2005 International Symposium on Nonlinear
Theory and its Applications (NOLTA2005)
Bruges, Belgium, October 18-21, 2005

Componentwise Verified Solutions of Linear Systems Suited for Java

K. Ozaki[1], T. Ogita[2,3], S. Miyajima[3], S. Oishi[3], S. M. Rump[4]

[1] Graduate School of Science and Engineering, Waseda University, Tokyo 169-8555, Japan
[2] CREST, Japan Science and Technology Agency (JST)
[3] Faculty of Science and Engineering, Waseda University, Japan
[4] Inst. f. Computer Science III, Hamburg University of Technology, Germany
Email: k_ozaki@suou.waseda.jp

Abstract—A fast and portable verification method is proposed for computing tight and componentwise error bounds for approximate solutions of linear systems. This method requires no switch of rounding mode so that it is applicable in Java keeping the portability. Finally numerical examples are presented to show the efficiency of this method.

1. Introduction

In this paper, we are concerned with a verification for an approximate solution of a linear system

$$Ax = b\,, \tag{1}$$

where A is a real $n \times n$ matrix and b a real n-vector. For verification for an approximate solution of (1), various methods have been proposed (e.g. [7, 8, 5, 11]). Especially in this paper, we consider the verification method for approximate solutions of linear systems implementable on a wide range of programming language including Java.

Here, it should be explained why special attention must be paid for Java. On the one hand, Java has remarkable splendid features. For instance, it is a portable programming language, i.e. it is designed to be independent of operating systems and compilers. Thus, once one develops a Java's program, one can obtain the same result on every platform. The performance of Java has recently been surprisingly increased via developments of its optimization techniques. As a result, recently, Java has been used in high performance computing. On the other hand, to keep the portability, the switch of rounding modes in IEEE 754 standard has not been supported independently in Java. Therefore, to verify an approximate solution of (1) keeping the portability, we should develop a method which does not use the directed rounding of IEEE 754.

We have recently proposed a fast verification method suited for Java [9], which does not use the directed rounding. In [9], we used only the round-to-nearest mode to give verified normwise error bounds. For this aim, we have utilized a priori error estimates for floating-point arithmetic. Moreover, an accurate and portable dot product algorithm proposed in [6] has been introduced into the method to avoid overestimation which is often observed when using a priori error estimates.

However, such a normwise estimation does not necessarily give a tight bound for each entry of an approximate solution because the normwise bound often depends on an entry whose absolute value is the largest in all entries of the solution vector. Namely, if there is a big difference among the solution in terms of the order of magnitude, its normwise error bound should be overestimated for an entry whose absolute value is relatively small.

The purpose of this paper is to propose a fast verification method which supplies a tight and componentwise error bound keeping the portability of Java. The proposed method is based on Ogita-Oishi-Ushiro method [5] which gives a tight and componentwise error bound for an approximate solution of (1). Following [9], we utilize a priori error estimates and a fast and portable dot product algorithm to avoid switching the rounding mode and to obtain a tight error bound in Java. Finally, the results of numerical experiments are presented to show the efficiency of this method.

2. Floating-Point Arithmetic in Java

In Java, the formats of IEEE 754 single and double precisions are adopted with respect to the floating-point numbers [3]. The round-to-nearest mode is set up in default. However, the switch of rounding modes is not supported. The extended precisions for single and double precisions are admitted by IEEE 754, respectively. In Java, the extended precisions as "widefp mode" are set up in default. However, such extended precisions depend on CPUs in use so that it lessens portability of computational result. To keep the portability, one should use "strictfp mode". In this mode computations are executed strictly in IEEE 754 single or double precision. Therefore, computational results are always the same in every computer environment provided that one uses strictfp mode.

Let $\mathbb{F}$ be a set of floating-point numbers. Let $\mathrm{fl}(\cdots)$ be the result of a floating-point computations, where all operations inside parentheses are executed by ordinary floating-point arithmetic only in round-to-nearest mode. We assume that no over/underflow occurs.

