" in (1)	MATLAB
to $+\infty$	$-1.\mathbf{5}\cdots$ $-0.1\mathbf{6}\cdots \pm i\,0.\mathbf{8}\cdots$	
to $-\infty$	$\mathbf{4.5}\cdots$ $-\mathbf{3.2}\cdots \pm i\,\mathbf{4.5}\cdots$	
to 0	$\mathbf{4.5}\cdots$ $-\mathbf{3.2}\cdots \pm i\,\mathbf{4.5}\cdots$	
to nearest	$-1.\mathbf{5}\cdots$ $-0.1\mathbf{6}\cdots \pm i\,0.\mathbf{8}\cdots$	$-1.66863117355877\cdots$ $0.1099314132206\cdots \pm i\,0.9044999271867\cdots$

Table 2: The cubic roots computed in four rounding modes and single precision compared with those computed by MATLAB. Inaccurate digits are shown in bold face.

We tested our method on the cubic problem in Example 1. Both formulas (1) with "+" and (3) were used. The optimizer used is Nelder-Mead simplex method obtained from [5]. The starting values were $[1.0, 1.0, 1.0]$. We used $\rightarrow +\infty$ and $\rightarrow -\infty$ rounding modes in the objective function and found coefficients:

$$a = 1.888494 \quad b = 1.197075 \quad c = 1.385306$$

in single precision and

$$a = 1.903262 \quad b = 1.207457 \quad c = 1.415465$$

in double precision. Then we used the computed coefficients of the equation to compute the roots in other rounding modes (nearest and $\rightarrow 0$). Table 2 shows the results for the single precision case and Table 3 shows the results for the double precision case. Also, the computed roots are compared with those computed by MATLAB function `roots( )` where bold figures indicate inaccurate digits. We consider the roots computed by MATLAB `roots( )` are accurate because `roots( )` computes the eigenvalues of the companion matrix and this eigenvalue method is stable [1].

The values of the cubic at the roots computed in the four rounding modes are

$$f(z) = O(10^{-1}) \quad \text{and} \quad f(z) = O(10^{-5})$$

in single precision and double precision respectively. Whereas the roots computed by MATLAB `roots( )` give

$$f(z) = O(10^{-15}).$$

Since the roots are well separated, we can conclude that this method for computing the roots is unstable. In this example, we used (1) with "+" to compute w^3. we found that our method found the coefficents where

$$\frac{p^3}{27} \approx \frac{q^2}{4}\epsilon$$

Rounding Mode	always "+" in (1)	MATLAB
to $+\infty$	$\mathbf{0.15}\cdots$ $-\mathbf{1.0}\cdots \pm i\,0.\mathbf{68}\cdots$	
to $-\infty$	$-1.\mathbf{4}\cdots$ $-0.\mathbf{2}\cdots \pm i\,0.\mathbf{6}\cdots$	
to 0	$-1.\mathbf{4}\cdots$ $-0.\mathbf{2}\cdots \pm i\,0.\mathbf{6}\cdots$	
to nearest	$-1.68517\mathbf{6}\cdots$ $-0.10904\mathbf{3}\cdots \pm i\,0.9099\mathbf{8}\cdots$	$-1.68517963010202\cdots$ $0.1090411849489\cdots \pm i\,0.9099775011221\cdots$

Table 3: The cubic roots computed in four rounding modes and double precision compared with those computed by MATLAB. Inaccurate digits are shown in bold face.

where ϵ is the machine precision. Thus catastrophic cancellation caused unstability.

Example 4 *Better formula.*

We replaced (1) with the better formula (3) to avoid cancellation and repeated Example 3. Using the starting values $[1.0, 1.0, 1.0]$ and $\to +\infty$ and $\to -\infty$ rounding modes we got

$$a = 2.224135 \quad b = -1.862519 \quad c = 1.321799$$

in single precision and

$$a = 0.960716 \quad b = 1.037320 \quad c = 1.084460$$

in double precision. Then we computed the roots using the computed coefficients in four rounding modes. Table 4 shows the results for the single precision case and Table 5 shows the results for the double precision case.

