Generation of Linear Systems with Specified Solutions for Numerical Experiments*

Katsuhisa Ozaki
Department of Mathematical Sciences, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan
ozaki@sic.shibaura-it.ac.jp

Takeshi Ogita
Division of Mathematical Sciences, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
ogita@lab.twcu.ac.jp

Abstract

The goal of this paper is to generate problems to test solvers for linear systems. Assume that a coefficient matrix A and a right-hand side vector b are given. If numerical computations are used to solve a linear system $Ax = b$, computed results are usually different from the exact solution due to accumulation of rounding errors. We propose a method to produce a coefficient matrix A and a right-hand side vector b such that the exact solution x is known. The method is useful for examining the accuracy of computed results obtained by some numerical algorithms, and it is useful for checking overestimation of the error bounds obtained by verified numerical computations.

Keywords: Linear systems, verified numerical computations, test problems
AMS subject classifications: 15A06, 65G30, 65G50

1 Introduction

This paper is concerned with test problems for numerical linear algebra. Let $\boldsymbol{F}$ be a set of some fixed precision binary floating-point numbers as defined by IEEE 754 [1]. For $A \in \boldsymbol{F}^{n\times n}$ and $b \in \boldsymbol{F}^n$, we consider the linear system $Ax = b$. Let $\widehat{x}$ be an approximate solution of the linear system. If the approximate solution $\widehat{x}$ is obtained by numerical computations, the result $\widehat{x}$ may be inaccurate due to accumulation of rounding errors. Since it is difficult to know the error $\|A^{-1}b - \widehat{x}\|$ exactly, the residual $\|b - A\widehat{x}\|$ often is used instead. However, it is sometimes seen that there is a gap between the residual $\|b - A\widehat{x}\|$ and the error $\|x - \widehat{x}\|$, i.e., $\|x - \widehat{x}\|$ may be large even

*Submitted: December 14, 2016; Revised: June 8, 2017; Accepted: August 20, 2017.

if $\|b - A\widehat{x}\|$ is small. If the exact solution of the linear system is known, then this is useful in the following cases:

- The accuracy of the numerical solution can be checked. For example, it will help to analyze the behavior of convergence for iterative methods, e.g., Jacobi and Gauss-Seidel methods as stationary iterative methods, and CG, BiCG, and GMRES as Krylov subspace methods.
- Verified numerical computations give an upper bound of $\|x - \widehat{x}\|$ by using only numerical computations. It is difficult to check whether the obtained upper bound is overestimated if the exact solution x is unknown. Linear systems with known solutions are useful for checking of overestimation of the verified numerical computations.

One may think that for given $A \in \boldsymbol{F}^{n\times n}$ and $x \in \boldsymbol{F}^n$, if we compute the right-hand side vector $b := Ax$, x is the exact solution of $Ax = b$. However, if rounding error occurs in the floating-point evaluation of Ax, then the vector x may not satisfy $Ax = b$. Miyajima, Ogita and Oishi developed a method [2] which produces $A' \in \boldsymbol{F}^{m\times m}$, $x' \in \boldsymbol{F}^m$ and $b' \in \boldsymbol{F}^m$ from given $A \in \boldsymbol{F}^{n\times n}$ and $x \in \boldsymbol{F}^n$ $(m \geq n)$ such that $A'x' = b'$. The matrix A' and the vector b' are produced using an error-free transformation [6], where x' is an extension of x; $x'_i = x_i$ for $1 \leq i \leq n$ and $x'_i = 1$ for $n+1 \leq i \leq m$. As an advantage, the condition number of the coefficient matrix and the solution $x \in \boldsymbol{F}^n$ are free to be set. However, the size m of A' is generally greater than n. In addition, the structure, e.g., symmetric, persymmetic, Toeplitz and Hankel, of A' and A is usually different, and the number of non-zero elements in A' is greater than in the original A.

We develop additional methods for generating test problems. For given $A \in \boldsymbol{F}^{n\times n}$ and $x \in \boldsymbol{F}^n$, we produce $A' \in \boldsymbol{F}^{n\times n}$ and $b \in \boldsymbol{F}^n$ to satisfy $A'x = b$. The advantage of our methods is that our methods preserve size of matrices and some structures of matrices such as symmetric, persymmetic, Toeplitz, Hankel and Hessenberg, which is useful for dealing with sparse matrices. Moreover, the number of non-zero elements in A' is the same as or less than that in the original matrix A. The disadvantage of our methods is that users cannot freely set the solution $x \in \boldsymbol{F}^n$ of the linear system. The condition number of the coefficient matrix is free to be set under some limitations.

2 Notations and Previous Work for Test Problems

In this section, we first introduce notation we will use in this paper. Let $\mathrm{fl}(\cdot)$ denote that all operations enclosed in the parenthesis are evaluated by floating-point arithmetic with a particular order. The rounding mode of $\mathrm{fl}(\cdot)$ is rounding to the nearest (roundTiesToEven in IEEE 754 [1]). We omit to write $\mathrm{fl}(\cdot)$ for each arithmetic operation in the parenthesis for simplicity, e.g., $\mathrm{fl}((a+b)+c) = \mathrm{fl}(\mathrm{fl}(a+b)+c)$ and $\mathrm{fl}((a+b)+(c+d)) = \mathrm{fl}(\mathrm{fl}(a+b)+\mathrm{fl}(c+d))$ for $a, b, c, d \in \boldsymbol{F}$. Let $\mathrm{float}(\cdot)$ denote that all operations enclosed in the parenthesis are evaluated by floating-point arithmetic with any order of computations. The notations $\mathrm{fl}(\cdot)$ and $\mathrm{float}(\cdot)$ are used in [3]. Note that there is no difference in working precision between them. For dot product and matrix multiplication, we assume that divide and conquer methods, e.g., Strassen's method [9] and Winograd's method [10], are not used for $\mathrm{fl}(\cdot)$ and $\mathrm{float}(\cdot)$. For the dot product x^Ty for x and $y \in \boldsymbol{F}^n$, the notation $\mathrm{float}\left(x^Ty\right)$ implies that the order of the sum of n terms is arbitrary after computing $\mathrm{fl}(x_iy_i)$. $\mathrm{fl}(x^Ty)$ indicates one of the orders in float $\left(x^T y\right)$, depending on a user's computational environment (for example, compiler, libraries or the number of cores in the CPU).

This manner is straightforwardly extended for matrix-vector products. Here, we explain the difference between $\mathrm{fl}(\cdot)$ and $\mathrm{float}(\cdot)$. For example, $\mathrm{float}(x^T y) < \alpha$ for x and $y \in \boldsymbol{F}^3$ indicates that all of the following are satisfied:

$$\begin{aligned}
\mathrm{fl}((x_1 y_1 + x_2 y_2) + x_3 y_3) &< \alpha, \\
\mathrm{fl}(x_1 y_1 + (x_2 y_2 + x_3 y_3)) &< \alpha, \\
\mathrm{fl}((x_1 y_1 + x_3 y_3) + x_2 y_2) &< \alpha.
\end{aligned}$$

Let $x_1 = 1, x_2 = x_3 = \mathbf{u}$ and $y_1 = y_2 = y_3 = 1$. $\mathrm{float}(x^T y) \nleq 1$ since

$$\mathrm{fl}(x_1 y_1 + (x_2 y_2 + x_3 y_3)) = 1 + 2\mathbf{u}.$$

Let $\mathbf{u}$ be roundoff unit, e.g., $\mathbf{u} = 2^{-24}$ for binary32 and $\mathbf{u} = 2^{-53}$ for binary64 in IEEE 754. The constant f_{max} denotes the maximum floating-point number. Define the function $\mathtt{ufp}(a)$ for $a \in \boldsymbol{R}$ as

$$\mathtt{ufp}(a) := \begin{cases} 0, & \text{if } a = 0, \\ 2^{\lfloor \log_2 |a| \rfloor}, & \text{otherwise.} \end{cases}$$

A product of two floating-point numbers is transformed into a sum of two floating-point numbers such that

$$r_1 r_2 = r_3 + r_4, \quad r_3 = \mathrm{fl}(r_1 r_2), \quad r_1, r_2, r_3, r_4 \in \boldsymbol{F}. \tag{1}$$

Here, assume that $|r_1 r_2| \geq \mathbf{u}_N \mathbf{u}^{-1}$, where $\mathbf{u}_N$ is the minimum positive normalized number. r_3 and r_4 are obtained by an error-free transformation in [5] or application of a fused multiply-add (FMA) operation.

