Doctoral Dissertation

STUDIES ON ACCURATE AND EFFICIENT SOLUTIONS OF ILL-CONDITIONED LINEAR SYSTEMS BY PRECONDITIONING METHODS

前処理を利用した悪条件連立一次方程式の
高速かつ高精度な数値計算法

November 30, 2017

YUKA KOBAYASHI

Graduate School of Science
Tokyo Woman's Christian University

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. Moreover, we aim to accelerate the preconditioning method from a practical point of view. For this purpose, we apply a more efficient method of accurate matrix multiplication based on BLAS in the preconditioning method. We demonstrate excellent performance of the BLAS-based preconditioning method by numerical experiments.

Contents

Chapter 1

Introduction

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

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad b \in \mathbb{R}^n \tag{1.1}$$

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 (1.1), A is considered as ill-conditioned. If that is the case, then a numerical solution $\widehat{x}$ of (1.1) 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 (1.1) 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 are the multiplicative corrections of an 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, (1.1) can be transformed into

$$RAx = Rb. \tag{1.2}$$

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

$$\kappa(RA) \approx 1 + u\kappa(A).$$

Consequently, (1.2) becomes more well-conditioned than (1.1).

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 (1.1) can be transformed into

$$L^{-1}Ax = L^{-1}b. \tag{1.3}$$

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 (1.3). The latter requires less computational cost than the former.

This paper is organized as follows: Chapter 2 is about basics of this paper. In Chapter 3, we present an algorithm for solving ill-conditioned linear systems. Moreover, we discuss two preconditioning methods and their computational cost. Chapter 4 shows results of numerical experiments by using the proposed methods. After that, in Chapter 5, we introduce an algorithm for accurate matrix multiplication based on BLAS. We demonstrate excellent performance of the proposed method using BLAS-based method by numerical experiments. Finally, we conclude the paper in Chapter 6.

Chapter 2

Basics

2.1 Norm

2.1.1 Vector Norm

If a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ satisfies the following properties for a vector $x \in \mathbb{R}^n$, $f(x)$ is called a vector norm.

Axiom 1 (cf. [5]).

$$\begin{aligned}
&f(x) \geq 0 \quad x \in \mathbb{R}^n, \quad (f(x) = 0 \iff x = 0) \\
&f(x+y) \leq f(x) + f(y) \quad x, y \in \mathbb{R}^n \\
&f(\alpha x) = |\alpha| f(x) \quad \alpha \in \mathbb{R},\ x \in \mathbb{R}^n.
\end{aligned}$$

We denote such a function with a double bar notation: $f(x) = \|x\|$.

The three most useful norms in numerical computation are

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|,$$
$$\|x\|_2 = (\sum_{i=1}^{n} |x_i|^2)^{1/2} = (x^T x)^{1/2}, \textit{ and}$$
$$\|x\|_\infty = \max_{1 \le i \le n} |x_i|.$$

2.1.2 Matrix Norm

If a real-valued function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ satisfies the following properties for a matrix $A \in \mathbb{R}^{m \times n}$, $f(A)$ is called a matrix norm.

Axiom 2 (cf. [5]).

$$f(A) \ge 0 \quad A \in \mathbb{R}^{m \times n}, \quad (f(A) = 0 \iff A = 0)$$
$$f(A + B) \le f(A) + f(B) \quad A, B \in \mathbb{R}^{m \times n}$$
$$f(\alpha A) = |\alpha| f(A) \quad \alpha \in \mathbb{R},\ A \in \mathbb{R}^{m \times n}.$$

We denote such a function with a double bar notation: $f(A) = \|A\|$.

The most basic matrix norm is

$$\|A\| = \max_{x \ne 0} \frac{\|Ax\|}{\|x\|}.$$

Moreover, the following holds:

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|,$$
$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|, \textit{ and}$$
$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sigma_{\max}(A),$$

where the spectral radius of $B \in \mathbb{R}^{n \times n}$ is defined as

$$\rho(B) = \max\{|\lambda| : \det(B - \lambda I) = 0\},$$

and $\sigma_{\max}(A)$ denotes the largest singular value of A.

2.2 Floating-point Arithmetic

A floating-point number expresses an approximation of a real number on a computer. A set of floating-point numbers $\mathbb{F} \subset \mathbb{R}$ consists of real numbers in the form

$$y = \pm m \times \beta^{e-t}. \tag{2.1}$$

The four integer parameters in (2.1) mean

- the base β,
- the precision t,
- the exponent range $e_{\min} \le e \le e_{\max}$, and
- the mantissa m, $0 \le m \le \beta^t - 1$. .

A floating-point number $y \in \mathbb{F}$ is generally expressed such that

$$y = \pm\beta^e \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right) = \pm\beta^e \times .d_1 d_2 \ldots d_t,$$

where the digit d_i satisfies $0 \le d_i \le \beta - 1$ $(i = 1, 2, \ldots, n)$ and $d_1 \ne 0$.