We cite here the notations used in this paper. Let $\mathbf{u}$ be the unit roundoff (especially, $\mathbf{u} = 2^{-53}$ in IEEE 754 double precision). For $a, b \in \mathbb{F}$ and $x, y \in \mathbb{F}^n$, we will use the following relations [7]:

$$|a+b| \le (1+\mathbf{u})\mathrm{fl}(|a+b|) \tag{2}$$

$$(1+\mathbf{u})^n|a| \le \mathrm{fl}\left(\frac{|a|}{1-(n+1)\mathbf{u}}\right) \tag{3}$$

$$\|x\|_\infty = \mathrm{fl}(\|x\|_\infty) \tag{4}$$

Note that $x \le y$ means $x_i \le y_i$ for all i. Moreover, we denote by $|x|$ the nonnegative vector with $|x| = (|x_1|, \ldots, |x_n|)^T$. For real matrices, similar notations will be used.

We briefly review an accurate algorithm of calculating dot products and matrix-vector products with error bounds proposed in [6]. For $x, y \in \mathbb{F}^n$, their algorithm

$$[\mathtt{res}, \mathtt{err}] = \mathtt{DotKErr}(x, y, K)$$

calculates an approximation $\mathtt{res}$ of $x^T y$ as if calculated in K-fold the working precision and its error bound $\mathtt{err}$ such that

$$\mathtt{res} - \mathtt{err} \le x^T y \le \mathtt{res} + \mathtt{err}.$$

The algorithm DotKErr consists of only ordinary floating-point operations, so that an accurate dot product and its error bound can be calculated without directed rounding. We can apply the algorithm for calculating matrix-vector products. We assume that we can calculate an accurate matrix-vector product $z = Ax + \alpha y$ for $A = (a_{ij}) \in \mathbb{F}^{m\times n}$, $x, y \in \mathbb{F}^n$ and $\alpha \in \mathbb{F}$ as if calculated in K-fold the working precision and its error bound as follows:

$$z = \mathtt{MVK}(A, x, \alpha, y, K)$$

or

$$z = \mathtt{MVKErr}(A, \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}, \alpha, y, K)$$

and

$$[z_{\mathrm{mid}}, z_{\mathrm{rad}}] = \mathtt{MVKErr}(A, \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}, \alpha, y, K).$$

The latter calculates z_{mid} and z_{rad} such that

$$z_{\mathrm{mid}} - z_{\mathrm{rad}} \le \sum_{k=1}^{m} Ax^{(k)} + \alpha y \le z_{\mathrm{mid}} + z_{\mathrm{rad}}.$$

3. Previous Verification Methods

In this section, we briefly review a method of obtaining a verified error bound for an approximate solution of (1).

3.1. Ogita-Rump-Oishi Method

A verification method for (1) without directed rounding has been proposed by Ogita, Rump and Oishi [7]. In [7], some a priori error estimates for floating-point arithmetic are presented.

If floating-point numbers α and β can be obtained such that $\|RA - I\|_\infty \le \alpha < 1$ and $\|R(A\tilde{x} - b)\|_\infty \le \beta$, then A is nonsingular and an error bound of an approximate solution $\tilde{x}$ of (1) is given as

$$\|\tilde{x} - A^{-1}b\|_\infty \le \mathrm{fl}\left(\frac{\beta/(1-\alpha)}{1-3\mathbf{u}}\right). \tag{5}$$

Ogita-Rump-Oishi method gives a way of calculating α and β without directed rounding. We denote the algorithm presented in [7] which calculates an upper bound α of $\|RA - I\|_\infty$ as $\alpha = \mathtt{Alpha.Std}(A, R)$. It holds that

$$\alpha \le \mathrm{fl}(\|RA - I\|_\infty) + c_n\mathbf{u}\cdot\mathrm{cond}(A), \tag{6}$$

where $\mathrm{cond}(A) := \|A\|_2 \cdot \|A^{-1}\|_2$. We also denote the algorithm presented in [7] which calculates an upper bound β of $\|R(A\tilde{x} - b)\|_\infty$ as $\beta = \mathtt{Beta.Std}(A, \tilde{x}, b, R)$.

It is known that the bound using a priori error estimates often becomes pessimistic to a certain extent. Therefore, in [9] we have improved the estimation of β by combining the a priori error analysis [7] with the accurate dot product algorithm [6]. The algorithm presented in [9] denotes $\beta = \mathtt{Beta.New}(A, \tilde{x}, b, R)$.

We here present an algorithm presented in [9] which calculates an upper bound of $\|\tilde{x} - A^{-1}b\|_\infty$.