The following lemmas are useful for the proofs in this paper.

Lemma 2.1 *Assume that $\mathbf{u}_N \leq \mathtt{ufp}(x) < f_{max}$ for $x \in \mathbb{R}$. If x is a multiple of $k\mathbf{u}$ for $k = 2^i$, $i \in \mathbb{Z}$ with $|x| \leq k$, then x can be represented by a floating-point number, i.e., $x \in \boldsymbol{F}$.*

This lemma is obtained by definition of floating-point numbers in IEEE 754.

Lemma 2.2 (Rump et al. [4]) *For a and $b \in \boldsymbol{F}$, there exists $\delta \in \boldsymbol{R}$ such that*

$$a + b = \mathrm{fl}(a + b) + \delta, \quad |\delta| \leq \mathbf{u} \cdot \mathtt{ufp}\left(\mathrm{fl}(a + b)\right).$$

Next, we introduce error-free splitting proposed by Rump-Ogita-Oishi [4].

Theorem 2.1 *Assume $\sigma = 2^k \cdot 2^{\lceil \log_2 |a| \rceil}, k \in \mathbb{Z}$, $a \in \boldsymbol{F}$ and $\sigma > |a| \in \boldsymbol{F}$. Floating-point numbers b and c are obtained by*

$$b = \mathrm{fl}\left((\sigma + a) - \sigma\right), \quad c = \mathrm{fl}(a - b).$$

Then, the following properties are satisfied:

$$|b| \leq 2^{-k}\sigma, \quad b \in \mathbf{u}\sigma\mathbb{Z}, \quad |c| \leq \mathbf{u}\sigma, \quad \mathrm{fl}\left((\sigma + a) - \sigma\right) = \mathrm{fl}\left(\sigma + a\right) - \sigma.$$

We briefly explain a method by Miyajima et al. [2]. First, a coefficient matrix $A \in \boldsymbol{F}^{n \times n}$ and a vector $x \in \boldsymbol{F}^n$ are given. A matrix-vector product Ax is transformed into an unevaluated sum of floating-point vectors such that

$$Ax = \sum_{i=1}^{k} b^{(i)}, \quad b^{(i)} \in \boldsymbol{F}^n, \quad k \in \mathbb{N}, \tag{2}$$

where accurate summation algorithms in [6] can be used to obtain (2). Let $I \in \boldsymbol{F}^{(k-1)\times(k-1)}$ and $O \in \boldsymbol{F}^{(k-1)\times(k-1)}$ be the identity matrix and the zero matrix, respectively. Define

$$B := [-b^{(2)}, -b^{(3)}, \dots, -b^{(k)}] \in \boldsymbol{F}^{n\times(k-1)}, \quad e := (1,1,\dots,1)^T \in \boldsymbol{F}^{k-1}.$$

Then, $A' \in \boldsymbol{F}^{(n+k-1)\times(n+k-1)}$, $x' \in \boldsymbol{F}^{n+k-1}$ and $b' \in \boldsymbol{F}^{n+k-1}$ are given by

$$A' := \begin{pmatrix} A & B \\ O & I \end{pmatrix}, \quad x' := \begin{pmatrix} x \\ e \end{pmatrix}, \quad b' := \begin{pmatrix} b^{(1)} \\ e \end{pmatrix},$$

where $A'x' = b'$ is satisfied from (2). It is advantageous that a user can set arbitrary $x \in \boldsymbol{F}^n$. In addition, if A is ill-conditioned, then this method produces an ill-conditional matrix A' satisfying $A'x = b'$. However, the size of A' is larger than that of A in many cases. Moreover, if A has a special structure, then a structure of A' is different from that of the original matrix A. For example, even if A is symmetric, A' becomes unsymmetric in many cases.

3 Check of Rounding Errors

If $Ax = \mathrm{fl}(Ax)$ is satisfied for a given coefficient matrix A and a vector x, then b is given by $\mathrm{fl}(Ax)$, and the vector x is the exact solution of $Ax = b$. We introduce methods which guarantee $Ax = \mathrm{fl}(Ax)$.

3.1 Check with Directed Rounding

Let $\mathrm{fl}_{\bigtriangledown}(\cdot)$ and $\mathrm{fl}_{\triangle}(\cdot)$ indicate that each operation in the parenthesis is evaluated by floating-point arithmetic with rounding-downwards and rounding-upwards, respectively. These rounding modes are defined in IEEE 754. The following theorem is useful for checking $Ax = \mathrm{fl}(Ax)$.

Theorem 3.1 *For x and $y \in \boldsymbol{F}^n$, $\mathrm{fl}_{\bigtriangledown}\left(x^T y\right) = \mathrm{fl}_{\triangle}\left(x^T y\right) \Rightarrow \mathrm{fl}\left(x^T y\right) = x^T y$. Here, assume that the orders of the computations are same for $\mathrm{fl}(\cdot)$, $\mathrm{fl}_{\bigtriangledown}(\cdot)$, and $\mathrm{fl}_{\triangle}(\cdot)$.*

This theorem is valid even when overflow or underflow occurs in the floating-point evaluation.

Proof.

For a and $b \in \boldsymbol{F}$, we obtain

$$\mathrm{fl}_{\bigtriangledown}(a \circ b) \le a \circ b \le \mathrm{fl}_{\triangle}(a \circ b), \quad \mathrm{fl}_{\bigtriangledown}(a \circ b) \le \mathrm{fl}(a \circ b) \le \mathrm{fl}_{\triangle}(a \circ b), \quad \circ \in \{+, *\}.$$

By using it recursively, the proof is finished. □

Theorem 3.1 can be straightforwardly extended to a product of the matrix A and the vector x, i.e., $\mathrm{fl}_{\bigtriangledown}(Ax) = \mathrm{fl}_{\triangle}(Ax) \Rightarrow \mathrm{fl}(Ax) = Ax$. Remark that $\mathrm{fl}_{\bigtriangledown}(Ax) \neq \mathrm{fl}_{\triangle}(Ax)$ does not imply $Ax \notin \boldsymbol{F}^n$.

We write an algorithm with codes for MATLAB, which detects a rounding error based on Theorem 3.1.

Algorithm 1 *The following function checks whether a rounding error occurs in* $\mathrm{fl}(Ax)$ *and produces a vector* $r \in \boldsymbol{F}^n$. *The output* $r_i = 0$ *indicates that a rounding error occurs in the evaluation of* $\sum_{j=1}^{n} a_{ij}x_j$. *The output* $r_i = 1$ *indicates that a rounding error never occurs in the evaluation of* $\sum_{j=1}^{n} a_{ij}x_j$.

```
function r = Check(A, x)
  r = zeros(size(A,1),1);
  a = feature('setround', Inf)   %a keeps the rounding mode before switching
  y1 = A * x;
  feature('setround', -Inf)
  y2 = A * x;
  r(y1 == y2) = 1;
  feature('setround', a);   %restore the previous rounding mode
end
```

Note that $\mathrm{fl}_{\triangledown}\left(x^Ty\right) = \mathrm{fl}_{\vartriangle}\left(x^Ty\right) \Leftarrow \mathrm{fl}\left(x^Ty\right) = x^Ty$ is not necessarily satisfied, as the following example illustrates.

$$\mathrm{fl}\left((1+1.5\mathbf{u})+0.5\mathbf{u}\right) = \mathrm{fl}\left((1+2\mathbf{u})+0.5\mathbf{u}\right) = 1+2\mathbf{u},$$
$$\mathrm{fl}_{\triangledown}\left((1+1.5\mathbf{u})+0.5\mathbf{u}\right) = 1, \quad \mathrm{fl}_{\vartriangle}\left((1+1.5\mathbf{u})+0.5\mathbf{u}\right) = 1+4\mathbf{u}.$$

