Accurate and efficient algorithm for solving ill-conditioned linear systems by preconditioning methods

Yuka Kobayashi\(^1\)\(^a\) and Takeshi Ogita\(^1\)

\(^1\)Division of Mathematics, Graduate School of Science, Tokyo Woman’s Christian University
2–6–1 Zempukuji, Suginami-ku, Tokyo 167–8585, Japan

\(^a\)yukako.109b@gmail.com

Received January 4, 2016; Revised March 10, 2016; Published July 1, 2016

Abstract: In this paper, we develop an accurate and efficient algorithm for solving ill-conditioned linear systems. For this purpose, we propose two preconditioning methods that are based on LU factorization. One is the method using the inverse of an LU factor. The other is the method using the residual of an LU factorization. The latter method requires less computational cost than the former one. Using an LU factorization with the iterative refinement, we can accurately solve a linear system \(Ax = b\) for \(\kappa(A) \lesssim u^{-1}\), where \(u\) is the relative rounding error unit in working precision. If we use the proposed algorithm with accurate dot product, we can obtain an accurate approximate solution for ill-conditioned linear systems beyond the limit of the working precision. Results of numerical experiments show that the proposed algorithm can work for \(\kappa(A) \lesssim u^{-2}\) in reasonable computing time.

Key Words: accurate numerical algorithm, solution of linear systems, preconditioning technique, ill-conditioned problem, floating-point arithmetic

1. Introduction

We present an accurate and efficient algorithm for solving an ill-conditioned linear system

\[ Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad b \in \mathbb{R}^n \]  

by using floating-point arithmetic. The relative rounding error unit of floating-point arithmetic is denoted by \(u\). We assume IEEE standard 754 binary64 (double precision) to be working precision. Then, \(u = 2^{-53} \approx 10^{-16}\).

Let \(\kappa(A)\) be the condition number of \(A\) such that

\[ \kappa(A) = \|A\| \cdot \|A^{-1}\|, \]

where \(\| \cdot \|\) denotes the spectral norm. If \(\kappa(A)\) is too large to solve (1), \(A\) is considered as ill-conditioned. If that is the case, then a numerical solution \(\hat{x}\) of (1) tends to be inaccurate. Such ill-conditioned
(or ill-posed) problems often arise in inverse problems (cf. e.g. [1]). In this paper, we deal with ill-conditioned problems such as

\[ \kappa(A) \leq (u^{-1})^2 \approx 10^{32}. \]

There are two standard methods to solve (1) accurately. One is the method using an LU factorization with the iterative refinement method. If \( \kappa(A) \) is not too large, the method is effective. Otherwise the method cannot work well. The other is an LU factorization using multiple-precision arithmetic. The method can work even if \( \kappa(A) \) is ill-conditioned. However, it takes significant computing time in spite of the magnitude of \( \kappa(A) \). To remedy these defects of the standard methods, we introduce preconditioning techniques using a result of an LU factorization by floating-point arithmetic.

In about 1984, Rump [2] presented an interesting algorithm obtaining approximate inverses of ill-conditioned matrices. The basis of this algorithm is the multiplicative corrections of approximate inverses using accurate dot product. Rump [3] also showed an algorithm for solving ill-conditioned linear systems using his algorithm for the accurate matrix inversion. Let \( R \) denote an approximate inverse of \( A \). If we use \( R \) as a preconditioner for \( A \), (1) can be transformed into

\[ RAx = Rb. \] (2)

Then the condition number of \( RA \) can be reduced by \( u \) such as

\[ \kappa(RA) \approx 1 + u\kappa(A). \]

Consequently, (2) becomes more well-conditioned than (1).

Ogita [4] showed that an approximate inverse of an LU factor of \( A \) can be used instead of the approximate inverse of \( A \). Assume \( A \approx LU \) with \( \kappa(A) \approx \kappa(L) \). If we use \( L^{-1} \) as a left preconditioner, the linear system (1) can be transformed into

\[ L^{-1}Ax = L^{-1}b. \] (3)

Then the condition number of \( L^{-1} \) can be reduced by \( u \) such as

\[ \kappa(L^{-1}A) \approx 1 + u\kappa(A). \]

For the purpose of this paper, we adopt the method in [4] and introduce two preconditioning methods. One is the method using explicitly obtained \( L^{-1} \). The other is the method using a residual of an LU factorization without calculating \( L^{-1} \) in (3). The latter requires less computational cost than the former.

This paper is organized as follows: In Section 2, we present an algorithm for solving ill-conditioned linear systems. In Section 3, we discuss two preconditioning methods and their computational cost. Section 4 shows results of numerical experiments by using the proposed methods. Finally, we conclude the paper in Section 5.

