Miskolc Mathematical Notes
Vol. 5 (2004), No 1, pp. 25-32

HU e-ISSN 1787-2413
DOI: 10.18514/MMN.2004.80

Automatic error analysis with Miller's method

Attila Gáti

Miskolc Mathematical Notes
Vol. 5 (2004), No. 1, pp. 25–32

HU ISSN 1586-8850

AUTOMATIC ERROR ANALYSIS WITH MILLER'S METHOD

ATTILA GÁTI

[Received: October 09, 2003]

Abstract. The first considerable roundoff analyser software, which can provide global error bounds for a given numerical algorithm, was designed by Webb Miller and David Spooner in the second half of the 70s. Despite its abilities, the recently software is not widely used, and since its publication it has not been upgraded. The software was written in Fortran IV for the IBM 360/370 series. Without any modifications, it is not usable in a PC environment with the most widely used compilers based on the Fortran 77 standard (GNU compiler, Watcom, etc.), because there are several non-standard solutions in the source. In addition, the user is bound by extraordinary strict limitations of the size of the algorithm to analyse. We have upgraded the package in order to make it suit the Fortran 77 standard, and to expand the size limits. With the upgraded version we have analysed two variants of the ABS method: the implicit LU and Huang methods.

Mathematics Subject Classification: Primary 12G34; Secondary 09G87

Keywords: Roundoff analysis, automatic roundoff analysis, ABS method, implicit LU method, Huang method

1. Introduction

On digital computers, the arithmetic operations are performed in finite precision. The analyses of propagation of the rounding errors in numerical algorithms, i. e., the numerical stability, is one of the major tasks of numerical analysis. Basically, we can distinguish three different approaches to decide whether an algorithm is stable or unstable. The most obvious technique is testing the numerical algorithm upon sets of data for which the exact, or a relatively accurate, solution is known. Another approach is formal roundoff analysis. The best known result of that type is Wilkinson's theorem. According to it, the $\hat{x}$ computed solution of the $n \times n$ system of linear equations $Ax = b$ found by the Gaussian elimination satisfies the relation

$$(A + \Delta A)\,\hat{x} = b,$$

where $\|\Delta A\|_\infty \leq 8n^3 \varrho_n \|A\|_\infty u + O(u^2)$ (ϱ_n is the growth factor of the pivot entries, and u is the machine rounding unit).

Both of these two approaches have defects. Testing is sometimes insufficient, and the formal analysis needs a great deal of effort, which cannot always be afforded. Because of this, even at the dawn of scientific computing already techniques appeared in which computer is used to help us analyse the stability of a numerical computation. So the third approach of analysing the effects of finite precision arithmetic is the automatic roundoff analysis. Miller makes a distinction between local and global techniques of automatic roundoff analysis [1]. Local methods are used to bound, or estimate the error incurred in a single computation. Most local techniques use special systems of computer arithmetic (e.g. interval arithmetic, or unnormalised arithmetic) to monitor the error in each computed value. Using global techniques we can determine the propagation of rounding errors in a given numerical method for all permissible sets of data. In the beginning only local techniques were developed, the first considerable roundoff analyser software, which can provide global error bounds for a given numerical algorithm, was designed by Webb Miller and David Spooner in the second half of the 1970s.

2. The Miller–Spooner roundoff error analyser software

The software can be found in the ACM TOMS library with serial number 532, and the Fortran language source code is available freely at the following location:

`http://www.netlib.org/toms/532`

Miller and his co-workers published the use and the theoretical background of the program in two articles [1, 2] and also in a book [3]. The software package is intended to help algorithm designers seeking numerically stable algorithms, or a user doing initial testing of a numerical algorithm proposed for use.

The software cannot deal with every kind of numerical algorithm. Instead, direct methods solving algebraic problems are most susceptible for analysis. The numerical method to be analysed must be expressed in a special Fortran-like language. The language allows for-loops and if-tests that are not based on the values of real variables. If an algorithm contains branching based on real variables, then the possible paths through comparisons must be treated separately. The parameters of the algorithm defining the size of the problem (sizes of vectors and matrices) must be fixed for analysis, and these parameters cannot be arbitrarily large due to memory limitations.