In addition, $\mathrm{fl}_{\triangledown}(Ax) = \mathrm{fl}_{\vartriangle}(Ax) \Rightarrow \mathrm{fl}(Ax) = Ax$ is valid, but $\mathrm{fl}_{\triangledown}(Ax) = \mathrm{fl}_{\vartriangle}(Ax) \Rightarrow \mathrm{float}(Ax) = Ax$ is not satisfied. Let the first row of $A \in \boldsymbol{F}^{3\times 3}$ and $x \in \boldsymbol{F}^3$ be

$$a_{11} = a_{12} = \mathbf{u},\ a_{13} = 1, \quad x_1 = x_2 = x_3 = 1.$$

Then,

$$\begin{aligned}
\mathrm{fl}_{\triangledown}\left((a_{11}x_1 + a_{12}x_2) + a_{13}x_3\right) &= \mathrm{fl}_{\triangledown}\left((\mathbf{u}+\mathbf{u})+1\right) = 1+2\mathbf{u},\\
\mathrm{fl}_{\vartriangle}\left((a_{11}x_1 + a_{12}x_2) + a_{13}x_3\right) &= \mathrm{fl}_{\vartriangle}\left((\mathbf{u}+\mathbf{u})+1\right) = 1+2\mathbf{u}.
\end{aligned}$$

Therefore, no rounding error occurs in this case. However, if we change the order of evaluation, we get different results:

$$\mathrm{fl}\left((a_{13}x_3 + a_{11}x_1) + a_{12}x_2\right) = \mathrm{fl}\left((1+\mathbf{u})+\mathbf{u}\right) = 1.$$

Next, assume that a routine for the dot product uses fused multiply-add (FMA). We use notation $\mathtt{FMA}(a,b,c)$ for $a,b,c \in \boldsymbol{F}$; the nearest floating-point number to $ab+c$ is obtained by $\mathtt{FMA}(a,b,c)$. Let x and $y \in \boldsymbol{F}^2$ be

$$x_1 = -\mathbf{u} + 2\mathbf{u}^2, \quad x_2 = 1+2\mathbf{u}, \quad y_1 = 1, \quad y_2 = 1-\mathbf{u}.$$

Then,

$$\mathtt{FMA}_{\triangledown}(x_2, y_2, \mathrm{fl}_{\triangledown}(x_1y_1)) = 1, \quad \mathtt{FMA}_{\vartriangle}(x_2, y_2, \mathrm{fl}_{\vartriangle}(x_1y_1)) = 1,$$

and

$$\mathrm{fl}_{\triangledown}(x_1y_1 + x_2y_2) = 1, \quad \mathrm{fl}_{\vartriangle}(x_1y_1 + x_2y_2) = 1+4\mathbf{u}.$$

The result of Algorithm 1 depends on the computational order of $\mathrm{fl}(\cdot)$ and users' computational environments.

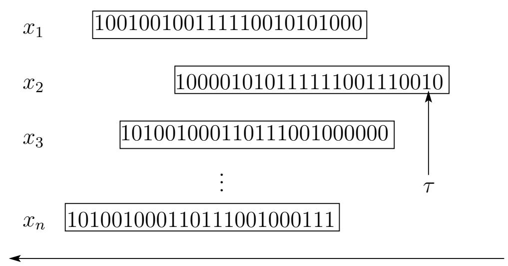

Figure 1: Image of τ in (5). τ_i is the unit in the last non-zero bit in the significand of x_i.

3.2 Check without Directed Rounding

Since there are some computational environments where we cannot switch the rounding modes, we will give a theorem for verifying $Ax = \text{fl}(Ax)$ using only rounding to the nearest mode (roundTiesToEven), which is the default rounding mode in many computational environments. The idea is based on a technique using overflow in [8]. Assume that $A \in \boldsymbol{F}^{n\times n}$ and $x \in \boldsymbol{F}^n$ are given. Let a constant $c \in \mathbb{R}$ be

$$c = 2^r,\ r \in \mathbb{N}, \quad \frac{c}{2} \in \boldsymbol{F}, \quad c \notin \boldsymbol{F}.$$

For binary64, $c = 2\texttt{ufp}(f_{max}) = 2^{1024}$. We define two constants d_1 and d_2 such that

$$d_1 d_2 = c\mathbf{u},\ \text{for example, } d_1 = 2^{486} \text{ and } d_2 = 2^{485} \text{ for binary64.} \tag{3}$$

We find a vector v and a constant τ such that

$$v_i := \min_{1\le j\le n} v_{ij}, \quad a_{ij} \in v_{ij}\mathbb{Z}, \quad a_{ij} \notin 2v_{ij}\mathbb{Z} \tag{4}$$

$$\tau := \min_{1\le i\le n} \tau_i, \quad x_i \in \tau_i\mathbb{Z}, \quad x_i \notin 2\tau_i\mathbb{Z}, \tag{5}$$

where $v_{ij}, \tau_i \in \{2^k \mid k \in \mathbb{Z}\}$. Figure 1 shows an image of τ. Then,

$$\dot{a}_{ij} := d_1/v_i \cdot a_{ij}, \quad \dot{x} := d_2/\tau \cdot x, \quad t := \text{fl}\left(\dot{A}\dot{x}\right). \tag{6}$$

Theorem 3.2 *Assume that a vector t is produced by (6). If*

$$t_i \notin \{\texttt{Inf},\ -\texttt{Inf},\ \texttt{NaN}\} \tag{7}$$

is satisfied for all i, then $\text{fl}\left(\dot{A}\dot{x}\right) = \dot{A}\dot{x}$.

Proof.
The proof consists of two parts; there is no rounding error in the all products $\dot{a}_{ik}\dot{x}_{kj}$ and no rounding error occurs in the summation. Using (4), (5), and the scalings (6), we have

$$\dot{a}_{ij} \in d_1\mathbb{Z}, \quad \dot{x}_j \in d_2\mathbb{Z}.$$

These and (3) yield

$$\dot{a}_{ik}\dot{x}_k,\ \text{fl}\left(\dot{a}_{ik}\dot{x}_k\right) \in d_1 d_2\mathbb{Z} = c\mathbf{u}\mathbb{Z}. \tag{8}$$

The assumption (7) says that overflow never occurred in $\mathrm{fl}\left(\dot{A}\dot{x}\right)$. It derives

$$|\mathrm{fl}\,(\dot{a}_{ik}\dot{x}_k)\,| < c. \tag{9}$$

The assumption of Lemma 2.1 is satisfied from (8) and (9). Hence, we obtain $\mathrm{fl}\,(\dot{a}_{ik}\dot{x}_k) = \dot{a}_{ik}\dot{x}_k$.

Let an intermediate result $\theta \in \boldsymbol{F}$ in the summation $\sum_{k=1}^{n} \mathrm{fl}\,(\dot{a}_{ik}\dot{x}_k)$ after computing all products $\mathrm{fl}\,(\dot{a}_{ik}\dot{x}_k)$ be obtained by a computation of

$$\theta = \mathrm{fl}(\alpha_1 + \alpha_2), \quad \alpha_1,\ \alpha_2 \in \boldsymbol{F}.$$

From (8) and the assumption "no overflow", we have

$$\theta,\ \alpha_1,\ \alpha_2 \in \mathbf{u}c\mathbb{Z}, \quad |\theta|,\ |\alpha_1|,\ |\alpha_2| < c. \tag{10}$$

If a rounding error first occurs at the evaluation of θ, then from Theorem 2.2, the following δ exists such that

$$\theta = \alpha_1 + \alpha_2 + \delta, \quad 0 \neq |\delta| \leq \mathbf{u} \cdot \mathtt{ufp}(\theta) \leq \frac{1}{2}\mathbf{u}c. \tag{11}$$

From (10), both θ and $\alpha_1 + \alpha_2$ are in $\mathbf{u}c\mathbb{Z}$. However, $\delta \notin \mathbf{u}c\mathbb{Z}$ from (11). Therefore, no rounding error occurs in $\mathrm{fl}\left(\dot{A}\dot{x}\right)$ because the equality (11) is contradiction. $\square$

