Japan J. Indust. Appl. Math., 26 (2009), 169–190 Area ⟨2⟩

Fast Verified Solutions of Linear Systems

Takeshi OGITA* and Shin'ichi OISHI†

*Department of Mathematical Sciences
Tokyo Woman's Christian University
2–6–1 Zempukuji, Suginami-ku, Tokyo 167–8585, Japan and
Visiting Associate Professor at
Faculty of Science and Engineering, Waseda University
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
E-mail: ogita@lab.twcu.ac.jp

†Department of Applied Mathematics
Faculty of Science and Engineering, Waseda University
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
E-mail: oishi@waseda.jp

Received June 5, 2008
Revised November 14, 2008

This paper aims to survey fast methods of verifying the accuracy of a numerical solution of a linear system. For the last decade, a number of fast verification algorithms have been proposed to obtain an error bound of a numerical solution of a dense or sparse linear system. Such fast algorithms rely on the verified numerical computation using floating-point arithmetic defined by IEEE standard 754. Some fast verification methods for dense and sparse linear systems are reviewed together with corresponding numerical results to show the practical use and efficiency of the verified numerical computation as much as possible.

Key words: verified numerical computation, verified solutions of linear systems, self-validating methods

1. Introduction

A number of mathematical problems arising in science and engineering can *numerically* be solved by using digital computers, especially using floating-point arithmetic, and then a numerical solution is obtained; for example, systems of even more than millions linear equations are solved. However, one may ask "How reliable is the numerical solution?" In such a case, a simple (and perhaps the best) answer to this question might be "The leading k decimal digits are correct." A possibility to enable us to do it is so-called *verified numerical computation*.

In this paper, we surveys the verified numerical computation for enclosing a solution of a linear system

$$Ax = b, \tag{1.1}$$

where A is a real $n \times n$ matrix and b is a real n-vector. If A is nonsingular, then there exists the unique solution $x^* := A^{-1}b$. Therefore, the verified numerical computation also aims to prove the nonsingularity of A. On the other hand, if A is singular or nearly singular, then the verified numerical computation may answer "I can't prove the nonsingularity of A." This means A might be singular, but it is not known. In general, the verified numerical computation cannot prove the singularity

of A, because it is an ill-posed problem. At least, however, it never answers a wrong result unless an unexpected error occurs on hardware or software. It is an important property of the verified numerical computation.

For the last decade, a number of *fast* algorithms (cf., for example, [20, 25, 36]) have been proposed to prove the nonsingularity of A and to compute a *normwise* error bound ϵ of $\tilde{x}$ such that

$$\|x^* - \tilde{x}\| \leq \epsilon, \tag{1.2}$$

or a *componentwise* error bound (vector) d such that

$$|x^* - \tilde{x}| \leq d. \tag{1.3}$$

Here for real n-vectors v and w, we denote by $|v|$ a nonnegative vector consisting of entrywise absolute values $|v_i|$, and an inequality $v \leq w$ is understood entrywise, i.e., $v_i \leq w_i$ for $1 \leq i \leq n$. We call a verification algorithm *fast* if the computational cost for the verification process is comparable to that for obtaining an approximate solution by a standard numerical algorithm. Such fast algorithms rely on the verified numerical computation using floating-point arithmetic defined by IEEE standard 754 [2].

In this paper, we do not treat the case where A and b are interval quantities. Strictly speaking, we focus on the case where all the entries of A and b are floating-point numbers. Although most of the discussions in the paper can be extended to not only the case of the entries being real numbers but also the case of those being (moderately narrow) interval numbers, we do not mention the details in this paper. Excellent overviews of the verified numerical computation for systems of real/interval linear equations can be found in [12, 16, 34, 26]. For further details of interval arithmetic, see [1], for example.

The rest of the paper is organized as follows: In Section 2, we state notation and definitions used in this paper. In Section 3, we introduce fast and verified numerical computations for vector and matrix operations proposed by Oishi and Rump. In Section 4, we review well-known fast and efficient verification methods for an approximate solution of a dense linear system. In Section 5, we review some useful verification methods for a sparse linear system with the coefficient matrix having such special properties as monotone (including M-matrix), H-matrix and symmetric positive definite. Finally in Section 6, we conclude the paper. Besides, some numerical results are presented suitably in each sections.

We express algorithms in Matlab-like style for readability. We want to stress that the verified numerical computation for the dense linear system has been equal to the practical task. Therefore, we try to present the practical use of fast and efficient verified numerical computation as much as possible.

2. Notation and definitions

Let $\mathbb{R}$ denote the set of real numbers. For real matrices $A = (a_{ij})$, $B = (b_{ij}) \in \mathbb{R}^{m\times n}$, we denote by $|A| = (|a_{ij}|) \in \mathbb{R}^{m\times n}$ a nonnegative matrix consisting of entry-wise absolute values, and an inequality $A \leq B$ is understood entrywise, i.e., $a_{ij} \leq b_{ij}$ for all (i, j). Moreover, the notation $A \geq O$ (or $A > O$) means that all elements of A are nonnegative (positive). For real vectors, similar notation is applied. For $p \in \{1, 2, \infty\}$ we denote p-norm of A by

$$\|A\|_1 := \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|, \quad \|A\|_2 := \sigma_{\max}(A), \quad \|A\|_\infty := \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|,$$

where $\sigma_{\max}(A)$ denotes the largest singular value of A. Then the condition number of A is defined by

$$\mathrm{cond}_p(A) := \|A\|_p \|A^{-1}\|_p \quad \text{for } p \in \{1, 2, \infty\}.$$

We denote the spectral radius of A by $\rho(A)$.

Throughout this paper, $\boldsymbol{e}$ and $\boldsymbol{o}$ denote $\boldsymbol{e} := (1, \ldots, 1)^{\mathrm{T}}$ and $\boldsymbol{o} := (0, \ldots, 0)^{\mathrm{T}}$ with length n, respectively, and I denotes the $n \times n$ identity matrix.

Let $\mathbb{IR}$ denote the set of interval real numbers. Interval quantities are written in brackets. For example, we denote an interval matrix including A by $[A] := [\underline{A}, \overline{A}] \in \mathbb{IR}^{n\times n}$ where $\underline{A}$ and $\overline{A}$ is a lower and an upper bound of A, respectively.

2.1. Special matrices

We collect here some definitions of matrices and lemmas:

Definition 2.1 (monotone). *A matrix $A \in \mathbb{R}^{n\times n}$ is called monotone if $Av \geq \boldsymbol{o}$ for $v \in \mathbb{R}^n$ implies $v \geq \boldsymbol{o}$.*

Definition 2.2 (M-matrix). *Let $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ with $a_{ii} > 0$ and $a_{ij} \leq 0$ for $i \neq j$. Then A is called an M-matrix if A is nonsingular and $A^{-1} \geq O$.*

Definition 2.3 (H-matrix). *A is called an H-matrix if $\mathcal{M}(A)$ is an M-matrix.*

Definition 2.4 (positive definite). *A matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if $v^{\mathrm{T}}Av > 0$ for all $v \neq \boldsymbol{o} \in \mathbb{R}^n$.*

From Definition 2.4, all eigenvalues of a symmetric positive definite matrix are positive.

Lemma 2.5. *A is monotone if and only if A is nonsingular with $A^{-1} \geq O$.*

Lemma 2.6. *A is an H-matrix if and only if there exists a vector $v > \boldsymbol{o}$ such that $\mathcal{M}(A)v > \boldsymbol{o}$.*

Lemma 2.7. *If A is an H-matrix, then $|A^{-1}| \leq \mathcal{M}(A)^{-1}$.*

From Lemma 2.7, it follows that

$$\|A^{-1}\|_\infty \le \|\mathcal{M}(A)^{-1}\|_\infty. \tag{2.1}$$

These special matrices does not appear until Section 4.3.