For $x \in \mathbb{R}$, $fl(x)$ denotes a floating-point number nearest to x. Then, x can be approximated by a floating-point number with a relative error u. Here, u is the relative rounding error unit of floating-point arithmetic.

Theorem 1 (cf. [6]). *If $x \in \mathbb{R}$ lies in the range of $\mathbb{F}$, then*

$$fl(x) = x(1+\delta), \quad |\delta| \le u.$$

Moreover, the following version is also useful.

Theorem 2 (cf. [6]). *If $x \in \mathbb{R}$ lies in the range of $\mathbb{F}$, then*

$$fl(x) = \frac{x}{x(1+\delta)}, \quad |\delta| \le u.$$

The accuracy of the basic arithmetic operations is assumed such that, for $x, y \in \mathbb{F}$,

$$fl(x \; op \; y) = (x \; op \; y)(1+\delta), \quad |\delta| \le u, \quad op \in \{+,-,*,/\}. \tag{2.2}$$

Moreover, (2.2) also holds for the square root operation $\sqrt{x}$.

2.3 LU factorization

Let $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ be a nonsingular matrix. Under some condition, A can be expressed as the product of the lower triangular matrix L and the upper triangular matrix U as follows.

$$A = LU, \quad L = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \mathrm{O} \\ \vdots & \vdots & \ddots & \\ l_{n1} & \cdots & \cdots & l_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ \mathrm{O} & & & u_{nn} \end{pmatrix}$$

$$l_{ii} \ne 0, \quad u_{ii} \ne 0, \quad 1 \le i \le n.$$

Here, L and U are nonsingular. Such decomposition is an LU factorization. Following is a theorem on the LU factorization.

Theorem 3 (cf. [5]). *$A \in \mathbb{R}^{n \times n}$ has an LU factorization if $\det(A(1:k, 1:k)) \ne 0$ for $1 \le k \le n-1$. If A is nonsingular and the LU factorization of A exists with $l_{ii} = 1,\ 1 \le i \le n$, then the LU factorization is unique and $\det(A) = u_{11} \cdots u_{nn}$.*

2.3.1 Crout's Method

By the Crout's method, U becomes a unit upper triangular matrix. The algorithm is as follow.

Algorithm 1 The Crout's method

```
Require:
    $u_{ii} = 1 \quad (i = 1, \cdots, n)$
    $l_{ij} = 0 \quad (i < j)$
    $u_{ij} = 0 \quad (i > j)$
1: for $k = 1 : n - 1$ do
2:     for $i = k : n$ do
3:         $l_{ik} = a_{ik}$
4:     end for
5:     for $j = k + 1 : n$ do
6:         $u_{kj} = \frac{a_{kj}}{a_{kk}}$
7:     end for
8:     for $i = k + 1 : n$ do
9:         for $j = k + 1 : n$ do
10:            $a_{ij} = a_{ij} - l_{ik} u_{kj}$
11:        end for
12:    end for
13: end for
```

2.3.2 Doolittle's Method

By the Doolittle's method, L becomes a unit lower triangular matrix. The algorithm is as follow.

Algorithm 2 The Doolittle's method

```
Require:
    $l_{ii} = 1 \quad (i = 1, \cdots, n)$
    $l_{ij} = 0 \quad (i < j)$
    $u_{ij} = 0 \quad (i > j)$
 1: for $k = 1 : n-1$ do
 2:     for $j = k+1 : n$ do
 3:         $u_{kj} = a_{kj}$
 4:     end for
 5:     for $i = k : n$ do
 6:         $l_{ik} = \frac{a_{ik}}{a_{kk}}$
 7:     end for
 8:     for $i = k+1 : n$ do
 9:         for $j = k+1 : n$ do
10:             $a_{ij} = a_{ij} - l_{ik}u_{kj}$
11:         end for
12:     end for
13: end for.
```

Chapter 3

Proposed Algorithm

Let $M \in \mathbb{R}^{n\times n}$ denote a left preconditioner of A. Then (1.1) 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

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

by heuristics (cf. e.g. [5, 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 an algorithm using preconditioning techniques.

Algorithm 3 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 $Ax = b$ by forward and backward substitutions for obtaining its approximate solution.

Step 2. Apply the iterative refinement method (cf. e.g. [5, 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.

3.1 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

$$U^T L^T P x = b,$$

and obtain

$$\widehat{x} \approx P^T L^{-T} U^{-T} b.$$

3.2 Step 2

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

$$r^{(k)} \approx b - A\widehat{x}^{(k)} \tag{3.1}$$

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

$$Ay = r^{(k)} \tag{3.2}$$

by using the LU factors obtained at Step 1. Let $\widehat{y}^{(k)}$ be an approximate solution of (3.2), then we update $\widehat{x}^{(k)}$ by

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

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

(S) Let ϵ denote a tolerance. If

$$|\widehat{x}_i^{(k+1)} - \widehat{x}_i^{(k)}| \le \epsilon |\widehat{x}_i^{(k+1)}| \quad \text{for all } i, \tag{3.3}$$

then this algorithm successfully stops.