Both $\dot{A}$ and $\dot{x}$ are obtained by the scaling from A and b using constants of powers of two, respectively. Therefore, if neither overflow nor underflow occurs in $\mathrm{fl}(Ax)$ and $\dot{A}\dot{x} = \mathrm{fl}\left(\dot{A}\dot{x}\right)$ is guaranteed by Theorem 3.2, then $Ax = \mathrm{fl}(Ax)$ is also satisfied. Note that even if we find $\pm$`Inf` or `NaN` in the vector t, $Ax = \mathrm{fl}(Ax)$ may be satisfied. Theorem 3.2 is a little weaker than Theorem 3.1. For example, let the first row of $A \in \boldsymbol{F}^{3\times 3}$ and $x \in \boldsymbol{F}^3$ be

$$a_{11} = a_{12} = \mathbf{u},\ a_{13} = 1, \quad x_1 = x_2 = x_3 = 1.$$

Then, $\left(\dot{A}\dot{x}\right)_1$ becomes `Inf`, however,

$$\begin{aligned} \mathrm{fl}_{\bigtriangledown}\left((a_{11}x_1 + a_{12}x_2) + a_{13}x_3\right) &= \mathrm{fl}_{\bigtriangleup}\left((a_{11}x_1 + a_{12}x_2) + a_{13}x_3\right) \\ &= 1 + 2\mathbf{u} = a_{11}x_1 + a_{12}x_2 + a_{13}x_3. \end{aligned}$$

Next, we develop a reproducible method for the check of a rounding error.

Theorem 3.3 *Define a vector* $\widehat{t} := \mathrm{fl}\left(|\dot{A}||\dot{x}|\right)$*, where* $\dot{A}$ *and* $\dot{x}$ *are obtained in (6). If*

$$\widehat{t}_i \notin \{\mathtt{Inf},\ -\mathtt{Inf},\ \mathtt{NaN}\}$$

is satisfied for all i*, then* $\mathrm{float}\left(\dot{A}\dot{x}\right) = \dot{A}\dot{x}$.

Proof.
$\mathrm{fl}\left(|\dot{A}||\dot{x}|\right) = |\dot{A}||\dot{x}|$ is proved similar to Theorem 3.2. From (8), we have

$$\mathrm{fl}\left(\dot{a_{ij}}\dot{x_j}\right), \dot{a_{ij}}\dot{x_j} \in \mathbf{u}c\mathbb{Z}.$$

Since

$$\mathbf{u}c\mathbb{Z} \ni \text{float}\left(\sum_{j\in K} |\dot{a_{ij}}||\dot{x_j}|\right) \le \text{float}\left(\sum_{j\in L} |\dot{a_{ij}}||\dot{x_j}|\right) < c, \quad \emptyset \ne K \subseteq L \subseteq \{1, 2, \ldots, n\},$$

float $\left(\dot{A}\dot{x}\right) = \dot{A}\dot{x}$ is satisfied. □

Note that we use fl$(\cdot)$ for Theorem 3.3, but float $\left(\dot{A}\dot{x}\right) = \dot{A}\dot{x}$ is guaranteed. Moreover, even if a routine for the matrix-vector product uses FMA or divide and conquer methods such as the Winograd method, Theorem 3.3 is still valid. It means that if $|\dot{A}||\dot{x}| = \text{fl}\left(|\dot{A}||\dot{x}|\right)$ is verified using a matrix-vector product routine, then it is also verified in other environments, even if a matrix-vector product routine is different. Therefore, the method in Theorem 3.3 is reproducible.

If $Ax = \text{fl}(Ax)$ is guaranteed by Theorem 3.1, 3.2, or 3.3, we obtain $b := \text{fl}(Ax)$ satisfying $Ax = b$. This check works as a filter, and it should be applied first. If this filter does not pass, then we move to new methods given in the next section.

4 Basic Perturbation Method

For a brief introduction, we propose several methods to generate A' and b such that $A'x = b$ with $x = (1, 1, \ldots, 1)^T$ from a given nonsingular matrix A.

4.1 General Matrix

First, we assume that the condition number of a given matrix A is at most smaller than $\mathbf{u}^{-1}$. Next, A is divided such that $A = A' + \Delta$, $A', \Delta \in \boldsymbol{F}^{n\times n}$. We adopt the error-free transformation introduced in [4, Algorithm 3.2] to obtain A' and Δ. Let a vector β be defined as

$$\beta_i := \lceil \log_2 n_i \rceil, \tag{12}$$

where n_i $(\le n)$ is the number of non-zero elements in i-th row in the matrix A. Alternatively, we can set $\beta_i = \lceil \log_2 n \rceil$ for simple implementation. Let σ be defined as

$$\sigma_i := 2^{\beta_i} \cdot 2^{g_i}, \quad g_i := \lceil \log_2 \max_{1\le j\le n} |a_{ij}| \rceil, \tag{13}$$

where $\max\limits_{1\le j\le n} |a_{ij}| \ne 0$ from the assumption $\det(A) \ne 0$. Let $e = (1, 1, \ldots, 1)^T \in \boldsymbol{F}^n$. The matrices A' and Δ are obtained by

$$A' := \text{fl}\left(\left(A + \sigma \cdot e^T\right) - \sigma \cdot e^T\right), \quad \Delta := \text{fl}\left(A - A'\right). \tag{14}$$

We now prove that $A'x = \text{float}\,(A'x)$.

Theorem 4.1 *Let A' be obtained by (14) for a given $A \in \boldsymbol{F}^{n\times n}$ and $x := (1, 1, \ldots, 1)^T$. Then, $A'x = \text{float}\,(A'x)$.*

Proof.

Since $x = (1, 1, \ldots, 1)^T$, the problem is to prove float $\left(\sum_{j=1}^{n} a'_{ij}\right) = \sum_{j=1}^{n} a'_{ij}$, for all i.

If $n_i > \mathbf{u}^{-1}$, then all a'_{ij} become zero, so that there is no rounding error in float $(A'x)$. Hereafter, we assume $n_i \le \mathbf{u}^{-1}$. From (2.1), we have

$$a'_{ij} \in \mathbf{u}\sigma_i\mathbb{Z}, \quad |a'_{ij}| \le 2^{-\beta_i}\sigma_i. \tag{15}$$

From the definition of β_i in (12), we have

$$n_i 2^{-\beta_i} \leq 1. \tag{16}$$

From (15), (16), and the assumption on n_i, we obtain

$$\mathbf{u}\sigma_i\mathbb{Z} \ni \text{float}\left(\sum_{j=1}^{n} |a'_{ij}|\right) \leq \text{float}\left(\sum_{j=1}^{n_i} 2^{-\beta_i}\sigma_i\right) \leq n_i 2^{-\beta_i}\sigma_i \leq \sigma_i. \tag{17}$$

This satisfies the assumption of Lemma 2.1, so there is no rounding error in float $(A'x)$.
□

Because the vector b is obtained by float $(A'x)$, $A'x = b$ is satisfied.

4.2 Structured Matrix

If a matrix A has a structure which gives rules for $a_{ij} = a_{kl}$, e.g., symmetric, persymmetric, Toeplitz, Hankel, and so forth, the structure of A' obtained in (14) is often different to that of A. In this subsection, we proposed a method to preserve the structure of A.