The method is based on the standard model of the floating point arithmetic, so we assume that the relative error of each arithmetic operation is bounded by the machine rounding unit, and we ignore the possibilities of overflow and underflow.

The basic idea of Miller's method is very simple. Given a numerical method to analyse, a number $\omega(d)$ is associated with each set d ($d \in \mathbb{R}^n$) of data. The function ω measures rounding error, i. e., $\omega(d)$ is large exactly when the method applied to d produces results which are excessively sensitive to rounding errors. A numerical maximiser is applied to search for large values of ω to provide information about the numerical properties of the method. Finding large values of ω can be interpreted to

mean that the given numerical method suffers from a specific kind of instability. With the software package we can also compare the numerical stability of two algorithms Q and R, which neglecting rounding errors, compute identical values. In this case ω is large exactly when Q produces a much larger error than R.

The computation of ω is based on the partial derivatives at the nodes of the program's computational graph. As we use derivatives, the second order effects of rounding errors are neglected. In general ω is not a differentiable function of data d, so a direct search method is used for maximising ω. As these methods are heuristic, failure of the maximiser to find large values of ω does not guarantee that none exists, so unstable methods may appear to be stable.

The software package consists of three programs, a minicompiler and two error analyser programs. The minicompiler takes as data the numerical method to analyse, written in a special programming language, and produces as output a translation of that program into a series of assignment statements. The output is presented in a readable form for the user, and in a coded form for use as input to the error analyser programs. One of the error analyser programs (i) is for deciding numerical stability of a single algorithm, and the other (ii) is for comparing two competing algorithms. Both (i) and (ii) form automatically the partial derivatives, and based on them, can compute various error-measuring numbers (various ways to assign ω).

First let us consider the error measuring numbers of program (i). The software provides four stability-measuring numbers, $JW_E(d)$, $JW_L(d)$, $WK_E(d)$, $WK_L(d)$, which measure the minimum problem perturbation equivalent to rounding error in a given program at data d. Other numbers $ER_E(d)$ and $ER_L(d)$ use a weaker comparison of computed and exact solutions. Let $Z : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be a vector-valued function, and R an algorithm with k arithmetic operations for evaluating Z. If the operations are performed in floating point arithmetic, with the assumption of the standard model, then the computed value is a function $R(d, \delta)$, where δ is the vector of individual relative rounding errors on the k arithmetic operations ($\delta \in \mathbb{R}^k$, $\|\delta\|_\infty \leq u$).

$JW_E(d)$ is the smallest number for which the equality

$$R(d, \delta) = Z(d + \pi)$$

holds, for some $\pi \in \mathbb{R}^n$, $|\pi_i| \leq JW_E(d) \cdot |d_i| \cdot u$, $1 \leq i \leq n$, whenever $\|\delta\|_\infty \leq u$. So, $JW_E(d)$ measures the componentwise backward error. The normwise backward error is measured by $JW_L(d)$, in this case $\|\pi\|_\infty \leq JW_L(d) \cdot \|d\|_\infty \cdot u$. WK_X measures ($X = E$, or L) the mixed forward-backward error, so we also allow perturbations in the output space. $WK_E(d)$ is the smallest number for which the equality

$$R(d, \delta) = Z(d + \pi) + \phi$$

holds for some $|\pi_i| \leq WK_E(d) \cdot |d_i| \cdot u$, and $\phi \in \mathbb{R}^m$, $|\phi_i| \leq WK_E(d) \cdot |Z_i(d)| \cdot u$, whenever $\|\delta\|_\infty \leq u$. In the case of the normwise measuring number $WK_L(d)$, we have