This paper is the vastly extended version of our previous paper [5]. There are two major differences. First, in this paper, we use more accurate computations for matrix multiplication than that in the previous paper. As a result, we can deal with more ill-conditioned problems. Second, we propose a new preconditioning method for solving ill-conditioned linear systems. Computational cost of this method becomes less than that of the method proposed in the previous paper.

2. Proposed algorithm

Let \( M \in \mathbb{R}^{n \times n} \) denote a left preconditioner of \( A \). Then (1) is transformed into

\[ MAX = Mb, \]

where it is expected that

\[ \kappa(MA) \ll \kappa(A). \]

We can choose some nonsingular matrix as \( M \), for example, an approximate inverse of \( A \), an LU factor or a QR factor. In this paper, we adopt the LU factor in terms of computational cost.
Let us consider the LU factorization of $A$ with partial pivoting such that $PA \approx LU$. By Doolittle’s method, $L$ becomes a unit lower triangular matrix. Then we know that the condition number of $A$ becomes

$$\kappa(A) \approx \min \{\kappa(U), u^{-1}\}$$

by heuristics (cf. e.g. [6, p.130]). Then $U^{-1}$ can work as a right preconditioner such that

$$\kappa(AU^{-1}) \approx 1 + u\kappa(A).$$

Namely, the condition number of $A$ can be reduced by a factor around $u$ down to 1 using $U$.

At present, LAPACK routines are well optimized for today’s computers. These routines are very fast. LAPACK adopts the Doolittle’s LU factorization method. To benefit from such LAPACK routines, we execute an LU factorization of $A^T$ to obtain a left preconditioning for $A$.

We present here an algorithm using preconditioning techniques.

Algorithm 1. The proposed algorithm using preconditioning techniques for accurate solutions of linear systems.

Part 1: The standard method solving $Ax = b$

Step 1. Execute an LU factorization of $A^T$ with partial pivoting by the Doolittle’s method. Then solve $A\hat{x} = b$ by forward and backward substitutions for obtaining its approximate solution.

Step 2. Apply the iterative refinement method (cf. e.g. [6, pp.126–127]) to the approximate solution obtained at Step 1. If the stopping criterion for the iterations is satisfied, then the algorithm successfully stops. Otherwise, go to Part 2.

Part 2: Preconditioning technique for reducing $\kappa(A)$

Step 3. Precondition $A$ to reduce the condition number of $A$ as $C := L^{-1}A$, where $L$ is the LU factor obtained at Step 1. Then, solve $Cx = d$ where $d := L^{-1}b$.

Step 4. Apply the iterative refinement method to the approximate solution.

In order to avoid extra computational cost, we intend to execute Part 2 only if $A$ is ill-conditioned.

In the following, we explain the details of Part 1 of the algorithm. Assume that Doolittle’s method for $A^T$ is used for a left preconditioning of $A$. We can also apply a similar way when using Crout’s method for $A$.

2.1 Step 1

To solve the linear system $Ax = b$, we execute a Doolittle’s LU factorization of $A^T$ with partial pivoting:

$$PA^T \approx LU.$$  

After that, we solve

$$U^TL^TPx = b,$$

and obtain

$$\hat{x} \approx P^TL^{-T}U^{-T}b.$$  

2.2 Step 2

By repeating the following steps starting from $k = 0$ and $\hat{x}^{(0)} := \hat{x}$, an accurate solution of $Ax = b$ can be obtained if $A$ is not ill-conditioned. First, we calculate the residual

$$e^{(k)} \approx b - A\hat{x}^{(k)}$$  

precisely by accurate computing. In this paper, we use the algorithm Dot2 in [7], which can calculate accurate dot product as if computed in quadruple precision. Next, we solve
by using the LU factors obtained at Step 1. Let $\hat{y}^{(k)}$ be an approximate solution of (5), then we update $\hat{x}^{(k)}$ by

$$\hat{x}^{(k+1)} = \hat{x}^{(k)} + \hat{y}^{(k)}.$$  

The more these steps repeat, the more accuracy of $\hat{x}$ gains up to the limit of computational precision. The iterations stop by any one of the following three reasons:

(S) Let $\epsilon$ denote a tolerance. If

$$|x_i^{(k+1)} - x_i^{(k)}| \leq \epsilon |x_i^{(k+1)}| \quad \text{for all } i,$$

then this algorithm successfully stops.