We use the constant β_i defined in (12) and the vector σ defined in (13). Let Q_{ij} be a set of indices, for example, $Q_{ij} = \{i, j\}$ for a symmetric matrix. Matrices A' and Δ are obtained as follows:

$$A' := \text{fl}\left((A + F) - F\right), \quad \Delta := \text{fl}\left(A - A'\right), \quad F \in \boldsymbol{F}^{n\times n}, \quad f_{ij} = \max_{k \in Q_{ij}} \sigma_k. \tag{18}$$

Since $f_{ij} \geq \sigma_i$ for all i and j, we can prove $\text{fl}(A'x) = A'x$ similar to Theorem 4.1. If we simply set $Q_{ij} = \{1, 2, \ldots, n\}$, then $a_{ij} = a_{kl} \Rightarrow a'_{ij} = a'_{kl}$ is satisfied, since the computations for a_{ij} and a_{kl} in (18) are the same. Therefore, the structures are preserved. It means that

$$\sigma := \max_{1 \leq i \leq m} 2^{\beta_i} \cdot 2^{g_i}, \quad g_i := \lceil \log_2 \max_{1 \leq j \leq n} |a_{ij}| \rceil$$

is defined, and

$$A' := \text{fl}\left((A + \sigma E) - \sigma E\right), \quad \Delta := \text{fl}\left(A - A'\right),$$

where $e_{ij} = 1$ for all (i, j) pairs. For a simple implementation, we can set

$$\sigma := 2^{\lceil \log_2 n \rceil} \cdot 2^h, \quad h := \lceil \log_2 \max_{1 \leq i,j \leq n} |a_{ij}| \rceil.$$

This approach cannot be directly applied to a skew symmetric matrix, e.g.,

$$\sigma_i = 1,\ a_{ij} = \mathbf{u} \Rightarrow a'_{ij} = 0, \quad \sigma_i = 1,\ a_{ji} = -\mathbf{u} \Rightarrow a'_{ji} = -\mathbf{u}.$$

For this matrix, we compute only a'_{ij} $(i \leq j)$, and a'_{ji} is obtained by $-a'_{ij}$.

4.3 Preserving Positive Definiteness

Let a matrix A be symmetric and positive definite. If a matrix A' is obtained by the methods introduced in subsections 4.2, the matrix A' may not be a positive definite. Hence, we proposed a method which preserves positive definiteness of a matrix. First, we compute a matrix B by

$$B := \text{fl}\left((A + 2F) - 2F\right), \quad \Delta := \text{fl}\left(A - B\right), \tag{19}$$

where the matrix F was defined in (18). Then $A = B + \Delta$. We set a diagonal matrix Δ' as

$$\delta'_{ii} := 2n\mathbf{u}f_{ii}, \quad 2n\mathbf{u} \le 1,$$

and we compute A' by

$$A' := \mathrm{fl}\left(B + \Delta'\right). \tag{20}$$

For the proof of the positive definiteness, we review two well-known lemmas in linear algebra.

Lemma 4.1 *Let $A = A^T$ and $B = B^T \in \boldsymbol{R}^{n\times n}$. If both A and B are positive definite, then $A + B$ is positive definite.*

Lemma 4.2 *For $A = A^T \in \boldsymbol{R}^{n\times n}$, if A is diagonal dominant, and all diagonal entries in A are positive, then A is positive definite.*

The following theorem explains why A' in (20) is also positive definite.

Theorem 4.2 *Assume that $2n_i\mathbf{u} \le 1$ for all i. For A' in (20) and $x = (1, 1, \ldots, 1)^T$, $\mathrm{fl}\,(A'x) = A'x$, and A' is symmetric and positive definite if A is symmetric and positive definite.*

Proof.

First, we prove $A' := \mathrm{fl}\,(B + \Delta') = B + \Delta'$. For off-diagonal elements in A', $a'_{ij} = b_{ij} + \delta'_{ij}$ is trivially proved since Δ' is a diagonal matrix. From (2.1), we have

$$b_{ij} \in 2\mathbf{u}f_{ij}\mathbb{Z}, \quad |b_{ij}| \le 2^{-\beta_i} f_{ij}. \tag{21}$$

Then, $2\mathbf{u}f_{ij}\mathbb{Z} \ni b_{ij} + \delta'_{ii} \le 2f_{ij}$ from the assumption on n_i. Hence, Lemma 2.1 says

$$b_{ij} + \delta'_{ii} = \mathrm{fl}\left(b_{ij} + \delta'_{ii}\right). \tag{22}$$

From (21), (22), and the assumption of n_i, we have

$$\begin{aligned} 2\mathbf{u}f_{ij}\mathbb{Z} \ni \mathrm{float}\left(\sum_{j=1}^{n} a'_{ij}\right) \quad &\le \quad \mathrm{float}\left(\left(2^{-\beta_i} f_{ij} + 2n_i\mathbf{u}f_{ij}\right)\right. \\ &\qquad\qquad \left. + 2^{-\beta_i} f_{ij} + \ldots + 2^{-\beta_i} f_{ij}\right) \\ &= \quad n_i 2^{-\beta_i} f_{ij} + 2n_i\mathbf{u}f_{ij} \le 2f_{ij}. \end{aligned}$$

Therefore, no rounding error is produced in the evaluation of $\sum_{j=1}^{n} a'_{ij}$. The rest of this proof is written for the positive definiteness of A'. From the computations (19) and (20), we have $A' = B + \Delta' = A - \Delta + \Delta'$. Since $-\Delta + \Delta'$ is diagonal dominant with positive diagonal entries, Lemmas 4.1 and 4.2 prove the positive definiteness of A'. □

4.4 Improvement by Iterations

We show how to make Δ in (14) as small as possible. A matrix A' is obtained by

$$a'_{ij} := \mathrm{fl}\left((a_{ij} + \frac{\sigma_i}{w_i}) - \frac{\sigma_i}{w_i}\right), \ \delta_{ij} := \mathrm{fl}\left(a_{ij} - a'_{ij}\right), \ w_i := 2^{k_i}, \ k_i \in \mathbb{N} \cup \{0\}, \tag{23}$$

where the vector σ is defined in (13). If we set $w_i := 1$ for all i, then $A'x = \mathrm{float}\,(A'x)$ is guaranteed by Theorem 4.1. The concern is to check whether a rounding error occurs in the evaluation of $A'x$ setting $w_i \geq 2$. It is possible to prove $A'x = \mathrm{fl}\,(A'x)$ using the methods introduced in Section 3. We employ a trial-and-error approach for the following two cases.

- The structure is preserved: If $\mathrm{fl}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) = \sum_{j=1}^{n} a'_{ij}x_j$ for $\forall i$, then $w_i := 2 * w_i$ for all i and compute (23). These procedures are continued until $\mathrm{fl}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) \neq \sum_{j=1}^{n} a'_{ij}x_j$ for $\exists i$.
- The structure need not to be preserved: Set

$$K = \left\{ i \mid \mathrm{fl}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) = \sum_{j=1}^{n} a'_{ij}x_j \right\}.$$

  We update $w_i := 2 * w_i$ for all $i \in K$ and compute (23). These procedures are continued until $\mathrm{fl}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) \neq \sum_{j=1}^{n} a'_{ij}x_j$ for $\forall i$.

After these procedures, we set $w_i := w_i/2$ and compute (23).

4.5 MATLAB Algorithms

We introduce several algorithms for generating A' and b such that $A'x = b$ for $x = (1, 1, \ldots, 1)^T$. All algorithms are written in MATLAB-like code. In algorithms, we use a matrix C instead of the matrix A', because the meaning of A' in MATLAB is the transposed matrix of A. First, we introduce algorithms based on Sections 4.1 and 4.2.

Algorithm 2 *The following algorithm produces $C \in \boldsymbol{F}^{n\times n}$ and $b \in \boldsymbol{F}^n$ from a non-singular matrix $A \in \boldsymbol{F}^{n\times n}$ such that $Cx = b,\ x = (1, 1, \ldots, 1)^T$.*

```
function [C, b] = generate_ones(A)
  n = size(A, 1);
  y = max(abs(A), [ ], 2);
  σ = 2ˆceil(log2(n)) * 2.ˆceil(log2(y));
  T = repmat(σ, 1, n);
  C = (A + T) − T;
  b = C * ones(n, 1);
end
```

Algorithm 3 *The following algorithm produces $C \in \boldsymbol{F}^{n\times n}$ and $b \in \boldsymbol{F}^n$ from a non-singular matrix $A \in \boldsymbol{F}^{n\times n}$ such that $Cx = b,\ x = (1, 1, \ldots, 1)^T$. If $a_{ij} = a_{kl}$, then*

$c_{ij} = c_{kl}$.

```
function [C, b] = generate_ones_str(A)
  n = size(A, 1);
  y = max(abs(A(:)));
  σ = 2ˆceil(log2(n)) * 2.ˆceil(log2(y));
  C = (A + σ) − σ;
  b = C * ones(n, 1);
end
```

Next, we introduce an algorithm for a sparse matrix.