Algorithm 1 (Ozaki et al. [9]) *Let $A \in \mathbb{F}^{n\times n}$ and $b \in \mathbb{F}^n$. Let R be an approximate inverse of A and $\tilde{x}$ an approximate solution of $Ax = b$. Then the following algorithm calculates an upper bound* err *of $\|\tilde{x} - A^{-1}b\|_\infty$.*

```
function err = Ozaki.Method($A, \tilde{x}, b, R$)
$\alpha$ = Alpha.Std($A, R$)
if $\alpha \ge 1$,   error('verification failed'),   end
$\beta$ = Beta.New($A, \tilde{x}, b, R$)
err = $\mathrm{fl}((\beta/(1-\alpha))/(1-3\mathbf{u}))$
```

4. A New Method

In this section, we propose a new method of calculating tight and componentwise error bounds for approximate solutions of linear systems.

We first introduce a theorem presented by Ogita, Oishi and Ushiro [5].

Theorem 1 (Ogita et al. [5]) *Let A, b, $\tilde{x}$ be as in Algorithm 1. Let $r := A\tilde{x} - b$ and $\tilde{z}$ be an approximate solution of $A\tilde{z} = r$. Then it holds that*

$$|\tilde{x} - A^{-1}b| \le |\tilde{z}| + \|A^{-1}\|_\infty \|A\tilde{z} - r\|_\infty e, \tag{7}$$

where $e = (1, \ldots, 1)^T \in \mathbb{R}^n$.

To obtain tight and componentwise error bounds, we here utilize the Ogita-Oishi-Ushiro method. Regarding $\tilde{z}$ in (7) as the sum of vectors $\sum_{k=1}^{m} |\tilde{z}^{(k)}|$ with $\tilde{z}^{(k)} \in \mathbb{F}^n$, we have

$$|\tilde{x} - A^{-1}b| \le \sum_{k=1}^{m} |\tilde{z}^{(k)}| + \|A^{-1}\|_\infty \left\| \sum_{k=1}^{m} A\tilde{z}^{(k)} - r \right\|_\infty e. \tag{8}$$

On estimating an upper bound of $\|A^{-1}\|_\infty$, if we obtain α such that $\|RA - I\|_\infty \le \alpha < 1$, then

$$\|A^{-1}\|_\infty \le \frac{\|R\|_\infty}{1-\alpha}. \tag{9}$$

From (2), an upper bound of $\|R\|_\infty$ can be calculated by

$$\|R\|_\infty \le (1+\mathbf{u})^{n-1} R_{\text{norm}}, \tag{10}$$

where $R_{\text{norm}} := \text{fl}\,(\|R\|_\infty)$. It follows that

$$\|A^{-1}\|_\infty \le \frac{\|R\|_\infty}{1-\alpha} \le (1+\mathbf{u})^{n-1}\frac{R_{\text{norm}}}{1-\alpha}. \tag{11}$$

We next consider estimating $\left\|\sum_{k=1}^{m} A\tilde{z}^{(k)} - r\right\|_\infty$. We denote $\tilde{z}^{(k)} \in \mathbb{F}^n$ for $k = 1,\dots,m$ as correction vectors for an approximate solution $\tilde{x}$, which are calculated as follows:

```
for $k = 1 : m$
    $r^{(k)} = \mathtt{MVK}(A, \{x, -\tilde{z}^{(1)}, \dots, -\tilde{z}^{(k-1)}\}, -1, b, k+1)$
    $\tilde{z}^{(k)} = \text{fl}(Rr^{(k)})$
end
```

Moreover, an inclusion of $r^{(m+1)} := r - \sum_{k=1}^{m} A\tilde{z}^{(k)}$ can be calculated using MVKErr by

$$[r_{\text{mid}}, r_{\text{rad}}] = \mathtt{MVKErr}(A, \{x, -\tilde{z}^{(1)}, \dots, -\tilde{z}^{(m)}\}, -1, b, m+2).$$

Using (2), we have

$$\|r^{(m+1)}\|_\infty \le (1+\mathbf{u}) r_{\text{norm}}, \tag{12}$$

where $r_{\text{norm}} := \text{fl}\,(\||r_{\text{mid}}| + r_{\text{rad}}\|_\infty)$. From (4), (11) and (12), we have

$$\begin{aligned} &\|A^{-1}\|_\infty \|r^{(m+1)}\|_\infty \\ \le\ & (1+\mathbf{u})^n \frac{R_{\text{norm}} r_{\text{norm}}}{1-\alpha} \\ \le\ & \text{fl}\left(\frac{(R_{\text{norm}} r_{\text{norm}})/(1-\alpha)}{1-(n+4)\mathbf{u}}\right) =: \beta_{\text{new}}. \end{aligned} \tag{13}$$