(F1) Let $\alpha$ be some constant with $0 < \alpha < 1$. If

$$\|\hat{y}^{(k)}\| \geq \alpha \|\hat{y}^{(k-1)}\|$$

is satisfied, then we go to Part 2. We recommend $\alpha = 0.1$. Even if (7) is satisfied with $\alpha > 0.1$, we guess $\hat{x}^{(k)}$ cannot be improved any more due to the ill-conditionedness of $A$.

(F2) Let $k_{\text{max}}$ be the maximum number of iterations. If the number of iterations reaches $k_{\text{max}}$, then we go to Part 2.

If (F1) or (F2) is fulfilled in Step 2, this algorithm fails and we cannot get an accurate solution.

### 3. Preconditioning methods

In this section, we concretely explain Part 2 of the proposed algorithm.

#### 3.1 Method A

We calculate $X_L \approx U^{-T}$ explicitly, and use $X_L$ as a left preconditioner of $A$. After that, we solve $X_L Ax = X_L b$. Here, the matrix product $X_L \cdot A$ should be calculated accurately. Then it becomes possible to obtain an accurate numerical solution of $Ax = b$ with reducing the condition number of $A$.

##### 3.1.1 Step 3

We try to reduce the condition number of $A$ by using a preconditioning technique. For this purpose, we adopt a result of an LU factorization of $A^T$ in Step 1 such as

$$X_L \approx U^{-T}. $$

Then we multiply $Ax = b$ by $X_L$ from the left. Let $C$ and $d$ denote an approximation of $X_L A$ and $X_L b$, respectively:

$$C \approx X_L A \quad \text{and} \quad d \approx X_L b.$$  

Here, $X_L A$ and $X_L b$ should be calculated accurately. To do this, we adopt the algorithm Dot2 in [7].

We now consider the linear system

$$C x = d. $$  

We expect $C$ to be not ill-conditioned. If that is the case, (10) can be solved using a standard way with an LU factorization and forward and backward substitutions.

Table I shows computational cost of Method A using Dot2. Total computational cost of Method A is $\frac{85}{7} n^3$ flops (floating-point operations). It costs about 22 times as much as computing time of an LU factorization. Moreover, if the Fused Multiply-Add (FMA) instruction is available, we can reduce the computational cost for Dot2. By using FMA for Dot2, the total computational cost of Method A becomes $\frac{20}{9} n^3$ flops.
### Table I. Computational cost of Method A.

| Operations with $O(n^3)$ flops | Computational Cost (flops) |
|--------------------------------|-----------------------------|
| LU factorization of $A^T$     | $\frac{2}{3}n^3$           |
| $X_L \approx U^{-T}$          | $\frac{1}{3}n^3$           |
| $X_LA$ using Dot2             | $\frac{25}{2}n^3 / \frac{10}{3}n^3$ (using FMA) |
| LU factorization of $C \approx X_LA$ | $\frac{2}{3}n^3$           |
| Total                         | $\frac{85}{6}n^3 / \frac{29}{3}n^3$ (using FMA for Dot2) |

#### 3.1.2 Step 4

This step is almost the same as Step 2 except

$$r^{(k)} \approx X_L \left(b - A\hat{x}^{(k)}\right).$$  \hspace{1cm} (11)

Note that (11) should be calculated by forward substitution using accurate computation based on Dot2.

#### 3.2 Method B

In this method, we solve

$$U^{-T}Ax = U^{-T}b$$  \hspace{1cm} (12)

without explicitly calculating $X_L \approx U^{-T}$. Instead, we calculate $U^{-T}A$ implicitly using a residual of an LU factorization. Here, the residual $PA^T - LU$ and $U^{-T}b$ should be calculated accurately. Using this method, the condition number of $A$ can be reduced, similar to Method A.

#### 3.2.1 Step 3

We define a residual of the LU factorization of $A^T$ such that

$$R := PAT - LU.$$  \hspace{1cm} (13)

Let $E \in \mathbb{R}^{n \times n}$ denote an error of an LU factorization satisfying

$$PAT = (L + E)U.$$  \hspace{1cm} (14)

Then $E$ is represented by a residual of an LU factorization such that

$$E = (PAT - LU)U^{-1} = RU^{-1}.$$  \hspace{1cm} (14)

In general, $E$ is not a lower triangular matrix. Now, we calculate $R$ in (13) accurately by Dot2 and obtain its approximation $\hat{R}$. Moreover we calculate $\hat{E} \approx \hat{RU}^{-1}$ in (14) by forward substitution in working precision. Therefore

$$U^{-T}A \approx (L + \hat{E})^TP,$$

and (12) is transformed into

$$(L + \hat{E})^TPx = U^{-T}b.$$  \hspace{1cm} (15)