2.2. Some properties of floating-point arithmetic

Let $\mathbb{F}$ be a set of floating-point numbers following IEEE standard 754. Let $\mathbf{u}$ and $\underline{\mathbf{u}}$ be the unit-roundoff and the underflow unit, respectively. Let $\underline{\mathbf{u}}_{\mathrm{N}}$ be the smallest positive normalized floating-point number in $\mathbb{F}$. For example, $\mathbf{u} = 2^{-53}$, $\underline{\mathbf{u}} = 2^{-1074}$ and $\underline{\mathbf{u}}_{\mathrm{N}} = 2^{-1022}$ in IEEE 754 double precision arithmetic. $\mathbb{F}$ is symmetric, i.e., $x \in \mathbb{F} \implies -x \in \mathbb{F}$, so that $|x|$ is exact for $x \in \mathbb{F}$. The following four types of rounding mode are defined in IEEE standard 754: Let $c \in \mathbb{R}$.

round-to-nearest, ties to even (default): Round c to the nearest floating-point number $\tilde{f} \in \mathbb{F}$ satisfying $|\tilde{f} - c| = \min_{f\in\mathbb{F}}|f - c|$. We represent it by $\mathrm{fl}_\square(c) = \tilde{f}$.

round-downwards: Round c to the largest floating-point number $\underline{f} \in \mathbb{F}$ satisfying $\underline{f} \le c$. We represent it by $\mathrm{fl}_\triangledown(c) = \underline{f}$.

round-upwards: Round c to the smallest floating-point number $\overline{f} \in \mathbb{F}$ satisfying $\overline{f} \ge c$. We represent it by $\mathrm{fl}_\triangle(c) = \overline{f}$.

round-towards-zero: Round c to one of the floating-point neighbors $\hat{f} \in \mathbb{F}$ satisfying $|\hat{f}| \le |c|$. We represent it by $\mathrm{fl}_\lozenge(c) = \hat{f}$.

The floating-point arithmetic following IEEE standard 754 is defined as follows: Let $x, y \in \mathbb{F}$. For $\circ \in \{+, -, *, /\}$ and $\bigcirc \in \{\square, \triangle, \triangledown, \lozenge\}$

$$\mathrm{fl}_\bigcirc(x \circ y) := \mathrm{fl}_\bigcirc(z) \quad \text{where } z = x \circ y \in \mathbb{R} \tag{2.2}$$

and

$$\mathrm{fl}_\bigcirc(\sqrt{x}) := \mathrm{fl}_\bigcirc(z) \quad \text{where } z = \sqrt{x} \in \mathbb{R}, \tag{2.3}$$

which also hold in the presence of underflow. Therefore, (2.2) and (2.3) are mathematically reliable statements.

Here we want to stress that we should carefully implement algorithms using floating-point arithmetic, especially in case where the algorithm is sensitive to its behavior, in terms of not only the programming code itself but also the compiler, the compile options and the processors in use:

> "So the mathematical specification of the algorithms for floating-point arithmetic can be correct and never-failing, but the implementation might not follow the specification. We assume in the following that the implementation of floating-point arithmetic follows its specification and therefore the IEEE 754 standard, and that the software and hardware in use are operating correctly. Then, we may ask again whether it is possible to validate results with the aid of digital computers." (Siegfried M. Rump [29])

In fact, we sometimes encounter a problem of the "optimization" overdone by the compiler. For example, the statement $(a+b)-a$ in C or Fortran code may be simplified to just b by some compilers with some compile options for higher level optimization. Of course, it does not necessarily hold that $\mathrm{fl}_{\bigcirc}((a+b)-a) = b$ since $\mathrm{fl}_{\bigcirc}(a+b) \neq a+b$ in general. Moreover, a wider register, e.g., Intel's floating-point 80-bit register, also affects this kind of computations which require the consistency of IEEE 754 arithmetic, although such a wider register usually tends to improve the result accuracy. Using some special compile options[1] for obeying the consistency of IEEE standard 754, we can normally avoid changing the intended code and using the wider register, but it may significantly slow down the computational speed.

According to the Higham's notation [10], a constant γ_n is defined by $\gamma_n := \frac{n\mathbf{u}}{1-n\mathbf{u}}$, which is very useful for rounding error analysis.

3. Vectorized interval arithmetic

In [31, 23, 25], Oishi and Rump proposed fast and verified numerical computations for vector and matrix operations, i.e., dot product, matrix-vector product and matrix-matrix product with result verification, which can be regarded as vectorized versions of interval arithmetic.

First, we present a classical interval approach adapted to dot product $x^{\mathrm{T}}y$ for $x, y \in \mathbb{F}^n$:

Algorithm 3.1. *A classical interval algorithm for an inclusion of dot product $x^{\mathrm{T}}y$ where $x, y \in \mathbb{F}^n$.*

---

function $[s] =$ `IncDot`(x, y)
  $\underline{s} = 0$; $\overline{s} = 0$;
  for $i = 1 : n$
    `setround`(-1);
    $\underline{s} = \mathrm{fl}_{\bigtriangledown}(\underline{s} + x_i * y_i)$; % *lower bound of* $x^{\mathrm{T}}y$
    `setround`$(+1)$;
    $\overline{s} = \mathrm{fl}_{\bigtriangleup}(\overline{s} + x_i * y_i)$; % *upper bound of* $x^{\mathrm{T}}y$
  end for % $[s] = [\underline{s}, \overline{s}]$

Here the instructions `setround`(-1) and `setround`$(+1)$ mean to adopt the round-downward mode and the round-upward mode, respectively. We assume that once the rounding mode is changed, it remains unchanged until the next instruction `setround` appears. This assumption is ensured on a wide class of computer systems following IEEE standard 754. Algorithm 3.1 requires $2n$ flops (floating-point operations) and $2n$ switches of the rounding modes. A switch of rounding mode costs relatively large compared to a floating-point operation.

[1] For example, the options `-fp-model source -pc64` are available for double precision arithmetic strictly following IEEE standard 754 in Intel compilers.

On the other hand, Oishi and Rump promote the following vectorized inclusion algorithm for dot product $x^{\mathrm{T}}y$:

Algorithm 3.2. *A vectorized version of an inclusion of dot product* $x^{\mathrm{T}}y$ *where* $x, y \in \mathbb{F}^n$.

---

function $[s] = \texttt{FastIncDot}(x, y)$
  $\texttt{setround}(-1)$*;*
  $\underline{s} = \mathrm{fl}_{\bigtriangledown}(x^{\mathrm{T}}y)$*; % lower bound of* $x^{\mathrm{T}}y$
  $\texttt{setround}(+1)$*;*
  $\overline{s} = \mathrm{fl}_{\bigtriangleup}(x^{\mathrm{T}}y)$*; % upper bound of* $x^{\mathrm{T}}y$

Algorithm 3.2 requires $2n$ flops and only two switches of the rounding modes, so that it is much more efficient than Algorithm 3.1. Note that the quality of the inclusion by Algorithm 3.2 is almost the same as or even better than that by Algorithm 3.1, because Algorithm 3.2 can benefit from an internal extended precision register, e.g., 80 bits in IA-32 architecture, while Algorithm 3.1 cannot.

We can also extend this idea to a fast inclusion of matrix multiplication:

Algorithm 3.3. *A vectorized version of an inclusion of matrix multiplication* $A \cdot B$ *where* $A \in \mathbb{F}^{m\times p}$ *and* $B \in \mathbb{F}^{p\times n}$.

---

function $[C] = \texttt{FastIncMM}(A, B)$
  $\texttt{setround}(-1)$*;*
  $\underline{C} = \mathrm{fl}_{\bigtriangledown}(A * B)$*; % lower bound of* $A \cdot B$
  $\texttt{setround}(+1)$*;*
  $\overline{C} = \mathrm{fl}_{\bigtriangleup}(A * B)$*; % upper bound of* $A \cdot B$

The main advantage of Algorithm 3.3 against a classical approach such as Algorithm 3.1 is the availability of BLAS [6], especially an optimized BLAS for the architecture in use, which aims to be as fast as possible in terms of measured computing time. For matrix multiplication, using the Level-3 BLAS impacts on computational speed very much, because such an optimized BLAS is designed to achieve near-peak performance and to be efficiently parallelized for both shared and distributed memory systems.

Remark 3.4. We have to be careful in the implementation of the black-box type of optimized BLAS, because some of them may use a special method such as Strassen's fast matrix multiplication algorithm [39], they may change the internal computational precision, or the switch of rounding mode may not work in them correctly.