$$\|\pi\|_\infty \leq WK_L(d) \cdot \|d\|_\infty \cdot u$$

and

$$\|\phi\|_\infty \leq WK_L(d) \cdot \|Z(d)\|_\infty \cdot u.$$

Note that

$$WK_L(d) \leq WK_E(d) \leq JW_E(d)$$

and

$$WK_L(d) \leq JW_L(d) \leq JW_E(d).$$

The numbers JW_X and WK_X require that the computed solution be obtained at (or close to) the exact solution. ER_X measures how much the data must be perturbed to create an error as large as that in the computed solution. So $ER_E(d)$ is the smallest number for which

$$\|R(d,\delta) - Z(d)\|_\infty = \|Z(d+\pi) - Z(d)\|_\infty$$

for some π, where $|\pi_i| \leq ER_E(d) \cdot |d_i| \cdot u$, $1 \leq i \leq n$. ER_L measures normwise, in this case the same equation holds for π, $\|\pi\|_\infty \leq ER_L(d) \cdot \|d\|_\infty \cdot u$. The inequalities

$$ER_L(d) \leq ER_E(d)$$

and

$$ER_X(d) \leq JW_X(d),$$

where X is either E or L, are obvious.

Now let us take a glance at the error-measuring numbers used by program (ii). Let Q and R be two algorithms for evaluating the function Z, and the computed values at data d, with rounding errors $\delta \in \mathbb{R}^k$ and $\gamma \in \mathbb{R}^l$ be $Q(d,\delta)$ and $R(d,\gamma)$. $JW_{Q/R}(d)$ is the smallest number for which the condition

$$Q(d,\delta) = R(d,\gamma)$$

holds whenever $\|\delta\|_\infty \leq u$, for some $\|\gamma\|_\infty \leq JW_{Q/R}(d) \cdot u$. $ER_{Q/R}(d)$ is the smallest number for which

$$\|Q(d,\delta) - Z(d)\|_\infty = \|R(d,\gamma) - Z(d)\|_\infty$$

for all $\|\delta\|_\infty \leq u$, for some $\|\gamma\|_\infty \leq ER_{Q/R}(d) \cdot u$. The inequalities

$$ER_{Q/R}(d) \leq JW_{Q/R}(d)$$

and

$$\frac{JW_X^Q}{JW_X^R} \leq JW_{Q/R}(d),$$

where $X = E$ or $X = L$, will also hold.

The input for the error-testing programs is arranged as follows: (1) the output of the minicompiler (for program (i), which is the compiled version of the algorithm to analyse; in the case of program (ii) we must provide the compiled version of Q followed by R), (2) a list of initial data for the numerical maximiser, (3) the code of the chosen error-measuring value, (4) target value for the maximiser. The programs return with the final set of data found by the maximiser routine and with the value

of the chosen error-measuring number at this set of data. If program (i) diagnoses instability (the value of the error-measuring number at the final set of data exceeds the target value), then the condition number is also computed at the final set of data.

3. Upgrading the software package

The Miller–Spooner roundoff error analyser software was written in Fortran IV for the IBM 360/370 series, and its final version was issued in 1979. Without any modifications, it is not usable in a PC environment with the most widely used compilers based on the Fortran 77 standard (GNU compiler, Watcom, etc.), because there are several non-standard solutions in the source. In addition, the user is bound to extraordinarily strict limitations of the size of the algorithm to analyse. The number of inputs cannot exceed 30, the number of outputs must be smaller than 20, and the total number of inputs and real arithmetic operations may not exceed 300. We have upgraded the package in order to make it suit the Fortran 77 standard, and to expand the size limits. The upgraded version works correctly with the compilers mentioned earlier, and we can analyse algorithms with inputs up to 3000, outputs up to 1000, and the total number of inputs and operations can be maximum 50000.

In [3], the abilities of the software are demonstrated by 14 important case studies. In these cases the answers of the software are consistent with the known formal analytical results. The program shows correctly the stability properties of the Gaussian elimination without pivoting and with partial pivoting, the Gauss–Jordan elimination, the Cholesky factorisation, several QR factorisation methods (Householder transformation, Givens transformation, method by solving the normal equations), the classical and modified Gram-Schmidt methods and so on. We have tested the upgraded version on these case studies, and with small dimensions we have obtained results similar to those of Miller's, with larger dimensions the results were consistent.