We calculate

$$C \approx (L + \hat{E})^T$$

in working precision and

$$d \approx U^{-T}b$$  \hspace{1cm} (16)

by forward substitution using accurate computation based on Dot2. After that, we solve

$$CPx = d.$$  \hspace{1cm} (17)
Table II. Computational cost of Method B.

| Operations with $O(n^3)$ flops | Computational Cost (flops) |
|---------------------------------|----------------------------|
| LU factorization of $A^T$       | $\frac{2}{3}n^3$          |
| $\hat{R} \approx PA^T - LU$ using Dot2 | $\frac{25}{3}n^3 / \frac{10}{3}n^3$ (using FMA) |
| $\hat{E} \approx RU^{-1}$      | $n^3$                     |
| LU factorization of $C \approx (L + E)^T$ | $\frac{2}{3}n^3$          |
| Total                           | $\frac{32}{3}n^3 / \frac{17}{3}n^3$ (using FMA for Dot2) |

using an LU factorization of $C$ with partial pivoting:

$$P_2C \approx L_2U_2.$$ 

After that, we solve

$$L_2U_2Px = P_2d$$

and obtain

$$\hat{x} \approx P^T U_2^{-1} L_2^{-1} P_2d.$$ 

Table II shows computational cost of Method B using Dot2. Total computational cost of Method B is $\frac{32}{3}n^3$ flops. It costs 16 times as much as the time of an LU factorization. The computational cost of Method B is $7\frac{2}{3}n^3$ flops less than that of Method A. If FMA is available, the total computational cost of Method B becomes $\frac{17}{3}n^3$ flops by using FMA for Dot2.

3.2.2 Error Bound of $C$

We explain why $CP$ in (17) becomes a good approximation of $U^{-T}A$. We assume

$$\kappa(A) > u^{-1}. \tag{18}$$

Let $\Delta_C$ denote an error of $C$ such that

$$C = U^{-T}AP^T + \Delta_C.$$ 

We consider to estimate $\|\Delta_C\|$. First, we define $\Delta_T$, $\Delta_S$ and $\Delta_R$ as rounding errors such that

$$C = (L + \hat{E} + \Delta_T)^T,$$

$$\hat{E} = \hat{R}U^{-1} + \Delta_S,$$

$$\hat{R} = PA^T - LU + \Delta_R.$$ 

Then

$$C^T = L + (\hat{R}U^{-1} + \Delta_S) + \Delta_T$$

$$= L + (PA^T - LU + \Delta_R) U^{-1} + \Delta_S + \Delta_T$$

$$= PA^TU^{-1} + \Delta_R U^{-1} + \Delta_S + \Delta_T$$

and

$$\Delta_C^T = \Delta_R U^{-1} + \Delta_S + \Delta_T.$$ 

Therefore,

$$\|\Delta_C\| \leq \|\Delta_R U^{-1}\| + \|\Delta_S\| + \|\Delta_T\|. \tag{19}$$

We can obtain upper bounds of $\|\Delta_R U^{-1}\|$, $\|\Delta_S\|$ and $\|\Delta_T\|$ as follows. In spite of $\kappa(A)$, the following relations usually hold in practice:
\[ \|A\| = \|A^T\| \approx \|L\|\|U\|, \quad (20) \]
\[ \|A\| \lesssim \|U\|, \quad (21) \]
\[ \kappa(U) \approx \min\{\kappa(A), u^{-1}\}, \quad (22) \]
\[ \|PA^T - LU\| \approx u\|A\|, \quad (23) \]
\[ \|L\| \approx n. \quad (24) \]

From (18) and (22),
\[ \kappa(U) = \|U\|\|U^{-1}\| \approx u^{-1}. \]

First, we consider an upper bound of \(\|\Delta R^{-1}\|\). Since we use \texttt{Dot2} to calculate \(R := PA^T - LU\),
\[ \|\Delta R\| \approx u\|PA^T - LU\| + c_0 u^2\|A^T\| + |L||U|| \quad (c_0 \approx n). \]

Here, according to [7], \(c_0 \approx n^2\) in the sense of error bound. However, in fact an actual error often becomes \(O(n)\). Therefore \(c_0 \approx n\) is more practical. Then (20) and (22) yield
\[
\|\Delta R\| \approx u^2\|A\| + c_0 u^2\|A\| \
\approx c_1 u^2\|A\| \quad (c_1 \approx n),
\]
and
\[
\|\Delta R^{-1}\| \leq \|\Delta R\| \cdot \|U^{-1}\| \approx c_1 u^2\|A\| \cdot \frac{u^{-1}}{\|U\|} \lesssim c_1 u. \quad (26)
\]