Using (2) and (3), we can calculate an upper bound of $\sum_{k=1}^{m} |z_k|$ as

$$\sum_{k=1}^{m} |\tilde{z}^{(k)}| \le \text{fl}\left(\frac{\sum_{k=1}^{m} |\tilde{z}^{(k)}|}{1-m\mathbf{u}}\right) =: t. \tag{14}$$

From (13) and (14), we finally have

$$|\tilde{x} - A^{-1}b| \le \text{fl}\left(\frac{t + \beta_{\text{new}} e}{1-2\mathbf{u}}\right). \tag{15}$$

Even if $\|A^{-1}\|_\infty$ is large, it can be expected to obtain sufficiently small $\|r^{(m+1)}\|_\infty$ as m increases. Therefore, the second term $\beta_{\text{new}} e$ is almost negligible by this approach. We now present the following algorithm.

Algorithm 2 *Let A, b, R and $\tilde{x}$ be as in Algorithm 1. Then the following algorithm calculates an upper bound y of $|\tilde{x} - A^{-1}b|$.*

```
function $y = \mathtt{Proposal.Method}(A, \tilde{x}, b, R, m)$
$\alpha = \mathtt{Alpha.Std}(A, R)$
if $\alpha \ge 1$,   error('verification failed'),   end
for $k = 1 : m$
    $r^{(k)} = \mathtt{MVK}(A, \{x, -\tilde{z}^{(1)}, \dots, -\tilde{z}^{(k-1)}\}, -1, b, k+1)$
    $\tilde{z}^{(k)} = \text{fl}(Rr^{(k)})$
end
$[r_{\text{m}}, r_{\text{r}}] = \mathtt{MVKErr}(A, \{x, -\tilde{z}^{(1)}, \dots, -\tilde{z}^{(m)}\}, -1, b, m+2)$
$r_{\text{norm}} = \text{fl}(\||r_{\text{mid}}| + r_{\text{rad}}\|_\infty)$
$R_{norm} = \text{fl}(\|R\|_\infty)$
$\beta_{\text{new}} = \text{fl}\left(\frac{(R_{\text{norm}} r_{\text{norm}})/(1-\alpha)}{1-(n+4)\mathbf{u}}\right)$
$t = \text{fl}\left(\frac{\sum_{k=1}^{m} |\tilde{z}^{(k)}|}{1-m\mathbf{u}}\right)$
$y = \text{fl}\,((t + \beta_{\text{new}} e)/(1-2\mathbf{u}))$   % $e = (1,\dots,1)^T$
```

In Algorithm 2, it requires $2n^3$ flops to calculate α for bounding $RA - I$ and $O(n^2)$ for the other computations, so that Algorithm 2 requires still $2n^3$ flops in total. Therefore, it can be expected that computing time for Algorithm 2 is almost the same as that for Algorithm 1. We will confirm it by numerical experiments in the next section.

5. Numerical Experiments

In this section, we illustrate the effectiveness of the proposed method. Following two methods are implemented on a PC with Pentium M 1.1GHz CPU and J2SDK1.4.2_06 as Java compiler and VM with IEEE 754 double precision and strictfp mode:

Method A Normwise error bound (Algorithm 1)

Method B Componentwise error bound (Algorithm 2)

We use JAMA [4], which is a Java matrix package, for calculating an approximate inverse R and an approximate solution $\tilde{x}$ of $Ax = b$.

First, we choose A a floating-point $n \times n$ matrix whose entries are pseudo-random numbers uniformly distributed in $[-1, 1]$. We put $b := \text{fl}(A \cdot c)$ with $c := (10, 20, \dots, 10n)^T$. As $\tilde{x}$, we use an approximate solution with almost the maximum accuracy in double precision obtained by the iterative refinement method (cf. e.g. [2, pp. 126–127]).

In Table 1, computing time for applying Methods A and B for various n, for an LU factorization (**LU**) using Gaussian elimination and for calculating R (**INV**) are displayed. The notation $\mathbf{B}(m)$ means that Method B is applied as $\mathtt{Proposal.Method}(A, \tilde{x}, b, R, m)$.