To see the difference of the computational speed between Algorithm 3.3 and the classical interval approach, we present a numerical example: Let $A, B \in \mathbb{F}^{n\times n}$. Table 3.1 displays results of computing time (sec) and MFlops for calculating an approximation $\tilde{C} = \mathrm{fl}(A \cdot B)$ by `DGEMM` in BLAS, an inclusion $[C] = [\underline{C}, \overline{C}]$ such

Table 3.1. Computing time (sec) and MFlops (inside parenthesis) for an approximation (by BLAS `DGEMM`) and an inclusion (by Algorithm 3.3 and by Classical interval, resp.) of matrix multiplication with Intel Core 2 Extreme X9650 (3.0 GHz, quad-core), GNU Compiler Collection 4.12 (gcc and gfortran) and GotoBLAS v1.25; peak performance: 48 GFlops $= 48 \cdot 10^3$ MFlops.

n	BLAS (`DGEMM`)	Algorithm 3.3	Classical interval
500	$1.32 \cdot 10^{-2}$	$2.66 \cdot 10^{-2}$	7.71
	$(18.9 \cdot 10^3)$	$(18.8 \cdot 10^3)$	(65)
1000	$5.09 \cdot 10^{-2}$	0.10	62.55
	$(39.3 \cdot 10^3)$	$(39.1 \cdot 10^3)$	(64)
2000	0.37	0.75	495.2
	$(42.9 \cdot 10^3)$	$(42.7 \cdot 10^3)$	(65)
5000	5.66	11.38	7892
	$(44.2 \cdot 10^3)$	$(43.9 \cdot 10^3)$	(63)
10000	44.73	89.45	$>$ half a day
	$(44.7 \cdot 10^3)$	$(44.7 \cdot 10^3)$	(—)

that $\underline{C} \leq A \cdot B \leq \overline{C}$ by Algorithm 3.2, which also utilizes `DGEMM`, and that by a classical interval approach similar to Algorithm 3.1, respectively. Here we use a PC with Intel Core 2 Extreme (3.0 GHz, quad-core) and GNU Compiler Collection 4.12 (gcc and gfortran). In addition, we adopt GotoBLAS v1.25 [7, 8] as the optimized BLAS with parallelization. Fortunately, GotoBLAS is now distributed as source code (though written in inline assembler code), from which we can confirm that there is no problem concerning Remark 3.4. Note that we implement a straightforward and serial code for the classical interval approach, which does not use any parallelization, because it even slows down the computational speed compared to the serial implementation, so that we decided to stop doing it.

From Table 3.1, we can observe that the performance of Algorithm 3.3 is very high since it relies on that of the optimized BLAS. For larger n, it is unrealistic to adopt the classical interval approach.

We stress that such an approach as Algorithms 3.2 and 3.3 is very suited for Matlab implementation in terms of not only readability of the algorithm but also its computational speed, because a classical interval approach like Algorithm 3.1 significantly suffers from interpretation overhead of Matlab.

4. Verification methods for dense linear systems

In this section, we will review some methods of calculating an error bound of an approximate solution $\tilde{x}$ of a dense linear system $Ax = b$ where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$. These methods are widely used in verified numerical computations.

4.1. Rump' theorem

First, we present Rump's approach [32, 34], which is one of the standard verification methods based on the Krawczyk operator [11, 13]:

$$K([x]) := \tilde{x} + R(b - A\tilde{x}) + (I - RA)[x],$$

where $\tilde{x} \in \mathbb{R}^n$ denotes an approximate solution of $Ax = b$, R denotes an approximate inverse of A, and $[x] \in \mathbb{IR}^n$ an interval n-vector supposed to be a potential inclusion of the solution $A^{-1}b$. If $\|I - RA\| < 1$ for some norm, then $K([x]) \subseteq \mathrm{int}[x]$ implies $A^{-1}b \in [x]$, where $\mathrm{int}([x])$ denote the interior of $[x]$ (cf., e.g., [16]).

Theorem 4.1 (Rump [32, 34]). *Let $A \in \mathbb{R}^{n\times n}$, $R \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ and $\tilde{x} \in \mathbb{R}^n$ be given. Let $[\epsilon] \in \mathbb{IR}^n$ be closed and bounded with $[\epsilon] \neq \emptyset$. Let $\mathrm{int}([\epsilon])$ denote the interior of $[\epsilon]$. If*

$$[y] := R(b - A\tilde{x}) + (I - RA)[\epsilon] \subseteq \mathrm{int}([\epsilon]), \tag{4.1}$$

then A and R are nonsingular and the unique solution $x^* = A^{-1}b$ of $Ax = b$ satisfies $x^* \in \tilde{x} + [y]$.

Rump proved that if $\rho(|I - RA|) < 1 - \delta$ for some $\delta > 0$, then (4.1) holds for some $[\epsilon]$. Conversely, if (4.1) holds, then $\rho(|I - RA|) < 1$. In [32], Rump devised the "epsilon-inflation" to obtain a valid $[\epsilon]$ together with an iteration scheme for (4.1):

Algorithm 4.2. *Rump's method of calculating a verified solution of a linear system $Ax = b$ with epsilon-inflation.*

$[G] = [I - RA]$; % *inclusion of* $I - RA$
$[z] = [R(b - A\tilde{x})]$; % *inclusion of* $R(b - A\tilde{x})$
$[y^{(0)}] = [z]$;
repeat $k = 1, 2, \ldots$
  $[\epsilon^{(k)}] = \varepsilon_{\mathrm{inflation}}([y^{(k-1)}], \varepsilon)$; % *epsilon-inflation*
  $[y^{(k)}] = [z] + [G] \cdot [\epsilon^{(k)}]$; % *corresponding to* (4.1)
until $[y^{(k)}] \subseteq \mathrm{int}([\epsilon^{(k)}])$

If the iteration in this algorithm stops, then A is nonsingular and $A^{-1}b \in \tilde{x} + [y^{(k)}]$.

The epsilon-inflation is defined by

$$\varepsilon_{\mathrm{inflation}}([y], \varepsilon) := [y] + [-\varepsilon, \varepsilon] \cdot \mathrm{diam}([y]) + [-u_\eta, u_\eta], \tag{4.2}$$

where $\mathrm{diam}([y])$ denotes the diameter of $[y]$, $\varepsilon \geq 0$ and $u_\eta > \boldsymbol{o}$. In [27], Rump noted that the small *absolute* term $[-u_\eta, u_\eta]$ is necessary for the correct behavior of the epsilon-inflation. According to his experience, he suggested to set $\varepsilon = 0$ and

$$u_\eta = 0.1 \cdot |R(b - A\tilde{x})| + \underline{\mathbf{u}}_{\mathrm{N}} \cdot \boldsymbol{e}, \tag{4.3}$$

which is obtained from the heuristics by replacing diam($[y]$) by diam($[y^0]$) in (4.2), and by setting $\varepsilon = 0.1$ and $u_\eta = \underline{\mathbf{u}}_{\mathrm{N}} \cdot \boldsymbol{e}$. In this case, the inflation part consists of nothing but the absolute term.

A verification method based on the above-mentioned approach is implemented as part of the function `verifylss` in INTLAB [31], a fast and efficient interval toolbox for Matlab. For more details of the iteration scheme and the epsilon-inflation for (4.1), see [32, 34, 27].

Thus if interval operations are used in the above iteration scheme and $\rho(|I-RA|) < 1-\delta$ for some $\delta > 0$ involving the effect of rounding errors, then it will eventually converge for some k. This method has been implemented in INTLAB as the first stage of `verifylss` for dense linear (interval) systems.

We present the following numerical example using Matlab with INTLAB's `verifylss` on our laptop PC, which is not intended to show the efficiency of the algorithm but the fact that the verified numerical solution can easily be obtained in this case:

```
>> n=1000; A=randn(n); b=A*ones(n,1);
>> tic; x=verifylss(A,b); toc, max_rel_err=max(abs(x.rad./x.mid))
Elapsed time is 2.699914 seconds.
max_rel_err =
   6.6613e-16
```

From this result, we can know

- `A` is nonsingular,
- it rigorously holds that `x.mid` $-$ `x.rad` $\le$ `A`$^{-1}$`b` $\le$ `x.mid` $+$ `x.rad`, and
- each component of the numerical solution `x.mid` has at least 15 correct decimal digits.

Thus in this case, the verified numerical computation clearly answered the first question of the paper.

4.2. Yamamoto's theorem

An alternative approach was proposed by Yamamoto [41]. We present in the following a linearized version of Yamamoto's theorem.

THEOREM 4.3 (Yamamoto [41]). *Let $A \in \mathbb{R}^{n\times n}$, $R \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ and $\tilde{x} \in \mathbb{R}^n$ be given. If $\|I - RA\|_p < 1$ for any $p \in \{1, 2, \infty\}$, then A is nonsingular and*

$$|A^{-1}b - \tilde{x}| \le |R(b - A\tilde{x})| + \frac{\|R(b - A\tilde{x})\|_p}{1 - \|I - RA\|_p}|I - RA|\boldsymbol{e}. \tag{4.4}$$

An advantage of applying this theorem is that it does not need the interval iteration process required for (4.1). In other words, if Algorithm 4.2 is executed in infinite precision as $k \to \infty$, a similar estimate can be formulated from Theorem 4.1.