Next, we consider an upper bound of \(\|\Delta S\|\). From (23) and (25),
\[ \|\hat{R}\| \leq \|PA^T - LU\| + \|\Delta R\| \
\lesssim u\|A\|. \]

Since \(\Delta S\) is a rounding error of \(\hat{R}U^{-1}\) in working precision, (21) yields
\[
\|\Delta S\| \lesssim c_2 u\|\hat{R}\|\|U^{-1}\| \approx c_2 u \cdot u\|A\| \cdot \frac{u^{-1}}{\|U\|} \lesssim c_2 u \quad (c_2 \approx n). \quad (27)
\]

Finally, we consider an upper bound of \(\|\Delta T\|\). Since \(\Delta T\) is a rounding error of \(L + \hat{E}\) in working precision,
\[ \|\Delta T\| \leq u(\|L\| + \|\hat{E}\|). \]

Here, using (27) we have
\[
\|\hat{E}\| \leq \|\hat{R}U^{-1}\| + \|\Delta S\| \leq \|\hat{R}\|\cdot\|U^{-1}\| + \|\Delta S\| \lesssim u\|A\| \cdot \frac{u^{-1}}{\|U\|} + c_2 u \lesssim 1. \quad (28)
\]

From this and (24),
\[ \|\Delta T\| \lesssim c_3 u \quad (c_3 \approx n). \quad (29) \]

Inserting (26), (27) and (29) into (19), we have
\[ \|\Delta C\| \lesssim (c_1 + c_2 + c_3)u \approx nu. \quad (30) \]

Moreover, from (24), (28) and (29),
\[ \|C\| \approx \|L\| + \|\hat{E}\| + \|\Delta T\| \approx n. \]

Therefore, it turns out that \(\Delta C\) is small enough to approximate \(U^{-T}AP^T\) by \(C\).
3.2.3 Step 4
This step is almost the same way as Step 2 and Section 3.1.2 except
\[ r^{(k)} = U^{-T} \left( b - A\hat{x}^{(k)} \right). \] (31)
Note that (31) should be calculated by forward substitution using accurate computation based on Dot2.

4. Numerical experiments
We apply the proposed algorithm using Methods A and B in Section 3 to some test matrices, and measure computing time and maximum relative errors of obtained approximate solutions. Computing environment is shown in Table III. Theoretical peak performance of the laptop PC is 32 GFLOPS (floating-point operations per sec \( \times 10^{-9} \)).

We apply the LAPACK routine \texttt{DTRTRI} for (8) by using MATLAB’s MEX function, which enables us to call the functions written in C on MATLAB. Moreover, we execute Dot2 using C with parallel computations by OpenMP and MEX function for (9), (11), (13), (16) and (31). We set \( \epsilon = 10^{-9} \) and \( k_{\text{max}} = 16 \) at Steps 2 and 4 of the algorithm for the iterative refinement method.

For comparison of the computational speed, we also solve \( Ax = b \) on MATLAB with Advanpix Multiprecision Computing Toolbox version 3.8.5.9059 [11], which utilizes well-known, fast, and reliable multiple-precision arithmetic libraries using GMP [12] and MPFR [10]. In particular, the toolbox is very fast in the case where the number of computational digits \( d \) is set as \( d = 34 \). Then, it is compliant with IEEE 754 binary128 (quadruple precision) arithmetic, and we adopt this setting.

The exact solution is denoted by \( x^* = A^{-1}b \). In this paper, we define the maximum relative error as
\[ \max \left| \frac{x^*_i - \hat{x}_i}{x^*_i} \right|. \]

We check whether \( \hat{C}_A \approx X_LA \) in Method A and \( \hat{C}_B \approx (L + E)^T \) in Method B are actually good approximations or not. Now, \( C_A = X_LA \) and \( C_B = U^{-T}A \) are calculated by MPFR. We confirm these relative errors \( E_A \) and \( E_B \) as follows:
\[ E_A = \frac{\|C_A - \hat{C}_A\|}{\|C_A\|}, \]
\[ E_B = \frac{\|C_B - \hat{C}_B\|}{\|C_B\|}. \]

4.1 Test matrices
We generate random matrices with specified matrix size and condition numbers. If the specified condition number is less than \( 10^{16} \), we use \texttt{randsvd} from Higham’s test matrices [8]:

\[ A := \text{gallery}('\text{randsvd}',n,cnd,3,n,n,1). \]