Algorithm 4 *The following algorithm produces $C \in \boldsymbol{F}^{n\times n}$ and $b \in \boldsymbol{F}^n$ from a non-singular sparse matrix $A \in \boldsymbol{F}^{n\times n}$ such that $Cx = b,\ x = (1, 1, \ldots, 1)^T$. If $a_{ij} = a_{kl}$, then $c_{ij} = c_{kl}$. Namely, the structure of A and C is the same.*

```
function [C, b] = generate_ones_sp(A)
  n = size(A, 1);
  y = max(abs(A(:)));
  σ = 2ˆceil(log2(n)) * 2ˆceil(log2(y));
  T = σ * spones(A);
  C = (A + T) − T;
  b = C * ones(n, 1);
end
```

Finally, we introduce the iterative refinement based on discussion in Section 4.4.

Algorithm 5 *The following algorithm produces C and b from a non-singular matrix A such that $Cx = b,\ x = (1, 1, \ldots, 1)^T$. The structure of C is the same to that of A.*

```
function [C, b] = generate_ones_itr(A)
  n = size(A, 1);   x = ones(n, 1);   r = check(C, x);
  if sum(r) == n
    C = A;   b = A * x;   return;
  end
  y = max(abs(A(:)));
  σ = 2ˆceil(log2(n)) * 2ˆceil(log2(y))/2;
  while 1
    C = (A + σ) − σ;
    r = check(C, x);
    if sum(r) ∼= n,   break;,   end
    σ = σ/2;
  end
  C = (A + 2 * σ) − 2 * σ;
  b = C * ones(n, 1);
end
```

In the beginning of Algorithm 5, we check whether rounding errors occur in $\mathrm{fl}(Ax)$. If this check is not applied, then the program may not terminate due to an infinite loop. For example, if we set

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \sigma = \begin{pmatrix} 2^k \\ 2^k \end{pmatrix},\ k \in \mathbb{N}.$$

then $\mathrm{fl}(A'x) = Ax$ is satisfied for all $k \in \boldsymbol{F}$. If $k < -1074$ for binary64, then 2^k is rounded to zero, so that the computation never terminates.

We now show numerical results. Floating-point matrices are generated by

$$A = \texttt{gallery}('randsvd', n, cnd, 3, n, n, 1); \tag{24}$$

in MATLAB. We present relative change of the condition number

$$\frac{|\texttt{cond}(A) - \texttt{cond}(A')|}{\texttt{cond}(A)}$$

by methods with and without the iterative refinement in Fig. 2 for $n = 1000$ and $n = 10000$. The figures indicate that the iterative refinement is effective.

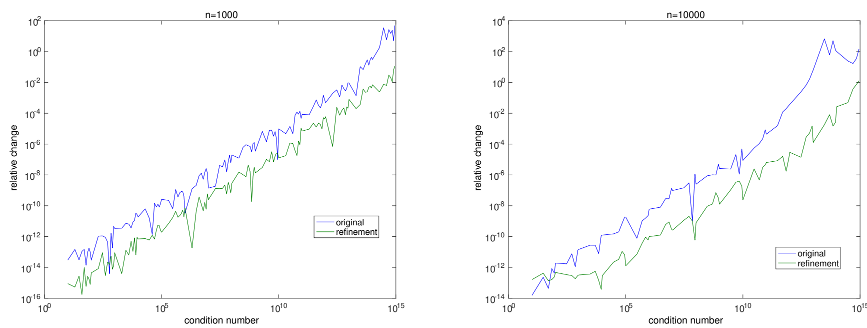

Figure 2: Relative changes of the condition number (left: $n = 1000$, right: $n = 10000$)

Note that even if the condition number of the matrix A is greater than $\mathbf{u}^{-1}$, it is possible to obtain A' such that $A'x = b$. However, the condition number of A is much different from that of A' in many cases.

5 Generalized Method

In the previous section, we only set $x = (1, 1, \cdots, 1)^T$ as the exact solution. Here, we extend this construction to any given x. Assume that a coefficient matrix A is non-singular, and a vector $x \neq 0$. Set the vector θ as

$$\begin{cases} \theta_j = 2^{k_j},\ k_j \in \mathbb{Z}, \quad x_j \in \theta_j \mathbb{Z}, \quad x_j \notin 2\theta_j \mathbb{Z}, & \text{if } x_j \neq 0, \\ \theta_j = 0 & \text{otherwise.} \end{cases} \tag{25}$$

Let several constants be defined as

$$\sigma_i := 2^{\beta_i} \cdot 2^{g_i}, \quad g_i := \begin{cases} \lceil \log_2 \varphi_i \rceil, & \varphi_i \neq 0 \\ 0, & \text{otherwise} \end{cases}, \quad \varphi_i := \max_{1 \le j \le n} |a_{ij} x_j|. \tag{26}$$

If the structure of the matrix needs to be preserved, then let φ be

$$\varphi_i := \max_{1 \le i,j \le n} |a_{ij} x_j|$$

instead of (26). We set a vector σ' as

$$\sigma' := \frac{2}{\min_j \theta_j} \cdot \sigma. \tag{27}$$

The matrix A' is obtained by

$$a'_{ij} := \mathrm{fl}\left(\left(a_{ij} + \sigma'_i\right) - \sigma'_i\right). \tag{28}$$

Here, we introduce the following lemma for a relation of a_{ij} and a'_{ij}.

Lemma 5.1 *For a'_{ij} in (28), $|a'_{ij}| \le 2|a_{ij}|$ is satisfied.*

Proof.

From the definition of σ' in (27), $\sigma'_i > |a_{ij}|$ holds for all i. If $\mathbf{u}\sigma'_i > |a_{ij}|$, then $\mathrm{fl}\,(\sigma'_i + a_{ij}) = 0$ yields $a'_{ij} = 0$, and the lemma is proven trivially. Hence, we assume $\mathbf{u}\sigma'_i \le |a_{ij}| < \sigma'_i$. We define δ_{ij} as a rounding error for $\mathrm{fl}\,(\sigma'_i + a_{ij})$ such that

$$\mathrm{fl}\left(\sigma'_i + a_{ij}\right) = \sigma'_i + a_{ij} + \delta_{ij}, \quad |\delta_{ij}| \le \mathbf{u} \cdot \mathtt{ufp}\left(\sigma'_i + a_{ij}\right),$$

which is obtained by Theorem 2.2. Since σ'_i is a power of two, $\mathtt{ufp}\,(\sigma'_i + a_{ij}) = \sigma'_i$. Therefore, $|\delta_{ij}| \le \mathbf{u}\sigma'_i$. From Theorem 2.1, we have

$$a'_{ij} = \mathrm{fl}\left(\left(\sigma'_i + a_{ij}\right) - \sigma'_i\right) = \mathrm{fl}\left(\sigma'_i + a_{ij}\right) - \sigma'_i = a_{ij} + \delta_{ij}.$$

Finally, we obtain

$$|a'_{ij}| = |a_{ij} + \delta_{ij}| \le |a_{ij}| + |\delta_{ij}| \le |a_{ij}| + \mathbf{u}\sigma'_i \le 2|a_{ij}|.$$

□

The following theorem proves that $A'x = \mathrm{float}\,(A'x)$ is satisfied.

Theorem 5.1 *Assume that $n_i\mathbf{u} \le 1$ for all i. For the matrix A' obtained by (28), $A'x = \mathrm{float}\,(A'x)$ is satisfied.*

Proof.