To apply Theorem 4.3, it is necessary to prove $\|I - RA\|_p < 1$ for some p. In practice, we aim to compute an upper bound α of $\|I - RA\|_\infty$ and check $\alpha < 1$ to prove whether A is nonsingular, which requires the main computational effort in the verification process. On the other hand, we see from (4.4) that any value

α less than 1/2, say, is sufficient for a reasonable error estimation. So not too much effort must be spent on estimating α. For this purpose, several verification methods have been proposed. For example, the following algorithm tries to prove the nonsingularity of A.

Algorithm 4.4 (cf. Oishi–Rump [25]). *Rigorous test for the nonsingularity of $A \in \mathbb{F}^{n\times n}$. If* res $= 1$, *A is proved to be nonsingular. Otherwise, the algorithm cannot prove it.*

---

function res = `IsNonsingular`(A)
  $R =$ `inv`(A)*; % approximate inverse of A*
  `setround`(-1)*;*
  $\underline{C} = \mathrm{fl}_{\triangledown}(R * A - I)$*; % lower bound of $RA - I$*
  `setround`$(+1)$*;*
  $\overline{C} = \mathrm{fl}_{\triangle}(R * A - I)$*; % upper bound of $RA - I$*
  $\overline{C} = \max(|\underline{C}|, |\overline{C}|)$*; % upper bound of $|I - RA|$*
  $\alpha = \mathrm{fl}_{\triangle}(\|\overline{C}\|_\infty)$*; % upper bound of $\|I - RA\|_\infty$*
  if $\alpha < 1$, res $= 1$*; else* res $= 0$*; end*

Note that it is not valid to compute $\underline{C} = \mathrm{fl}_{\triangledown}(I - R * A)$ instead, because $\underline{C}$ does not necessarily become a lower bound of $I - RA$, i.e., $\underline{C} = \mathrm{fl}_{\triangledown}(I - (\text{lower bound of } RA))$. Let us assume that LU factors of A have been computed by Gaussian elimination with partial pivoting as usual, which requires $\frac{2}{3}n^3$ flops. Then Algorithm 4.4 requires $\frac{16}{3}n^3$ flops ($\frac{4}{3}n^3$ flops for calculating an approximate inverse R using the LU factors and $4n^3$ flops for calculating the upper bound of $|I - RA|$).

In case of A being a general full matrix, the fastest known method of calculating an upper bound of $\|I - RA\|_\infty$ was proposed by Oishi and Rump [25]. The fastness of Oishi–Rump method relies on a priori error bounds by backward error analysis for the LU factorization of A and for solving triangular matrix equations: Assume that computed LU factors L, U and P such that $PA \approx LU$ are obtained by a standard numerical algorithm. Then the following estimate holds [10]:

$$|PA - LU| \le \gamma_n |L|\,|U| \tag{4.5}$$

Moreover, for a given (lower or upper) triangular matrix $T \in \mathbb{F}^{n\times n}$, assume that a triangular matrix equation $XT = I$ is solved by standard (floating-point) forward/backward substitution and then an approximate solution $X_T \in \mathbb{F}^{n\times n}$ is obtained. Then the following estimate also holds [10]:

$$|I - X_T T| \le \gamma_n |X_T|\,|T| \tag{4.6}$$

Let X_L and X_U be approximate inverses of L and U, respectively. To compute each of them requires $\frac{1}{3}n^3$ flops. Set $R := X_U X_L P$. Using (4.5) and (4.6) yields

$$\begin{aligned}
|I - RA| &= |I - X_U X_L PA| = |I - X_U X_L (PA - LU + LU)| \\
&\le |I - X_U X_L LU| + |X_U X_L (PA - LU)| \\
&\le |I - X_U (I - I + X_L L)U| + |X_U|\,|X_L|\,|PA - LU| \\
&\le |I - X_U U| + |X_U (I - X_L L)U| + |X_U|\,|X_L|\,|PA - LU| \\
&\le \gamma_n |X_U|\,|U| + 2\gamma_n |X_U|\,|X_L|\,|L|\,|U|.
\end{aligned}$$

Hence we have

$$\begin{aligned}
\|I - RA\|_\infty &\le \gamma_n \big(2 \big\| |X_U|\,|X_L|\,|L|\,|U| \big\|_\infty + \big\| |X_U|\,|U| \big\|_\infty \big) \\
&= \gamma_n \big(2 \big\| |X_U| (|X_L| (|L| (|U| \boldsymbol{e}))) \big\|_\infty + \big\| |X_U| (|U| \boldsymbol{e}) \big\|_\infty \big).
\end{aligned}$$

In [25], the presence of underflow is also taken into account and

$$\|I - RA\|_\infty \le \gamma_n \big(2 \big\| |X_U| (|X_L| (|L| (|U| \boldsymbol{e}))) \big\|_\infty + \big\| |X_U| (|U| \boldsymbol{e}) \big\|_\infty \big) + c_1 \underline{\mathbf{u}}, \quad (4.7)$$

where c_1 is some computable factor. To compute (an upper bound of) the right-hand side of (4.7) requires only $\mathcal{O}(n^2)$ flops. Thus Oishi–Rump method requires $\frac{2}{3}n^3$ flops in total, the same computational effort for calculating an approximate solution by Gaussian elimination with partial pivoting. Namely, Oishi–Rump method is 8 times faster than Algorithm 4.4.

Another possibility is the following approach by the authors [18]:

$$\begin{aligned}
|I - RA| &= |I - X_U X_L PA| = |I - X_U U - X_U (X_L PA - U)| \\
&\le |I - X_U U| + |X_U|\,|X_L PA - U| \\
&\le \gamma_n |X_U|\,|U| + |X_U|\,|X_L PA - U| \\
&= |X_U| (|X_L PA - U| + \gamma_n |U|)
\end{aligned}$$

Involving the presence of underflow yields

$$\|I - RA\|_\infty \le \big\| |X_U| ((|X_L PA - U| + \gamma_n |U|) \boldsymbol{e}) \big\|_\infty + c_2 \underline{\mathbf{u}}, \quad (4.8)$$

where c_2 is also some computable factor. This approach requires $\frac{8}{3}n^3$ flops after obtaining the LU factors.

On the other hand, the fastness of the verification method produces a side effect on its robustness, i.e., there is a trade-off between the fastness and the robustness. To see it, we present numerical examples using `randsvd` from Higham's test matrices [10] on Matlab:

```
RANDSVD Random matrix with pre-assigned singular values.
  A = GALLERY('RANDSVD', N, KAPPA, MODE, KL, KU) is a banded random
  matrix of order N with COND(A) = KAPPA and singular values from the
  distribution MODE. If N is a two-element vector, A is N(1)-by-N(2).
```

```
MODE may be one of the following values:
    1: one large singular value,
    2: one small singular value,
    3: geometrically distributed singular values,
    4: arithmetically distributed singular values,
    5: random singular values with uniformly distributed logarithm.
```

We compare the quality of the upper bound α of $\|I - RA\|_\infty$ by Algorithm 4.4, Oishi–Rump method [25] based on (4.7) and Ogita–Oishi method [18] based on (4.8). We vary $\mathtt{N} = 8, 16, \ldots, 1024$ with setting $\mathtt{KAPPA} = 10^6 \approx \mathrm{cond}_2(A)$ for $\mathtt{MODE} = 1, 2, \ldots, 5$. The parameters `KL` and `KU` are omitted, so that A is generated as a full matrix. The results are displayed in Fig. 4.1. Here we do not display the case $\mathtt{MODE} = 4$ since the result for $\mathtt{MODE} = 4$ is quite similar to that for $\mathtt{MODE} = 2$.