4. Analysis of the implicit LU and Huang methods

Using the software package we have analysed the numerical stability of solving the $Ax = b$ linear system ($A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$) with the implicit LU and Huang methods. These methods are special variants of the ABS method. The ABS method:

Algorithm 1. $x_1 \in \mathbb{R}^n$, $H_1 = I$, $V = [v_1, ..., v_n] \in \mathbb{R}^{n\times n}$,
$W = [w_1, ..., w_n] \in \mathbb{R}^{n\times n}$, $Z = [z_1, ..., z_n] \in \mathbb{R}^{n\times n}$
for $k = 1, ..., n$

$p_k = H_k^T z_k \qquad (z_k \in \mathbb{R}^n,\ p_k^T A^T v_k \neq 0)$

$x_{k+1} = x_k - \frac{p_k v_k^T (Ax_k - b)}{v_k^T A p_k}$

$H_{k+1} = H_k - \frac{H_k A^T v_k w_k^T H_k}{w_k^T H_k A^T v_k} \qquad (w_k \in \mathbb{R}^n,\ w_k^T H_k A^T v_k \neq 0)$

end

$x_{n+1} = A^{-1} b$

4.1. The implicit LU method. The implicit LU algorithm is given by the choice $W = Z = I$. We consider the special case, when $V = I$ and $x_1 = 0$. So the algorithm to analyse is the following one:

Algorithm 2. $x_1 = 0,\ H_1 = I$
for $k = 1, ..., n$
$x_{k+1} = x_k - \frac{H_k^T e_k e_k^T (Ax_k - b)}{e_k^T A H_k^T e_k}$
$H_{k+1} = H_k - \frac{H_k A^T e_k e_k^T H_k}{e_k^T H_k A^T e_k}$
end

Let the initial data for maximising be:

$$A_0 = \begin{bmatrix} 3 & 1 & 1 & 1 \\ 1 & 4 & 1 & 1 \\ 1 & 1 & 5 & 1 \\ 1 & 1 & 1 & 6 \end{bmatrix}, \qquad b_0 = \begin{bmatrix} 6 \\ 7 \\ 8 \\ 9 \end{bmatrix}.$$

We have chosen $10,000$ as target value. The maximiser finds values greater than the target value for both WK_L and ER_L error-measuring numbers. Because of the inequalities held between the error-measuring numbers, a set of data exists for all error-measuring numbers, where their values exceed $10,000$. In addition, at the final set of data the condition number is extremely low. In the case of WK_L it is 12.3, and in the case of ER_L it is 8.1. So the implicit LU algorithm is unstable even at very well-conditioned data.

Comparing the algorithm with the Gaussian elimination without pivoting, we obtain

$$JW_{\text{LU/Gauss}} > 10,000, \quad JW_{\text{Gauss/LU}} > 10,000, \quad ER_{\text{LU/Gauss}} > 10,000,$$

and

$$ER_{\text{Gauss/LU}} < 29.$$

So, the implicit LU has worse stability properties than the Gaussian elimination without pivoting.

4.2. The Huang method. The algorithm is obtained by the choice

$$w_i = z_i = A^T v_i.$$

We have analysed four variants of the method. With the given substitutions applied to the general ABS algorithm, we get the first variant:

Algorithm 3. $x_1 \in \mathbb{R}^n,\ H_1 = I$
for $k = 1, ..., n$
$p_k = H_k^T A^T e_k$
$x_{k+1} = x_k - \frac{p_k e_k^T (Ax_k - b)}{e_k^T A p_k}$

$$H_{k+1} = H_k - \frac{H_k A^T e_k e_k^T A H_k}{e_k^T A H_k A^T e_k}$$

end

We have started the maximiser from the same initial data A_0, b_0 as in the case of the implicit LU method. Applying the maximiser, ER_L and WK_L have exceeded $10,000$, so the method is unstable. Unlike the case of implicit LU, large values were found at ill-conditioned data ($61,710$ for WK_L and $32,365$ for ER_L). So, the algorithm is more sensitive to ill-conditioning than it is reasonable. Comparing the algorithm with the modified Gram-Schmidt method, we get the following results:

$$JW_{\text{Huang /MGS}} > 10,000, \quad JW_{\text{MGS/ Huang}} < 3,400, \quad ER_{\text{Huang /MGS}} > 10,000,$$

and $ER_{\text{MGS/ Huang}} < 2050$. So, they are both sets of data where the Huang and sets of data where the MGS method give much poorer results.