Table 1: Comparison of computing time (sec) for various n.

n	**LU**	**INV**	**A**	**B**(1)	**B**(2)	**B**(3)
100	0.1	0.4	0.03	0.04	0.06	0.09
500	0.41	2.98	1.35	1.36	1.68	2.34
1000	3.41	23.9	9.64	10.1	11.5	14.4
2000	26.7	199	74.4	75.8	82.4	95.6

Table 2: Comparison of error bounds for various cond(A) with $n = 1000$.

cond(A)	**A**	**B** (for x_1)	**B** (for x_{1000})
10^2	9.09e-13	5.00e-16	7.19e-14
10^4	9.06e-13	8.29e-18	7.05e-13
10^6	9.05e-13	8.39e-16	6.10e-14
10^8	9.04e-13	1.51e-17	2.94e-13
10^{10}	9.85e-13	2.19e-16	1.77e-13

From Table 1, it can be seen that computing time for Method B is comparable to that for Method A even if m increases. Computing time for **B**(1) is almost the same as that for Method A. Although computing time for **B**(3) is about 30% as much as that for Method A, computing time for the whole verification process strongly depends on calculating the approximate inverse R.

Next, we fix $n = 1000$ and choose A a floating-point $n \times n$ matrix with an arbitrary condition number cond(A) presented by Rump [10]. We again put $b := \mathrm{fl}(A \cdot c)$ with $c := (10, 20, \ldots, 10n)^T$. Table 2 displays the error bounds obtained by Methods A and B for the smallest solution x_1 and the largest solution x_{1000}. Of course, Method A gives the same error bound for x_1 and x_{1000}. The number of iterations m in Method B is adapted to cond(A).

Actually, when cond(A) is from 10^2 to 10^4, Method B with $m = 1$ is sufficient to obtain a tight and componentwise error bound for $\tilde{x}$. From Table 2, we can confirm that Methods A and B give a tight normwise error bound and a tight componetwise error bound for $\tilde{x}$, respectively.

In conclusion, it turns out that we can obtain a tight componentwise error bound for an approximate solution of a linear system by Algorithm 2 using Gaussian elimination with a little additional cost.

Acknowledgments

This research was partially supported by CREST program, Japan Science and Technology Agency (JST), 21st Century COE Program (Productive ICT Academia Program, Waseda University) from the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] ANSI/IEEE, *IEEE Standard for Binary Floating Point Arithmetic*, Std 754–1985 edition, IEEE, New York, 1985.

[2] G.H. Golub, C.F. Van Loan: *Matrix Computations*, 3rd edition, Johns Hopkins University Press, Baltimore and London, 1996.

[3] J. Gosling, B. Joy, G. Steele, G. Bracha: *The Java Language Specification*, 2nd edition, Addison-Wesley, 2000.

[4] MathWorks, NIST: JAMA – A Java Matrix Package. `http://math.nist.gov/javanumerics/jama/`

[5] T. Ogita, S. Oishi, Y. Ushiro: Computation of sharp rigorous componentwise error bounds for the approximate solutions of systems of linear equations, *Reliable Computing* 9:3 (2003), 229–239.

[6] T. Ogita, S.M. Rump, S. Oishi: Accurate sum and dot product, *SIAM J. Sci. Comput.*, 26:6 (2005), 1955–1988.

[7] T. Ogita, S.M. Rump, S. Oishi: Verified solution of linear systems without directed rounding, Technical Report No. 2005-04, Advanced Research Institute for Science and Engineering, Waseda University, 2005.

[8] S. Oishi, S. M. Rump: Fast verification of solutions of matrix equations. *Numer. Math.*, 90:4 (2002), 755–773.

[9] K. Ozaki, T. Ogita, S. Miyajima, S. Oishi, S.M. Rump: A method of obtaining verified solutions for linear systems suited for Java, *Journal of Computational and Applied Mathematics*, to appear.

[10] S.M. Rump: A class of arbitrarily ill-conditioned floating-point matrices, *SIAM J. Matrix Anal. Appl.*, 12:4 (1991), 645–653.

[11] S.M. Rump: Verification methods for dense and sparse systems of equations, *Topics in Validated Computations – Studies in Computational Mathematics (J. Herzberger ed.)*, 63–136, Elsevier, Amsterdam, 1994.