Here \( n \) is the order of the matrix \( A \), and \( cnd \) is an expected condition number of \( A \). If the specified condition number is greater than \( 10^{16} \), it is difficult to generate such ill-conditioned matrices using \texttt{randsvd}. Then we use Rump’s algorithm \texttt{randmat} [9] to generate ill-conditioned matrices. Since the

| CPU | Intel Core i7 2 GHz 2 Cores |
|-----|-----------------------------|
| Memory | 8GB |
| Software | MATLAB R2013b |
| Compiler | gcc version 4.4.7 |
| Theoretical peak performance | IEEE 754 binary64 (\( u = 2^{-53} \approx 10^{-16} \)) |

381
function `randmat` costs significant computing time to generate large size matrices such as $n > 1000$, we modify the way as follows.

First, we generate a small ill-conditioned matrix $A_{11} \in \mathbb{R}^{m \times m}$ for $m \ll n$ using `randmat`:

$$A_{11} := \text{randmat}(m, \text{cnd}).$$

Now, we set $m = 100$ for $n > m$. Second, we generate a random matrix $A_{22} \in \mathbb{R}^{(n-m) \times (n-m)}$ using the MATLAB function:

$$A_{22} := \text{randn}(n-m).$$

Next, we set $A' \in \mathbb{R}^{n \times n}$ with $A_{11}$ and $A_{22}$ such as

$$A' := \begin{pmatrix} A_{11} & O \\ O & A_{22} \end{pmatrix}.\$$

Finally, for a random permutation matrix $P$,

$$A := PA'P^T.$$  

Here, $P$ is chosen by using `randperm` function in MATLAB.

For right-hand side vectors, set $b$ as

$$b := A \ast \text{ones}(n,1).$$

### 4.2 Numerical results

First, we display the maximum relative error for $n = 2000$ in Table IV. The item ‘cnd’ is a specified condition number of $A$ ($\kappa(A) \approx \text{cnd}$). The item ‘$A\backslash b$’ is to solve $Ax = b$ by the standard MATLAB command in working precision. The item ‘MP’ is to solve $Ax = b$ with the Multiprecision Computing Toolbox with $d = 34$ (compliant with IEEE 754 binary128). From the results, it is confirmed that we cannot get an accurate approximate solution by $A\backslash b$ if $A$ is ill-conditioned. In case of using binary128 arithmetic (MP with $d = 34$) without the iterative refinement, we cannot obtain an accurate approximate solution for $\kappa(A) > 10^{30}$. Using the proposed algorithm (Methods A and B) with `Dot2`, we can obtain an accurate approximate solution until $\kappa(A) \approx 10^{30}$.

Next, we display computing time for $n = 2000$ in Table V. The meanings of the items in the tables are shown below.

**Table IV.** Maximum relative error (laptop PC), $n = 2000$.

| cnd | $A\backslash b$ | Method A | Method B | MP$_{d=34}$ | $\bar{E}_A$ | $\bar{E}_B$ |
|-----|----------------|----------|----------|-------------|------------|------------|
| $10^8$ | $1.7 \cdot 10^{-08}$ | $2.2 \cdot 10^{-16}$ | $2.2 \cdot 10^{-16}$ | $2.2 \cdot 10^{-16}$ | $2.3 \cdot 10^{-17}$ | $2.3 \cdot 10^{-17}$ |
| $10^{15}$ | $7.3 \cdot 10^{-02}$ | $1.3 \cdot 10^{-12}$ | $1.3 \cdot 10^{-12}$ | $2.2 \cdot 10^{-16}$ | $2.3 \cdot 10^{-17}$ | $3.5 \cdot 10^{-17}$ |
| $10^{16}$ | $8.1 \cdot 10^{-01}$ | $1.5 \cdot 10^{-15}$ | $1.8 \cdot 10^{-15}$ | $2.2 \cdot 10^{-16}$ | $2.2 \cdot 10^{-17}$ | $2.8 \cdot 10^{-16}$ |
| $10^{24}$ | $1.6 \cdot 10^{+02}$ | $2.2 \cdot 10^{-16}$ | $2.2 \cdot 10^{-16}$ | $1.1 \cdot 10^{-11}$ | $8.6 \cdot 10^{-18}$ | $5.1 \cdot 10^{-15}$ |
| $10^{30}$ | $6.9 \cdot 10^{+02}$ | $9.6 \cdot 10^{-15}$ | $3.7 \cdot 10^{-14}$ | $1.6 \cdot 10^{-06}$ | $1.0 \cdot 10^{-17}$ | $6.5 \cdot 10^{-15}$ |
| $10^{32}$ | $2.6 \cdot 10^{+01}$ | $7.1 \cdot 10^{+00}$ | $8.5 \cdot 10^{-02}$ | $1.8 \cdot 10^{-03}$ | $2.6 \cdot 10^{-17}$ | $2.0 \cdot 10^{-15}$ |