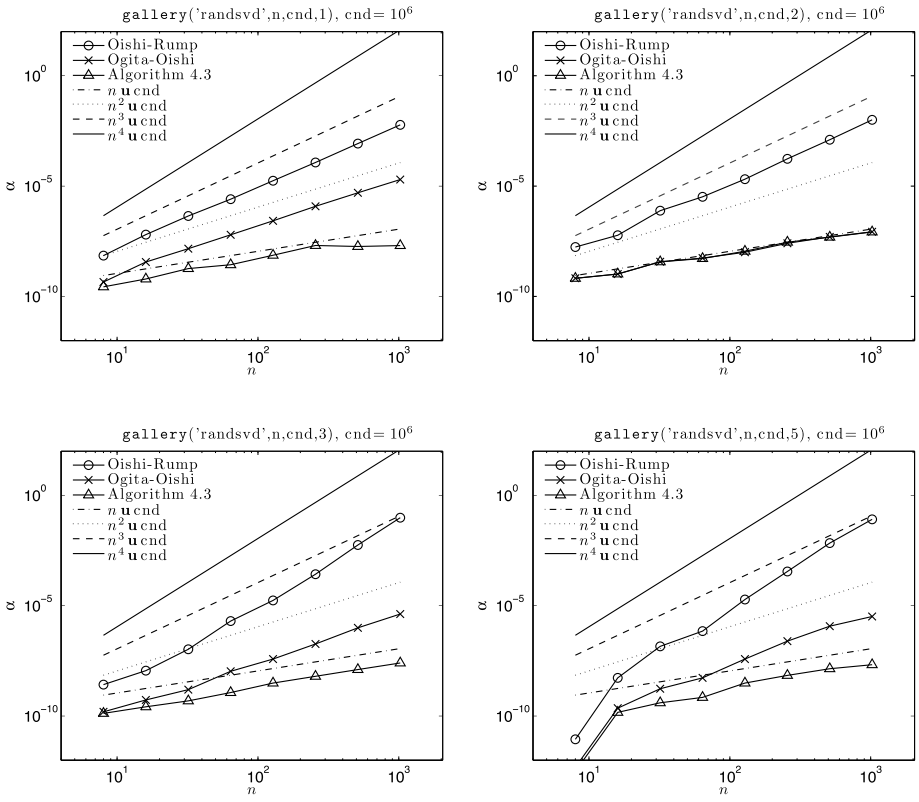

Fig. 4.1. Estimates for $\|I - RA\|_\infty$ by several verification methods for Higham's test matrix `randsvd` with $\mathrm{cond}_2(A) = 10^6$ and $\mathtt{MODE} = 1, 2, 3, 5$.

From Fig. 4.1, we can empirically observe that the applicable range of each verification method is limited to

$$\mathrm{cond}_2(A) \lesssim \mathcal{O}(n^{-k})\mathbf{u}^{-1} < \mathbf{u}^{-1} \tag{4.9}$$

according to the following table:

Verification method	(Empirical) k in (4.9)	Flops
Oishi–Rump method [25] based on (4.7)	$3 \le k \le 4$	$\frac{2}{3}n^3$
Ogita–Oishi method [18] based on (4.8)	$1 \le k \le 2$	$\frac{8}{3}n^3$
Algorithm 4.4	1	$\frac{16}{3}n^3$

Note that we can seamlessly switch the verification method from Oishi–Rump method to Ogita–Oishi method even if the former failed in verification, so that we can easily combine both of the methods. We now call it Oishi–Rump–Ogita method.

The above-mentioned verification methods can be used together with Theorem 4.3. To compare the performance of the verification methods for linear systems based on Algorithm 4.4 and Oishi–Rump–Ogita method, we present numerical examples[2] using the same architecture as before and Matlab 2007b. Let $A \in \mathbb{F}^{n\times n}$ be generated by a Matlab's built-in function `randn` as a matrix containing pseudo-random values drawn from a normal distribution with mean zero and standard deviation one. Let $b := \mathrm{fl}_\square(A \cdot \boldsymbol{e})$. We vary n from 2,000 to 10,000. In each case, we observe $\mathrm{cond}_2(A) \approx 10^4$. To compute a numerical solution $\tilde{x}$ of a linear system $Ax = b$, we use a standard numerical algorithm via an LU factorization of A.

In Table 4.1, we display the computing time for calculating $\tilde{x}$ (labeled "Approx.") and its verification based on Oishi–Rump–Ogita method and Algorithm 4.4, respectively. The ratio of computing time for "Approx." to each verification method is also shown inside parenthesis by "Approx." being normed to one. Here the column labeled "Oishi–Rump–Ogita (MEX)" displays the results by the same algorithm as "Oishi–Rump–Ogita," which uses our compiled MEX-functions for executing *triangular matrix multiplication* (n^3 flops) and *triangular matrix inversion* ($n^3/3$ flops) based on BLAS and LAPACK routines. The reason why we use such MEX-functions is that they have still not been implemented as the built-in functions on the latest Matlab 2008a, unfortunately.[3] Without using the MEX-functions as in labeled "Oishi–Rump–Ogita," they have to be executed as *full matrix multiplication* ($2n^3$ flops) and *triangular matrix equation* (n^3 flops), respectively; of course, it is not optimal. We can confirm the efficiency of the MEX-functions from Table 4.1. In the numerical examples, $n = 8000$ is the switch point from Oishi–Rump method to Ogita–Oishi method, so that the ratio of the computing time suddenly increases from this point.

In Table 4.2, we also display an upper bound of the maximum relative error $\max_{1\le i\le n}|x_i^* - \tilde{x}_i|/|x_i^*|$ for $x^* := A^{-1}b$ by Yamamoto's theorem applying each verification method. Here we do not use any higher precision arithmetic nor the iterative refinement of the numerical solution. Nevertheless, we can obtain the numerical solutions with their error bounds after verifying the nonsingularity of A. Although the obtained error bounds are a little pessimistic compared to the exact

[2]Thanks to Dr. Katsuhisa Ozaki for helping with the numerical experiments.

[3]It seems to be not so difficult to incorporate them into Matlab since the built-in optimized BLAS and LAPACK library (Intel Math Kernel Library) used in Matlab has already included them, so the authors desire them to be available for keeping the portability of the Matlab-code.

Table 4.1. Computing time (sec) for calculating numerical solutions (labeled "Approx.") and their verifications with Intel Core 2 Extreme X9650 (3.0 GHz, quad-core) and Matlab 2007b.

n	Approx.	Oishi–Rump–Ogita	Oishi–Rump–Ogita (MEX)	Algorithm 4.4
2,000	0.38	0.99 (2.6)	0.62 (1.6)	1.72 (4.5)
4,000	2.17	5.27 (2.4)	3.57 (1.6)	11.3 (5.2)
6,000	6.17	15.1 (2.4)	10.6 (1.7)	35.5 (5.8)
8,000	13.3	83.4 (6.3)	58.9 (4.4)	76.9 (5.8)
10,000	24.1	158 (6.6)	109 (4.5)	157 (6.5)

Table 4.2. Error bounds of numerical solutions of linear systems with random matrices.

n	Oishi–Rump–Ogita	Algorithm 4.4
2,000	$2.26 \cdot 10^{-8}$	$7.55 \cdot 10^{-10}$
4,000	$1.69 \cdot 10^{-7}$	$4.79 \cdot 10^{-9}$
6,000	$1.15 \cdot 10^{-6}$	$6.22 \cdot 10^{-9}$
8,000	$1.13 \cdot 10^{-6}$	$8.52 \cdot 10^{-9}$
10,000	$2.08 \cdot 10^{-6}$	$2.49 \cdot 10^{-8}$

error, we can improve such an overestimation with not so much computational effort. For more details of the tight estimation, see [19, 21].

4.3. Other methods

To compute a lower bound $\underline{\sigma}$ of the smallest singular value $\sigma_{\min}(A)$ of A is also useful for the verified solution of $Ax = b$, e.g.,

$$\begin{aligned}\|A^{-1}b - \tilde{x}\|_\infty \le \|A^{-1}b - \tilde{x}\|_2 \le \|A^{-1}\|_2 \|b - A\tilde{x}\|_2 &= \frac{\|b - A\tilde{x}\|_2}{\sigma_{\min}(A)} \\ &\le \frac{\|b - A\tilde{x}\|_2}{\underline{\sigma}}.\end{aligned}$$

However, this approach frequently overestimates the error $\|A^{-1}b - \tilde{x}\|_\infty$ because separating the norm causes the worst case estimation and the quality of $\underline{\sigma}$ directly influences that of the error estimation. To reduce the overestimation, we store the approximate solution in two parts $\tilde{x}$ and $\tilde{y}$. This approach was used in Rump's Ph.D. thesis [32] and later called "staggered correction." Then

$$\begin{aligned}|A^{-1}b - \tilde{x}| &= |A^{-1}b - (\tilde{x} + \tilde{y} - \tilde{y})| \le |\tilde{y}| + |A^{-1}b - (\tilde{x} + \tilde{y})| \\ &\le |\tilde{y}| + \|A^{-1}b - (\tilde{x} + \tilde{y})\|_\infty \boldsymbol{e}.\end{aligned}$$

Hence

$$|A^{-1}b - \tilde{x}| \le |\tilde{y}| + \|A^{-1}\|_p \|b - A(\tilde{x} + \tilde{y})\|_p \boldsymbol{e} \quad \text{for } p \in \{1, 2, \infty\}.$$

From this, we obtain the following theorem.