If $\varphi_i = 0$, then $\sum_{j=1}^{n} a'_{ij}x_j = \mathrm{float}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) = 0$, so that we assume $\varphi_i \ne 0$ hereafter. Using Lemma 3.3 in [4], $a'_{ij} \in \mathbf{u}\sigma'_i$ for all (i, j) pairs. Therefore, from (25), we have

$$a'_{ij}x_j \in \mathbf{u}\sigma'_i\theta_j\mathbb{Z} \subseteq \mathbf{u}\sigma'_i \cdot \min_j \theta_j\mathbb{Z}. \tag{29}$$

From the definition of σ in (26),

$$\sigma_i = 2^{\beta_i} \cdot 2^{\lceil \log_2 \max |a_{ij}x_j| \rceil} \ge 2^{\beta_i} \cdot 2^{\log_2 \max |a_{ij}x_j|} = 2^{\beta_i} \cdot \max |a_{ij}x_j| \ge 2^{\beta_i}|a_{ij}x_j|. \tag{30}$$

Using (30) and (27), an upper bound of $|a_{ij}x_j|$ is obtained as

$$|a_{ij}x_j| \le \frac{1}{2^{\beta_i}}\sigma_i = \frac{1}{2^{\beta_i+1}}\sigma'_i \min_j \theta_j. \tag{31}$$

From Lemma 5.1 and (31), we have

$$|a'_{ij}x_j| \le 2|a_{ij}x_j| \le \frac{1}{2^{\beta_i}}\sigma'_i \min_j \theta_j. \tag{32}$$

From (29) and (32), $\mathrm{fl}\left(a'_{ij}x\right) = a'_{ij}x$ by Lemma 2.1. Hence, using the assumption of n_i, we finally obtain

$$\mathbf{u}\sigma'_i \cdot \min_j \theta_j \mathbb{Z} \ni \mathrm{float}\left(\sum_{j=1}^{n} a'_{ij}x_j\right) \leq \mathrm{float}\left(\sum_{j=1}^{n} \mathrm{fl}\left(|a'_{ij}x_j|\right)\right) \leq \frac{n_i}{2^{\beta_i}} \min_j \theta_j \sigma'_i \leq \min_j \theta_j \sigma'_i.$$

From this and Lemma 2.1, it is proved that no rounding error occurs in $\mathrm{fl}\,(A'x)$. □

If $\mathtt{ufp}(x_i)/\theta_i$ is large, then σ'_i becomes huge compared to $|a_{ij}|$. In the worst case, A' becomes the zero-matrix. For example, setting $x_i = 1 + 2\mathbf{u}$ makes A' the zero matrix. It is possible to preserve positive definiteness by the diagonal shift as in the discussion in Section 3.3.

6 Applications of Linear Systems with Exact Solutions

In this section, we apply linear systems with exact solutions to verified numerical computations and iterative methods in turn.

6.1 Application to Verified Numerical Computations

We examine upper bounds obtained by verified numerical computations. We focus on a method based on

$$\|\widehat{x} - A^{-1}b\|_\infty \leq \frac{\|R(A\widehat{x} - b)\|_\infty}{1 - \|RA - I\|_\infty} \leq \alpha \in \boldsymbol{F}, \quad \|RA - I\|_\infty < 1, \quad R \approx A^{-1}, \tag{33}$$

where $\widehat{x} \in \boldsymbol{F}^n$ is an approximate solution, and I is the identity matrix. An accurate dot product algorithm with error bounds [6, Dot2Err] is used for enclosure of $A\widehat{x} - b$ in (33). We generate matrices using (24) and execute Algorithm 5. We set $\widehat{x} = (c, c, \dots, c)^T \in \boldsymbol{F}^n$. Figures 3 and 4 show a ratio

$$\frac{\alpha}{\|\widehat{x} - A^{-1}b\|_\infty} = \frac{\alpha}{|1 - c|} \tag{34}$$

for $c = 1 + 2\mathbf{u} \cdot 10^p,\ p = 0, 3, 6, 9$ and $n = 1000$ (100 examples for each condition number: $\mathtt{cond(A)} = i \cdot 10^j,\ i = \{1, 2, \dots, 9\},\ j = \{1, 2, \dots, 12\}$). Table 1 shows the minimum, median, average, maximum of the ratio (34).

Table 1: Comparison of the ratio (34)

p	minimum	median	average	maximum
0	1.0000000012	1.00098	44.7	2721
3	1.0000000012	1.0000013	1.053	4.25
6	1.0000000000024	1.00000026	1.0081	1.18
9	1.0000000000024	1.00000026	1.0082	1.18

If $\mathtt{cond}(A) < 10^{10}$, the ratio (34) becomes very close to 1, namely, an upper bound α expresses almost the exact error. If an approximate solution is inaccurate, then the verification method produces a reasonable upper bound, even if a generated matrix is ill-conditioned. For example, if $c = 1 + 2\mathbf{u} \cdot 10^6$ or $c = 1 + 2\mathbf{u} \cdot 10^9$, the ratio (34) is less than 1.2, which is independent of the condition number. However, if an approximation is very accurate ($c = 1 + 2\mathbf{u}$), and the condition number of A is large, then it is observed that the ratio (34) is over 2000 in the worst case. Therefore, the error bound α is overestimated in this case.

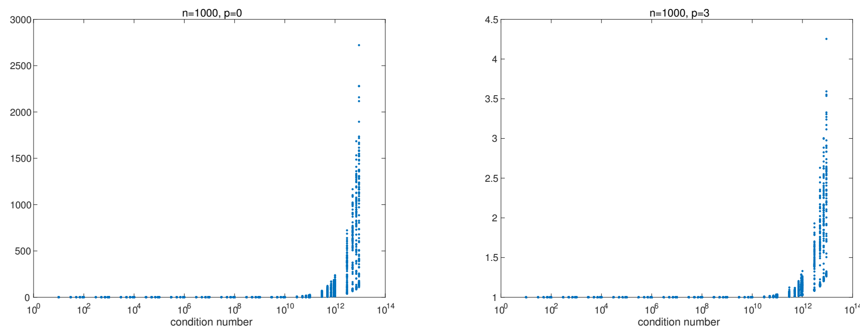

Figure 3: $c = 1 + 2\mathbf{u}$ (left) and $c = 1 + 2 \cdot 10^3\mathbf{u}$ (right)

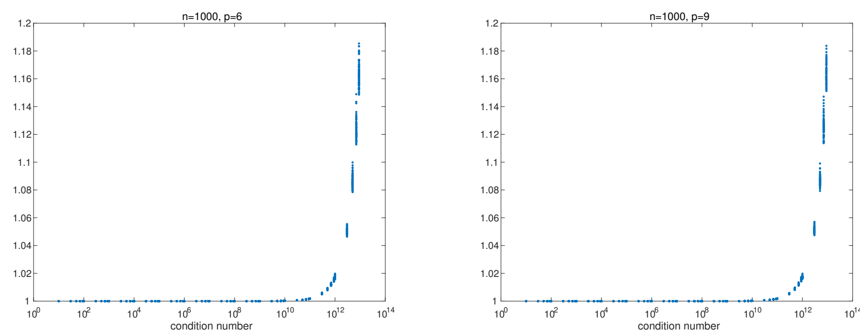

Figure 4: $c = 1 + 2 \cdot 10^6\mathbf{u}$ (left) and $c = 1 + 2 \cdot 10^9\mathbf{u}$ (right)

6.2 Application to Iterative Methods

We introduce an application to iterative methods for a linear system $Ax = b$. Let $\widehat{x}$ be an approximate solution. For checking the convergence of iterative methods, the residual norm $\|A\widehat{x} - b\|_2$ is usually used. However, there are sometimes gaps between the residual and the error. Therefore, if linear systems with exact solutions are given by the proposed methods, then we can check the error norm $\|x - \widehat{x}\|_2$ exactly, and it is useful for researchers working on the iterative methods.

We examine the conjugate gradient method (CG method) with an initial vector

x_0 as follows.