**Table V.** Computing time (sec) and ratio (laptop PC), $n = 2000$.

| cnd | LU | $T_{\text{total}}$ (s) | $k_1$ | $k_2$ | $R_{LU}$ | Method A | Method B | MP$_{d=34}$ |
|-----|----|---------------------|------|------|--------|----------|----------|------------|
| $10^8$ | 0.22 | 0.27 | 2(S) | – | 1.23 | 0.27 | 2(S) | – | 1.23 | 78.78 |
| $10^{15}$ | 0.23 | 0.35 | 6(S) | – | 1.52 | 0.35 | 6(S) | – | 1.52 | 78.86 |
| $10^{16}$ | 0.25 | 13.44 | 1(F1) | 1(S) | 53.76 | 13.30 | 1(F1) | 1(S) | 53.20 | 78.53 |
| $10^{24}$ | 0.22 | 13.06 | 1(F1) | 2(S) | 59.36 | 13.36 | 1(F1) | 2(S) | 60.73 | 73.12 |
| $10^{30}$ | 0.21 | 13.12 | 2(F1) | 4(S) | 62.48 | 13.55 | 2(F1) | 5(S) | 64.52 | 73.09 |
| $10^{32}$ | 0.22 | 13.06 | 2(F1) | 1(F1) | 59.36 | 13.30 | 2(F1) | 1(F1) | 60.45 | 73.16 |
• \( cnd \): Specified condition number of \( A \) \( (\kappa(A) \approx cnd) \)
• \( LU \): Computing time for an LU factorization of \( A^T \)
• \( T_{\text{total}} \): Total computing time for Method A or B
• \( k_1, k_2 \): The numbers of iterations at Steps 2 and 4 of the proposed algorithm, respectively
• Subscript S, F1: A type of stopping criterion for iteration in Section 2.2
• \( R_{LU} \): Ratio of total computing time to computing time for LU
• \( T_{MP} \): Total computing time to solve \( Ax = b \) with Multiprecision Computing Toolbox with \( d = 34 \) (compliant with IEEE 754 binary128)

Table V indicates that if \( \kappa(A) \leq 10^{15} \), total computing time for Method A or B is comparable to that for an LU factorization, because the algorithm successfully stops without preconditioning. On the other hand, if \( \kappa(A) > 10^{15} \), the preconditioning techniques are applied, and total computing time for Method A or B is about 60 times as much as that for an LU factorization. Although the theoretical values of \( R_{LU} \) of Methods A and B are 22 and 16, respectively, the numerical results show that the values of \( R_{LU} \) are fairly greater than the theoretical values. Moreover, although the theoretical computational cost of Method B is less than that of Method A, measured computing time for Method B is almost the same as that for Method A. These differences are due to computing performance.

Table VI shows the computing performance using the laptop PC in GFLOPS. The meanings of the items in the table are shown below.

• \( LU \): LU factorization by MATLAB
• \( \text{Dense} \times \text{Dense} \): Dense-dense matrix multiplication by MATLAB
• \( \text{Tri} \times \text{Dense} \): Triangular-dense matrix multiplication by Dot2
• \( \text{Tri} \times \text{Tri} \): Triangular-triangular matrix multiplication by Dot2

In this table, it can be seen that the performance of triangular matrix multiplications \( (\text{Tri} \times \text{Dense} \) and \( \text{Tri} \times \text{Tri} \)) using Dot2 is considerably worse than those of LU. In the proposed algorithm, Method A has a triangular-dense matrix multiplication for calculating \( X_LA \). Moreover, Method B has a triangular-triangular matrix multiplication using Dot2 for calculating \( PA^T - LU \). This is the reason why \( R_{LU} \) of Methods A and B becomes greater than the theoretical values. On the other hand, computing time for the proposed algorithm is much less than that for MP with \( d = 34 \) in all the cases. Moreover, \( C_A \) and \( C_B \) are small enough to confirm that \( C_A \) and \( C_B \) are good approximation.