If we consider that $H_i = H_i^T$, then the algorithm can be modified as follows:

Algorithm 4. $x_1 \in \mathbb{R}^n$, $H_1 = I$
for $k = 1, ..., n$

$$p_k = H_k^T A^T e_k$$
$$x_{k+1} = x_k - \frac{p_k e_k^T (Ax_k - b)}{e_k^T A p_k}$$
$$H_{k+1} = H_k - \frac{p_k p_k^T}{e_k^T A p_k}$$

end

In this case, we have results very similar to the previous algorithm. Comparing the two methods (Algorithm 3 and Algorithm 4), we see that all the error-measuring numbers are bounded by 6.1. Thus, the two variants have the same stability properties.

The following version is based on the fact that $H_i^2 = H_i$:

Algorithm 5. $x_1 \in \mathbb{R}^n$, $H_1 = I$
for $k = 1, ..., n$

$$p_k = H_k^T A^T e_k$$
$$x_{k+1} = x_k - \frac{p_k e_k^T (Ax_k - b)}{e_k^T A p_k}$$
$$H_{k+1} = H_k - \frac{p_k p_k^T}{p_k^T p_k}$$

end

We have values for ER_L and WK_L exceeding $10,000$ also at ill-conditioned sets of data. Comparing the algorithm with the MGS method we get results similar to the previous two cases. If we compare the method with the second variant (Algorithm 4), we get that all error-measuring numbers are not bounded by $10,000$ except that

$$ER_{\text{Huang 3/ Huang 2}} < 1.5.$$

So, the third variant is more stable than the previous ones.

The fourth analysed algorithm was the so-called modified Huang method:

Algorithm 6. $x_1 \in \mathbb{R}^n$, $H_1 = I$
for $k = 1, ..., n$
$p_k = H_k \left(H_k^T A^T e_k \right)$
$x_{k+1} = x_k - \frac{p_k e_k^T (Ax_k - b)}{e_k^T A p_k}$
$H_{k+1} = H_k - \frac{p_k p_k^T}{p_k^T p_k}$
end

In this case the maximiser cannot find a value for JW_L greater than 4.3, so the algorithm is backward stable. A comparison with the MGS method produces the following results:

$$JW_{\text{Huang}/\text{MGS}} < 561, \quad JW_{\text{MGS}/\text{ Huang}} > 10,000, \quad ER_{\text{Huang}/\text{MGS}} < 20,$$

and

$$ER_{\text{MGS}/\text{ Huang}} < 4,038.$$

So, the modified Huang method has better stability properties than the MGS method. If we compare the method with the previous (third) variant, we have that

$$ER_{\text{Huang 4}/\text{ Huang 3}} < 1.23, \qquad JW_{\text{Huang 4}/\text{ Huang 3}} < 1.6,$$

and the other two numbers are not bounded by $10,000$. So, the method is much more stable than the previous variants of the Huang method.

References

[1] Miller, W.: *Software for roundoff analysis*, ACM Trans. Math. Software, **1** (1975), No. 2, 108–128.

[2] Miller, W. and Spooner, D.: *Software for roundoff analysis II*, ACM Trans. Math. Software, **4** (1978), No. 4, 388–390.

[3] Miller, W. and Wrathall, C.: *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, New York, 1980.

[4] Abaffy, J. and Spedicato, E.: *ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations*, Ellis Horwood Limited, Chichester, 1989.

Author's Address

Attila Gáti:
Institute of Mathematics, University of Miskolc, 3515 Miskolc-Egyetemváros, Hungary
E-mail address: matgati@gold.uni-miskolc.hu