$r_0 = b - Ax_0;$
$p_0 = r_0;$
while 1
  $\alpha_k = (r_k^T r_k)/(p_k^T A p_k);$
  $x_{k+1} = x_k + \alpha_k p_k;$
  $r_{k+1} = r_k - \alpha_k A p_k;$
  if $\|r_{k+1}\|_2 < 1e - 15 * \|b\|_2$
    break;
  end
  $\beta_k = (r_{k+1}^T r_{k+1})/(r_k^T r_k);$
  $p_{k+1} = r_{k+1} + \beta_k p_k;$
end

We check the relative residual norm, the stopping criterion, and the relative error norm such as

$$\frac{\|b - A\widehat{x}_k\|_2}{\|b\|_2}, \quad \frac{\|r_{k+1}\|_2}{\|b\|_2}, \quad \frac{\|\widehat{x} - x\|_2}{\|x\|_2}$$

by numerical examples. Test matrices are obtained from Matrix Market [7]. A' and b are generated by Algorithm 5. Then, $A'x = b$ with $x = (1, 1, \ldots, 1)^T$. Figures 5 and 6 show the relative residual norm, the stopping criterion, and the relative error norm for several matrices for the CG method. The initial vector is $x_0 = (0, 0, \ldots, 0)^T$. We observe that the residual norm and the value for the stopping criterion are both small, but the error norm is relatively large in Table 5 (bcsstk15). In addition, the value for the stopping criterion decreases, but the error norm is not changed in Table 6 (nos7).

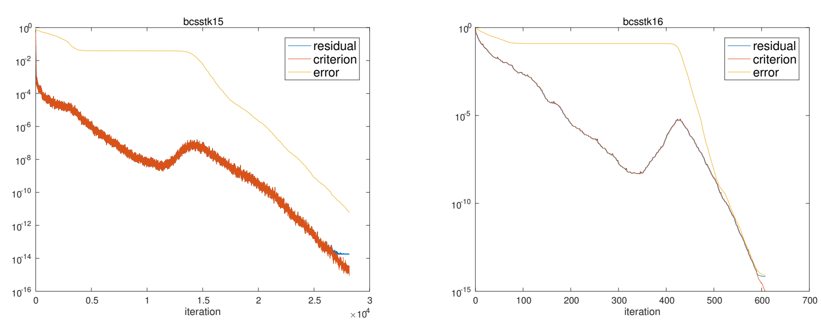

Figure 5: bcsstk15 and bcsstk16 from Matrix Market

One may think that analysis for $A'x = b$ is not useful because the original coefficient matrix is A. Therefore, we next check the difference of the convergence between $Ax = b$ and $A'x = b$. The matrix A is bcsstk15 from Matrix Market, and we set all $x_i = 2(1 - 2^{-k}) = \sum_{j=1}^{k} 2^{-j+1}$. The leading k bits in the significand in x_i are 1's, and the rest in the significand are 0's. Figures 7 – 9 show the behavior of the convergence for several k. If $k = 5$ and $k = 20$, the behaviors of the CG method for $Ax = b$ and $A'x = b$ are almost identical. However, the result is meaningless for $k = 35$, since there are big differences between A and A'.

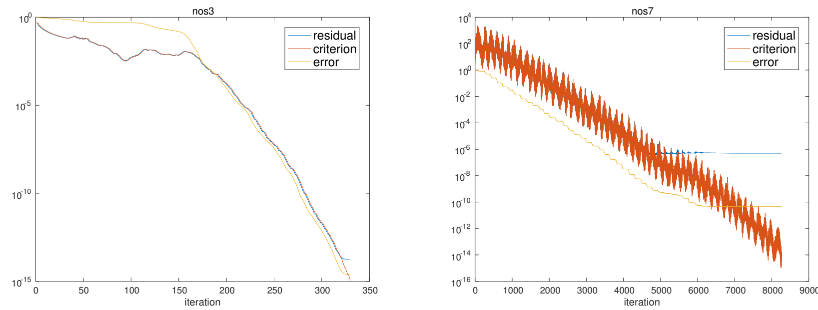

Figure 6: nos3 and nos7 from Matrix Market

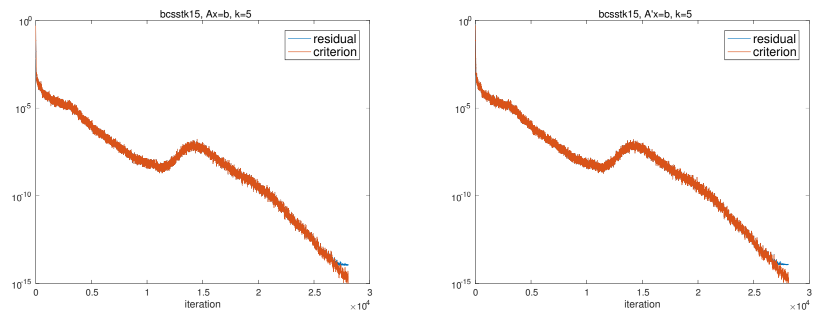

Figure 7: $Ax = b$ and $A'x = b$ for $k = 5$

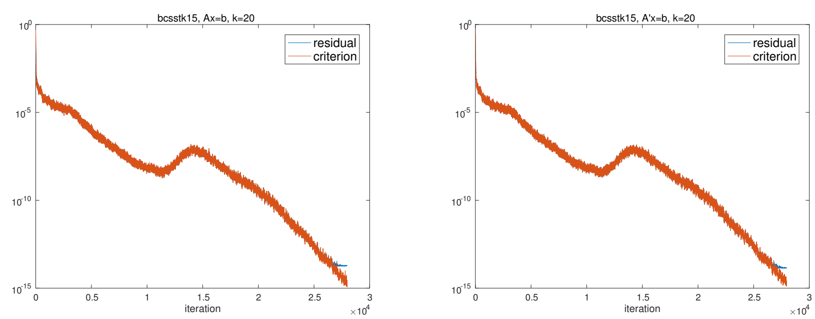

Figure 8: $Ax = b$ and $A'x = b$ for $k = 20$

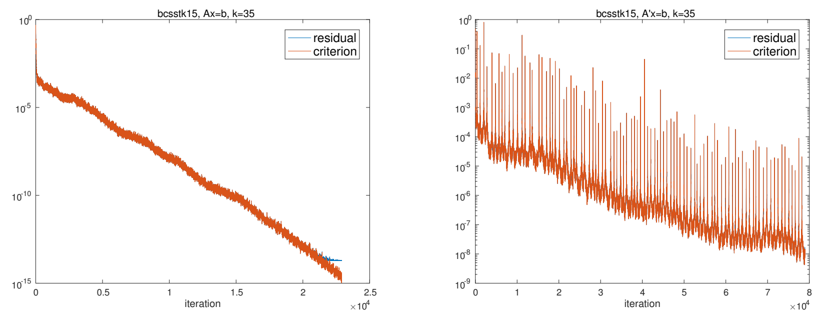

Figure 9: $Ax = b$ and $A'x = b$ for $k = 35$

Conclusion

We proposed a method to produce a linear system with the exact solution. The system is useful for checking the overestimation of verified numerical computations. We hope our method contributes the progress of iterative methods for linear systems.

We do not think that our method is an unique method for obtaining $A'x = b$. The alternative is to replace some of the significant bits of a_{ij} by 0 properly, or to transform integer data for A after proper scaling for the matrix A. Our methods use only matrix operations, so they are easy to implement in MATLAB or C with BLAS.

Acknowledgements

The authors appreciate the constructive comments by the reviewers. This research is partially supported by JST / CREST and JSPS KAKENHI Grant Number 16H03917.

References

[1] ANSI: IEEE Standard for Floating-Point Arithmetic, Std 754–2008, 2008.

[2] S. Miyajima, T. Ogita, S. Oishi. A method of generating linear systems with an arbitrarily ill-conditioned matrix and an arbitrary solution, *Proc. 2005 International Symposium on Nonlinear Theory and its Applications (NOLTA 2005)*, Bruges, Belgium, pp. 741-744, October 2005

[3] C.-P. Jeannerod and S.M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM. J. Matrix Anal. & Appl.*, 34(2):338–344, 2013.

[4] S.M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189—224, 2008.

[5] T. J. Dekker. A floating-point technique for extending the available precision, *Numer. Math.*, 18(3):224-242, 1971.

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

[7] Matrix Market: http://math.nist.gov/MatrixMarket/

[8] K. Ozaki, T. Ogita, S. Oishi. Error-free transformation of matrix multiplication with a posteriori validation, *Numer. Lin. Algebra Appl.*, 23(5):931-946, 2016.

[9] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[10] S. Winograd. On multiplication of 2x2 matrices. *Linear Algebra Appl.*, 4:381–388, 1971.