THEOREM 4.5 (Ogita–Oishi–Ushiro [21]). *Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$ and $\tilde{y} \in \mathbb{R}^n$ be given. Suppose $\|A^{-1}\|_p \le \tau$ for any $p \in \{1, 2, \infty\}$. Then*

$$|A^{-1}b - \tilde{x}| \le |\tilde{y}| + \tau\|b - A(\tilde{x} + \tilde{y})\|_p \boldsymbol{e}. \tag{4.10}$$

Usually, we take $\tilde{y}$ as an approximate solution of a linear system $Ay = r$ where $r := b - A\tilde{x}$. We see from (4.10) that the term $\|b - A(\tilde{x} + \tilde{y})\|_p$ can arbitrarily be decreased if $\tilde{y}$ approaches to the exact error of $\tilde{x}$. Note that the technique makes only sense when a more accurate dot product is available, which is necessary to compute accurate residuals $b - A\tilde{x}$ and $b - A(\tilde{x} + \tilde{y})$. For further details of fast and efficient algorithms for accurate dot product, see [22, 37, 38].

The Hansen–Bliek–Rohn–Ning–Kearfott enclosure [14] for linear interval equations whose coefficient matrix is an H-matrix can also be used. For $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, we define the comparison matrix $\mathcal{M}(A) = (\hat{a}_{ij})$ of A as

$$\hat{a}_{ij} = \begin{cases} |a_{ij}| & (i = j), \\ -|a_{ij}| & (i \ne j). \end{cases}$$

A simplified version of the Ning–Kearfott theorem [17] for non-interval inputs A and b is as follows:

THEOREM 4.6 (Ning–Kearfott [17]). *Let an H-matrix $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given. Let $y, z \in \mathbb{R}^n$ be defined by*

$$y := \mathcal{M}(A)^{-1}|b| \quad \textit{and} \quad z_i := [\mathcal{M}(A)^{-1}]_{ii}.$$

Let $p, q \in \mathbb{R}^n$ be defined by

$$p_i := [\mathcal{M}(A)]_{ii} - z_i \quad \textit{and} \quad q_i := y_i/z_i - |b_i|.$$

Then $A^{-1}b \in [x]$ where

$$[x_i] := \frac{b_i + [-q_i, q_i]}{A_{ii} + [-p_i, p_i]}.$$

To apply this theorem, it must be known in advance that A is an H-matrix. For example, by preconditioning A as RA where R is some preconditioner (usually an approximate inverse of A), RA is expected to be an H-matrix. Then we try to find a vector $v > \boldsymbol{o}$ which satisfies $RAv > \boldsymbol{o}$. If such a vector v is found, then a linear system $RAx = Rb$ is considered. This method has also been implemented in INTLAB as the second stage of `verifylss` for dense linear (interval) systems. In the INTLAB-code `verifylss.m`, it is mentioned that the results of Hansen–Bliek–Rohn–Ning–Kearfott enclosure may be of better quality (than that of Rump's approach in Section 4.1) for extremely ill-conditioned linear systems; normally the quality is similar.

If $\mathrm{cond}_2(A) \geq \mathbf{u}^{-1}$, then all the verification methods mentioned in this section cannot verify the nonsingularity of A. In such case, one may simply increase the working precision, e.g., quadruple precision. Another possibility is to apply Rump's method [32, 24] for inverting arbitrarily ill-conditioned matrices, and again we need a more accurate dot product; In the Rump's method, the computational precision of dot product is adaptively increased with iteratively updating an approximate inverse R until R satisfying $\|I - RA\|_\infty < 1$.

5. Verification methods for sparse linear systems

Fast verification for *large sparse* linear systems is still difficult in terms of both computational complexity and memory requirements except a few cases where it is known *in advance* or to be proved that the coefficient matrix A belongs to a certain special matrix class, e.g., diagonally dominant and M-matrix. The difficulty is mainly due to the destruction of the sparsity of A which is caused in the verification process. Let $\mathrm{nnz}(X)$ denote the number of nonzero elements in a matrix X. Usually, $\mathrm{nnz}(A) = \mathcal{O}(n)$, e.g., when using finite difference or finite element method. If we compute A^{-1} explicitly, then $\mathrm{nnz}(A^{-1}) = n^2$ in general, the explosion of nonzero elements due to fill-in (see Fig. 5.1). Therefore, we do not want to calculate A^{-1} nor its approximate full matrix R. Thus the verification for sparse systems of linear (interval) equations becomes one of the open problems posed in *Grand Challenges and Scientific Standards in Interval Analysis* [15] by Neumaier.

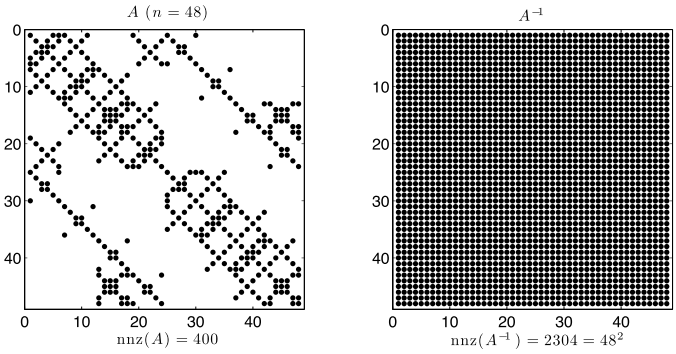

Fig. 5.1. Destruction of the sparsity.

For diagonally dominant matrices, we do not need to compute a full matrix R: Suppose A is strictly (row) diagonally dominant. Let $D := \mathrm{diag}(a_{11}, \ldots, a_{nn})$ and $\tilde{A} := A - D$. If setting $R := D^{-1} = \mathrm{diag}(a_{11}^{-1}, \ldots, a_{nn}^{-1})$, then we have

$$\|I - RA\|_\infty = \|I - D^{-1}A\|_\infty = \|D^{-1}\tilde{A}\|_\infty < 1,$$

since $\sum_{j \neq i} |a_{ij}| < |a_{ii}|$ for all i.

In this section, we restrict ourselves to the following cases:

- A is monotone (including an M-matrix).
- A is an H-matrix (or equivalently a generalized strictly diagonally dominant matrix).
- A is symmetric positive definite.

The main point is to develop a fast verification method which is suited for treating a sparse matrix A. For example, we may use an iterative solution method in the verification process. In any case, after bounding $\|A^{-1}\|_p$, we can apply Theorem 4.5.

5.1. Verification methods for inverse nonnegative matrices

The class of inverse nonnegative matrices ($A^{-1} \geq O$) has wide applications in engineering and scientific computations [3]. We can exploit the inverse nonnegativity for verified solutions of sparse linear systems.

We first present a theorem for calculating an upper bound of $\|A^{-1}\|_\infty$ in case of A being monotone.

THEOREM 5.1 (Ogita–Oishi–Ushiro [20]). *Let $A \in \mathbb{R}^{n\times n}$ be monotone. For given $\tilde{y} \in \mathbb{R}^n$, define a residual vector s by $s := \boldsymbol{e} - A\tilde{y}$. If $\|s\|_\infty < 1$, then*

$$\|A^{-1}\|_\infty \leq \frac{\|\tilde{y}\|_\infty}{1 - \|s\|_\infty}. \tag{5.1}$$

Usually, we set $\tilde{y}$ as an approximate solution of a linear system $Ay = \boldsymbol{e}$.

If A is an H-matrix, then we can extend Theorem 5.1 to the following corollary from (2.1) and Theorem 5.1:

COROLLARY 5.2. *Let $A \in \mathbb{R}^{n\times n}$ be an H-matrix. For given $\tilde{z} \in \mathbb{R}^n$, define a residual vector t by $t := \boldsymbol{e} - \mathcal{M}(A)\tilde{z}$. If $\|t\|_\infty < 1$, then*

$$\|A^{-1}\|_\infty \leq \frac{\|\tilde{z}\|_\infty}{1 - \|t\|_\infty}. \tag{5.2}$$

Clearly, these approaches can also be extended when the signs of the entries of A^{-1} are known, e.g., for totally positive matrices and others. Of course, if A is an H-matrix, then we can also apply the Hansen–Bliek–Rohn–Ning–Kearfott enclosure [14] introduced in Section 4.3.