The values of the cubic at the roots computed in the four rounding modes are

$$f(z) = O(10^{-7}) \quad \text{and} \quad f(z) = O(10^{-7})$$

in single precision and double precision respectively. The roots computed by MATLAB `roots( )` give

$$f(z) = O(10^{-15}).$$

6 Remarks

Obviously, the performance of our method depends heavily on the optimizer, which in turn depends on factors such as the initial values.

Although our method automatically finds a set of input data to reveal possible numerical instability, it still requires some human analysis to determine whether the algorithm under investigation is unstable.

Rounding Mode	always "+" in (1)	MATLAB
to $+\infty$	$-2.993750\cdots$ $0.384809\cdots \pm i\,0.54170\mathbf{4}\cdots$	
to $-\infty$	$-2.99375\mathbf{2}\cdots$ $0.384807\cdots \pm i\,0.54170\mathbf{2}\cdots$	
to 0	$-2.99375\mathbf{2}\cdots$ $0.384807\cdots \pm i\,0.54170\mathbf{2}\cdots$	
to nearest	$-2.99375\mathbf{1}\cdots$ $0.38480\mathbf{8}\cdots \pm i\,0.541703\cdots$	$-2.99375091865185\cdots$ $0.38480795932592\cdots \pm i\,0.54170305326116\cdots$

Table 4: The cubic roots computed in four rounding modes and single precision compared with those computed by MATLAB. Inaccurate digits are shown in bold face.

Rounding Mode	always "+" in (1)	MATLAB
to $+\infty$	$-1.003699\cdots$ $0.02149\mathbf{2}\cdots \pm i\,1.03923\mathbf{1}\cdots$	
to $-\infty$	$-1.00\mathbf{37}\cdots$ $0.021491\cdots \pm i\,1.039230\cdots$	
to 0	$-1.00\mathbf{37}\cdots$ $0.021491\cdots \pm i\,1.039230\cdots$	
to nearest	$\cdots$ $0.021492\cdots \pm i\,1.03923\mathbf{1}\cdots$	$-1.00369964615475\cdots$ $0.02149182307738\cdots \pm i\,1.0392308559587\cdots$

Table 5: The cubic roots computed in four rounding modes and single precision compared with those computed by MATLAB. Inaccurate digits are shown in bold face.

results	our method	MATLAB
solution x	-1.1466572×10^4	-1.0584446×10^4
	1.5859014×10^4	1.4638992×10^4
	1.5944596×10^3	1.4718593×10^3
residual	$\approx 10^{-3}$	$\approx 10^{-11}$

Table 6: The solution and residual of the system $Ax = b$ given by our method compared with those computed by MATLAB.

Example 5 *Gaussian elimination without pivoting.*

We applied our method to Gaussian elimination without pivoting. Starting with a 3-by-3 identity matrix and using $[1.0, 1.0, 1.0]^{\mathrm{T}}$ as the right-hand-side vector, our method found the matrix

$$A = \begin{bmatrix} 1.837961 & 1.226373 & 1.020434 \\ 0.855716 & 0.522740 & 0.955182 \\ 1.006330 & 0.615218 & 1.118500 \end{bmatrix}$$

maximizing the error between the solutions computed in single precision in two different rounding modes: $\to +\infty$ and $\to -\infty$. The solution and the residual computed in single precision and the nearest rounding mode, comparing with MATLAB results (double precision and the nearest rounding mode), are shown in Table 6. In this particular example, although our method found a matrix producing large residual, we cannot conclude that our method has revealed the instability of Gaussian elimination without pivoting. The condition number of A is

$$\mathrm{cond}(A) = 5.212 \times 10^5.$$

So, this problem is ill-conditioned. Only when the problem is well-conditioned, large errors imply numerical instability as shown in Examples 3 and 4.

References

[1] Alan Edelman and H. Murakami. *Polynomial Roots From Companion Matrix Eigenvalues.* Math. Comp. 64(210):763–776, 1995.

[2] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithm.* Society for Industrial and Applied Mathematics, 1996.

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

[4] W. Kahan and Joseph D. Darcy. *How Java's Floating-Point Hurts Everyone Everywhere,* 1998.
URL=http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf

[5] http://home.earthlink.net/mfhutt/neldermead.htm

[6] Virginia J. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. Ph.D. thesis, Rice University, Houston, TX, USA, May 1989.