In addition, we perform numerical experiments for larger problems using our workstation in the same way. This computing environment is shown in Table VII. Theoretical peak performance of the workstation is 556.8 GFLOPS. We display the maximum relative error for \( n = 5000 \) and \( n = 10000 \) in Tables IX and VIII, respectively. Computing times for \( n = 5000 \) and \( n = 10000 \) is shown in Tables X and XI. Moreover, computing performance using the workstation is presented in Table XII. These results are similar to those using the laptop PC. As these results, we can see that our proposed algorithm also works well for large size matrices.

**Table VI.** Computing performance (laptop PC, GFLOPS).

| \( n \) | \( \text{LU} \) MATLAB | \( \text{Dense} \times \text{Dense} \) MATLAB | Tri\( \times \)Dense Dot2 | Tri\( \times \)Tri Dot2 |
|------|----------------|----------------|----------------|----------------|
| 1000 | 14.80          | 22.27          | 7.90           | 5.30           |
| 2000 | 22.22          | 31.85          | 8.00           | 5.34           |
| 4000 | 27.19          | 33.92          | 8.07           | 5.37           |
We presented an accurate and efficient algorithm for solving ill-conditioned linear systems. If we use this algorithm, the condition number can be reduced by preconditioning using a result of an LU factorization. We can obtain approximate solutions of linear systems with the condition number up to
Table XII. Computing performance (workstation, GFLOPS).

| n   | 5000   | 10000  | 20000  |
|-----|--------|--------|--------|
| LU  | MATLAB | MATLB  | Dot2   | Dot2   |
| Dense×Dense | 99.4   | 133.4  | 152.5  | 58.8   |
| Tri×Dense   | 443.9  | 477.8  | 502.0  | 59.0   |
| Tri×Tri     | 89.4   | 89.1   | 87.5   | 57.5   |

$10^{30}$ by the proposed algorithm. Moreover, the proposed algorithm is much faster than the standard method using multiple precision arithmetic even if it is specialized for IEEE 754 binary128 arithmetic.

In the numerical experiments, the ratio of total computing time for the proposed algorithm to that for an LU factorization becomes considerably greater than the theoretical value. It is because triangular matrix multiplication takes much time, in particular, the performance of triangular-triangular matrix multiplication using Dot2 is very low compared to the LU factorization. Thus, we have future works to increase computational performance of accurate triangular-dense and triangular-triangular matrix multiplication, for example, using fast and accurate algorithms for matrix multiplication [13]. Moreover, if we apply the FMA instruction to Dot2, we can reduce computational cost. We wish to use FMA in our future works.

Acknowledgments

The authors would like to thank the anonymous reviewers for careful reading and insightful suggestions. This work was partially supported by CREST, JST.

References

[1] R.C. Aster, B. Borchersa, and C.H. Thurber, Parameter Estimation and Inverse Problems Second Edition, Academic Press, New York, 2012.
[2] S.M. Rump, “Inversion of extremely ill-conditioned matrices in floating-point,” Japan J. Indust. Appl. Math., vol. 26, pp. 249–277, 2009.
[3] S.M. Rump, “Accurate solution of dense linear systems, part I: Algorithms in rounding to nearest,” J. Comp. Appl. Math., vol. 242, pp. 157–184, 2013.
[4] T. Ogita, “Accurate matrix factorization: Inverse LU and inverse QR factorizations,” SIAM J. Matrix Anal. Appl., vol. 31, pp. 2477–2497, 2010.
[5] Y. Kobayashi and T. Ogita, “A fast and efficient algorithm for solving ill-conditioned linear systems,” JSIAM Letters, vol. 7, pp. 1–4, 2015.
[6] G.H. Golub and C.F. Van Loan, Matrix Computations 3rd Edition, The Johns Hopkins University Press, Baltimore and London, 1996.
[7] T. Ogita, S.M. Rump, and S. Oishi, “Accurate sum and dot product,” SIAM J. Sci. Comput., vol. 26, pp. 1955–1988, 2005.
[8] N.J. Higham, Accuracy and Stability of Numerical Algorithms 2nd edition, SIAM, Philadelphia PA, 2002.
[9] S.M. Rump, “A Class of Arbitrarily Ill-conditioned Floating-Point Matrices,” SIAM J. Matrix Anal. Appl., vol. 12, pp. 645–653, 1991.
[10] The GNU MPFR Library, http://www.mpfr.org
[11] Advanpix, Multiprecision Computing Toolbox for MATLAB, http://www.advanpix.com/
[12] The GNU Multiple Precision Arithmetic Library, https://gmplib.org
[13] K. Ozaki, T. Ogita, S. Oishi, and S.M. Rump, “Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications,” Numerical Algorithms, vol. 59, pp. 95–118, 2012.