So far we assume that A has monotone or H-matrix property. However, sometimes we do not have any information on the property of A. That means it is necessary to prove whether A has such property or not. For the purpose, several criteria (e.g., [9]) have been proposed. Using such methods, we can obtain a vector $v > \boldsymbol{o}$ which is *expected* to satisfy $\mathcal{M}(A)v > \boldsymbol{o}$. Using the verified numerical computation, it is easy to check whether it is true (see [14] and the INTLAB-code `verifylss.m`).

In the above cases, the only we need is a solver for a linear system $Ay = \boldsymbol{e}$ (or $\mathcal{M}(A) = \boldsymbol{e}$), which means we can apply the same solver for $Ax = b$. Therefore, we can keep the sparsity of A if an iterative solution method is used for solving $Ay = \boldsymbol{e}$

(or $\mathcal{M}(A) = \boldsymbol{e}$). Usually, we set a stopping criterion for the iterative solution method like $\|b - A\tilde{x}\|_2/\|b\|_2 < tol_x$ for some tolerance tol_x. On the other hand, $\|s\|_\infty$ in Theorem 5.1 (or $\|t\|_\infty$ in Corollary 5.2) should be less than one but could be considerably larger than tol_x; for example, $\|s\|_\infty < tol_y$ with $tol_y = 0.1$ while $tol_x = 10^{-9}$. Thus it can be expected that computing an error bound of $\tilde{x}$ is fairly faster than computing $\tilde{x}$ when choosing tol_x such that $tol_x \ll tol_y$.

We present the following numerical result with block tridiagonal M-matrix from Poisson's equation using Matlab. To solve the linear systems, we adopt the function `pcg`, which is ICCG (Incomplete Cholesky Conjugate Gradient) method, as an iterative solution method.

```
>> m=300; A=gallery('poisson',m); b=A*ones(m^2,1);
>> tol=1e-9; maxit=1000;
>> tic, M=cholinc(A,'0'); x=pcg(A,b,tol,maxit,M',M); toc
pcg converged at iteration 228 to a solution with
relative residual 9.7e-10
Elapsed time is 121.035008 seconds.
>> tic; xerr = verifymonotonelss(A,b,x,M), toc
pcg converged at iteration 116 to a solution with
relative residual 0.00033
xerr =
   7.6815e-06
Elapsed time is 10.111394 seconds.
```

Then it rigorously holds that $\mathtt{x} - \mathtt{xerr} \le \mathtt{A}^{-1}\mathtt{b} \le \mathtt{x} + \mathtt{xerr}$. This result is only for the reference, but the fastness of the verification method can be seen.

5.2. Verification methods for symmetric positive definite matrix

Some fast verification methods for symmetric positive definite linear systems have been proposed, e.g., [33, 36]. To prove the positive definiteness of a symmetric matrix A is related to this topic [35]. Even if we have an information that A is positive definite, we need to compute a lower bound $\underline{\lambda}$ of the smallest eigenvalue $\lambda_{\min}(A)$ of A. For a symmetric positive definite A, $\sigma_{\min}(A) = \lambda_{\min}(A)$, so that $\|A^{-1}\|_2 \le 1/\underline{\lambda}$.

Let $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ with $B = B^{\mathrm{T}}$ be given. The following algorithm executes a Cholesky factorization of B such that $B = R^{\mathrm{T}}R$ where $R = (r_{ij})$ is an upper triangular matrix.

ALGORITHM 5.3. *Cholesky factorization.*

---

for $j = 1 : n$
  for $i = 1 : j-1$
    $r_{ij} = \big(b_{ij} - \sum_{k=1}^{i-1} r_{ki}r_{kj}\big)/r_{ii}$
  end for
  $r_{jj} = \big(b_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\big)^{1/2}$
end for

If B is positive definite, then Algorithm 5.3 runs to completion,[4] and vice versa. Every known verification method for symmetric positive definite linear systems relies on the *floating-point* Cholesky factorization for some B. Although the verification process causes fill-in, such a verification method can be applied for moderately large sparse matrices when utilizing the band or sparse property of B; At least, computing an approximate (full) inverse of B is not necessary.

First, we present a fast verification method proposed in [33]: We set an *estimated* lower bound $\tilde{\lambda}$ of the smallest eigenvalue $\lambda_{\min}(A)$ of A, which is, for example, computed by some inverse iterations using the Cholesky factor of A. Then we execute a floating-point Cholesky factorization of $B := A - \tilde{\lambda}I$. If the factorization ends prematurely with an imaginary square root, then the positive definiteness of A cannot be proved. In such a case, one may change $\tilde{\lambda}$ and try the floating-point Cholesky factorization again. Suppose it runs to completion and denote the computed Cholesky factor by an upper triangular matrix $\tilde{R}$ such that $B \approx \tilde{R}^{\mathrm{T}}\tilde{R}$. Let $E := B - \tilde{R}^{\mathrm{T}}\tilde{R}$. Then using well-known Weyl's theorem, we have

$$|\lambda_{\min}(B) - \lambda_{\min}(\tilde{R}^{\mathrm{T}}\tilde{R})| \leq \|E\|_2.$$

Here we find

$$\begin{aligned} 0 &\leq \lambda_{\min}(\tilde{R}^{\mathrm{T}}\tilde{R}) \\ &\leq \lambda_{\min}(B) + \|E\|_2 = \lambda_{\min}(A - \tilde{\lambda}I) + \|E\|_2 = \lambda_{\min}(A) - \tilde{\lambda} + \|E\|_2. \end{aligned}$$

Hence

$$\sigma_{\min}(A) = \lambda_{\min}(A) \geq \tilde{\lambda} - \|E\|_2 =: \beta.$$

If $\beta > 0$, then A is proved to be positive definite. To compute $\|E\|_2$ or its upper bound, we can use a backward error analysis for Cholesky factorization [40, 33, 34] or explicitly compute (an inclusion of) $B - \tilde{R}^{\mathrm{T}}\tilde{R}$. Of course, the latter needs much more amount of memory but the estimation becomes more sharp.

If an approximate solution of $Ax = b$ is computed via (sparse) Cholesky factorization, then the above-mentioned verification process needs the same amount of memory as the approximation process. Moreover, the result of symbolic (Cholesky) factorization of A for finding all fill-in can be reused to save some computational effort since the sparse pattern of $B = A - \tilde{\lambda}I$ is the same as that of A.

Recently, a *super-fast* verification method for symmetric positive definite linear systems was proposed in [36]. Here "super-fast" means the computational effort for the verification is almost negligible compared to that for calculataing an approximate solution of $Ax = b$. The method relies on the following theorem due to Rump (cf. [35, 36]), which is an improved version of a theorem by Demmel [5].

[4]It means no imaginary square root appears in the factorization process.

THEOREM 5.4. *Suppose* Algorithm 5.3 *is applied to a symmetric matrix* $B = (b_{ij})$ *with* $b_{jj} \geq 0$, *and set* $\varphi_k := \gamma_k(1-\gamma_k)^{-1}$. *Then for execution in finite precision and barring overflow and underflow the following is true:*

i) *If* $\lambda_{\min}(B) \geq \sum_{j=1}^{n} \varphi_{j+1} b_{jj}$, *then* Algorithm 5.3 *runs to completion.*

ii) *If* $\lambda_{\min}(B) < -\sum_{j=1}^{n} \varphi_{j+1} b_{jj}$, *then* Algorithm 5.3 *ends prematurely with an imaginary square root.*

By the contraposition of ii) in Theorem 5.4, if Algorithm 5.3 runs to completion, then

$$\lambda_{\min}(B) \geq -\sum_{j=1}^{n} \varphi_{j+1} b_{jj}. \tag{5.3}$$

Suppose $\beta_1 \geq \sum_{j=1}^{n} \varphi_{j+1} a_{jj}$, and set $B := A - \beta_2 I$ for some[5] $\beta_2 > \beta_1$. If the floating-point Cholesky factorization of B ends prematurely, then the positive definiteness of A cannot be proved. Suppose it runs to completion. Even then it does not necessarily imply B to be positive definite because the factorization process suffers rounding errors. Nevertheless, it holds from (5.3) that

$$\begin{aligned} \lambda_{\min}(A) - \beta_2 &= \lambda_{\min}(A - \beta_2 I) = \lambda_{\min}(B) \\ &\geq -\sum_{j=1}^{n} \varphi_{j+1} b_{jj} = -\sum_{j=1}^{n} \varphi_{j+1}(a_{jj} - \beta_2) \\ &\geq -\sum_{j=1}^{n} \varphi_{j+1} a_{jj} \geq -\beta_1 \end{aligned}$$

and

$$\lambda_{\min}(A) \geq \beta_2 - \beta_1 > 0,$$

which implies A to be positive definite. Moreover, $\tilde{x}$ can be computed by forward and backward substitution using $\tilde{R}$, the computed Cholesky factor of B. If A is ill-conditioned, then we need some iterative refinement [36]: For $k = 0, 1, 2, \ldots,$

$$\tilde{x}^{(k+1)} := \tilde{x}^{(k)} + B^{-1}(b - A\tilde{x}^{(k)}) \tag{5.4}$$

with $\tilde{x}^{(0)} := \tilde{x}$. In practical application, multiplication of B^{-1} is replaced by forward and backward substitution using $\tilde{R}$. It is shown in [36] that if $c := \beta_2 \|A^{-1}\|_2 < 1$, then

$$\|A^{-1}b - \tilde{x}^{(k+1)}\|_2 \lesssim \frac{c}{1-c} \|A^{-1}b - \tilde{x}^{(k)}\|_2 \lesssim \left(\frac{c}{1-c}\right)^{k+1} \|A^{-1}b - \tilde{x}\|_2.$$

So the residual iteration (5.4) with perturbed iteration matrix $B = A - \beta_2 I$ instead of A behaves similar to the usual residual iteration provided $\beta_2 \|A^{-1}\|_\infty < 1$, which means $\beta_2 < \sigma_{\min}(A)$.

[5]For example, $\beta_2 = 2\beta_1$ in [36].

6. Conclusions

We surveyed the fast verification methods for dense and sparse linear systems. We do not have enough space to mention the details of rounding error analysis in floating-point arithmetic, interval arithmetic, concrete algorithms for obtaining verified solutions of linear systems, the efficient use of the iterative refinement with result verification, nor the extension to the case where A and b are real/interval quantities. We have to refer the interested reader to [1, 10, 12, 16, 25, 34] and the literature cited there. Moreover, we could not talk about applications. For example, an efficient verification method for saddle point problems has been proposed in [4], in which the fast verification methods presented in this paper are used.

Nevertheless, we hope the reader has now believed that the verified numerical computation can actually answer the first question "How reliable is the numerical solution?" for a certain class of mathematical problems.

References

[1] G. Alefeld and J. Herzberger, Introduction to Interval Computations. Academic Press, New York, 1983.

[2] ANSI/IEEE, IEEE Standard for Binary Floating Point Arithmetic, Std 754-1985 edition. IEEE, New York, 1985.

[3] A. Berman and R.J. Plemmons, Nonnegative Matrices in the Mathematical Sciences. Classics Appl. Math., **9**, SIAM, Philadelphia, 1994.

[4] X. Chen and K. Hashimoto, Numerical validation of solutions of saddle point matrix equations. Numerical Linear Algebra with Applications, **10** (2003), 661–672.

[5] J.B. Demmel, On floating point errors in Cholesky. LAPACK Working Note 14 CS-89-87, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1989.

[6] J. Dongarra, Basic linear algebra subprograms technical forum standard. International Journal of High Performance Applications and Supercomputing, **16** (2002), 1–111.

[7] K. Goto and R.A. van de Geijn, Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw., **34** (2008), 12:1–12:25.

[8] K. Goto and R.A. van de Geijn, High performance implementation of the Level-3 BLAS. ACM Trans. Math. Softw., **35** (2008), 4:1–4:14.

[9] A. Hadjidimos, An extended compact profile iterative method criterion for sparse H-matrix. Linear Alg. Appl., **389** (2004), 329–345.

[10] N.J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd edition. SIAM, Philadelphia, PA, 2002.

[11] R. Krawczyk, Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. Computing, **4** (1969), 187–201.

[12] G. Mayer, On regular and singular interval systems. J. Comp. Appl. Math., **199** (2007), 220–228.

[13] R.E. Moore, A test for existence of solutions for non-linear systems. SIAM J. Numer. Anal., **4** (1977), 611–615.

[14] A. Neumaier, A simple derivation of the Hansen–Bliek–Rohn–Ning–Kearfott enclosure for linear interval equations. Reliable Computing, **5** (1999), 131–136; Erratum. Reliable Computing, **6** (2000), 227.

[15] A. Neumaier, Grand challenges and scientific standards in interval analysis. Reliable Computing, **8** (2002), 313–320.

[16] A. Neumaier, Interval Methods for Systems of Equations. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1990.

[17] S. Ning and R.B. Kearfott, A comparison of some methods for solving linear interval equations. SIAM J. Numer. Anal., **34** (1997), 1289–1305.

[18] T. Ogita and S. Oishi, Fast verification method for large-scale linear systems. Trans. IPSJ, **46** (2005), 10–18 (in Japanese).
[19] T. Ogita and S. Oishi, Tight enclosures of solutions of linear systems. International Series of Numerical Mathematics, **157** (2009), 167–178 (Inequalities and Applications, C. Bandle, A. Gilányi, L. Losonczi, Z. Páles and M. Plum eds., Birkhäuser Verlag).
[20] T. Ogita, S. Oishi and Y. Ushiro, Fast verification of solutions for sparse monotone matrix equations. Computing Suppl., **15** (2001), 175–187.
[21] T. Ogita, S. Oishi and Y. Ushiro, Computation of sharp rigorous componentwise error bounds for the approximate solutions of systems of linear equations. Reliable Computing, **9** (2003), 229–239.
[22] T. Ogita, S.M. Rump and S. Oishi, Accurate sum and dot product. SIAM J. Sci. Comput., **26** (2005), 1955–1988.
[23] S. Oishi, Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation. Linear Alg. Appl., **324** (2001), 133–146.
[24] S. Oishi, K. Tanabe, T. Ogita and S.M. Rump, Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices. J. Comp. Appl. Math., **205** (2007), 533–544.
[25] S. Oishi and S.M. Rump, Fast verification of solutions of matrix equations. Numer. Math., **90** (2002), 755–773.
[26] J. Rohn, A Handbook of Results on Interval Linear Problems. Internet text available at `http://www.cs.cas.cz/rohn/handbook/`
[27] S.M. Rump, A note on epsilon-inflation. Reliable Computing, **4** (1998), 371–375.
[28] S.M. Rump, Approximate inverses of almost singular matrices still contain useful information. Forschungsschwerpunktes Informations- und Kommunikationstechnik, Technical Report 90.1, Hamburg University of Technology, 1990.
[29] S.M. Rump, Computer-Assisted Proofs and Self-Validating Methods. Accuracy and Reliability in Scientific Computing (Chapter 10), SIAM, Philadelphia, PA, 2005.
[30] S.M. Rump, Fast and parallel interval arithmetic. BIT, **39** (1999), 534–554.
[31] S.M. Rump, INTLAB—INTerval LABoratory. Developments in Reliable Computing, T. Csendes ed., Kluwer Academic Publishers, Dordrecht, 1999, 77–104, `http://www.ti3.tu-harburg.de/rump/intlab/`.
[32] S.M. Rump, Kleine Fehlerschranken bei Matrixproblemen. Universität Karlsruhe, Ph.D. thesis, 1980.
[33] S.M. Rump, Validated solution of large linear systems. Computing Suppl., **9** (1993), 191–212.
[34] S.M. Rump, Verification methods for dense and sparse systems of equations. Topics in Validated Computations—Studies in Computational Mathematics, J. Herzberger ed., Elsevier, Amsterdam, 1994, 63–136.
[35] S.M. Rump, Verification of positive definiteness. BIT Numerical Mathematics, **46** (2006), 433–452.
[36] S.M. Rump and T. Ogita, Super-fast validated solution of linear systems. J. Comp. Appl. Math., **199** (2007), 199–206.
[37] S.M. Rump, T. Ogita and S. Oishi, Accurate floating-point summation, Part I: Faithful rounding. SIAM J. Sci. Comput., **31** (2008), 189–224.
[38] S.M. Rump, T. Ogita and S. Oishi, Accurate floating-point summation, Part II: Sign, K-fold faithful and rounding to nearest. SIAM J. Sci. Comput., **31** (2008), 1269–1302.
[39] V. Strassen, Gaussian elimination is not optimal. Numer. Math., **13** (1969), 354–356.
[40] J.-G. Sun, Rounding-error and perturbation bounds for the Cholesky and LDL^{T} factorizations. Linear Alg. Appl., **173** (1992), 77–97.
[41] T. Yamamoto, Error bounds for approximate solutions of systems of equations. Japan J. Appl. Math., **1** (1984), 157–171.