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

$$\|\widehat{y}^{(k)}\| \ge \alpha \|\widehat{y}^{(k-1)}\| \tag{3.4}$$

is satisfied, then we also go to Part 2. We recommend $\alpha = 0.1$. Even if (3.4) is satisfied with $\alpha > 0.1$, we guess $\widehat{x}^{(k)}$ cannot be improved any more due to the ill-conditionedness of A.

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

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

3.3 Preconditioning Methods (Step 3 & 4)

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

3.3.1 Method A

We calculate $X_L \approx U^{-T}$ explicitly, and use X_L as a left preconditioner of A. After that, we solve $X_L A x = X_L b$. 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.

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

$$X_L \approx U^{-T}. \tag{3.5}$$

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:

$$C \approx X_L A \quad \text{and} \quad d \approx X_L b. \tag{3.6}$$

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

$$Cx = d. \tag{3.7}$$

We expect C to be not ill-conditioned. If that is the case, (3.7) can be solved using a standard way with an LU factorization and forward and backward substitutions.

Table 3.1 shows computational cost of Method A using `Dot2`. Total computational cost of Method A is $\frac{85}{6}n^3$ flops (floating-point operations). It costs 21.25 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 $\frac{20}{3}n^3$ flops.

Table 3.1: Computational cost (flops) of Method A and ratio to LU

Operations requiring $\mathcal{O}(n^3)$ flops	With Dot2	Dot2 (FMA)
LU factorization of A^T	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
$X_L := U^{-T}$	$\frac{1}{3}n^3$	$\frac{1}{3}n^3$
$C := X_L A$	$\frac{25}{2}n^3$	$5n^3$
LU factorization of C	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
Total cost	$\frac{85}{6}n^3$	$\frac{20}{3}n^3$
Ratio to LU	21.25	10

Step 4

This step is almost the same as Step 2 except

$$r^{(k)} \approx X_L \left(b - A\widehat{x}^{(k)}\right). \tag{3.8}$$

Note that (3.8) should be calculated by forward substitution using accurate computation based on Dot2.

3.3.2 Method B

In this method, we solve

$$U^{-T}Ax = U^{-T}b \tag{3.9}$$

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.

Step 3

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

$$R := PA^T - LU. \tag{3.10}$$

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

$$PA^T = (L + E)U.$$

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

$$E = (PA^T - LU)U^{-1} = RU^{-1}. \tag{3.11}$$

In general, E is not a lower triangular matrix. Now, we calculate R in (3.10) accurately by `Dot2` and obtain its approximation $\widehat{R}$. Moreover we calculate $\widehat{E} \approx \widehat{R}U^{-1}$ in (3.11) by forward substitution in working precision. Therefore

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

and (3.9) is transformed into

$$(L + \widehat{E})^T Px = U^{-T}b. \tag{3.12}$$

We calculate

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

in working precision and

$$d \approx U^{-T}b \tag{3.13}$$

Table 3.2: Computational cost (flops) of Method B and ratio to LU

Operations requiring $\mathcal{O}(n^3)$ flops	With Dot2	Dot2 (FMA)
LU factorization of A^T	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
$\widehat{R} \approx PA^T - LU$	$\frac{25}{3}n^3$	$\frac{10}{3}n^3$
$\widehat{E} \approx RU^{-1}$	n^3	n^3
LU factorization of $C \approx (L+E)^T$	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
Total cost	$\frac{32}{3}n^3$	$\frac{17}{3}n^3$
Ratio to LU	16	8.5

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

$$CPx = d \tag{3.14}$$

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

$$\widehat{x} \approx P^TU_2^{-1}L_2^{-1}P_2d.$$

Table 3.2 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 $\frac{7}{2}n^3$ 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.

Error Bound of C

We explain why CP in (3.14) becomes a good approximation of $U^{-T}A$. We assume

$$\kappa(A) > u^{-1}. \tag{3.15}$$

Let Δ_C denote an error of C such that

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

We consider to estimate $\|\Delta_C\|$. First, we define Δ_T, Δ_S and Δ_R as rounding errors such that

$$\begin{aligned} C &= (L + \widehat{E} + \Delta_T)^T, \\ \widehat{E} &= \widehat{R}U^{-1} + \Delta_S, \\ \widehat{R} &= PA^T - LU + \Delta_R. \end{aligned}$$

Then

$$\begin{aligned} C^T &= L + (\widehat{R}U^{-1} + \Delta_S) + \Delta_T \\ &= L + \left(PA^T - LU + \Delta_R\right)U^{-1} + \Delta_S + \Delta_T \\ &= PA^TU^{-1} + \Delta_R U^{-1} + \Delta_S + \Delta_T \end{aligned}$$

and

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

Therefore,

$$\|\Delta_C\| \le \|\Delta_R U^{-1}\| + \|\Delta_S\| + \|\Delta_T\|. \tag{3.16}$$

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:

$$\begin{aligned}
\|A\| = \|A^T\| &\approx \||L||U|\|, & (3.17)\\
\|A\| &\lesssim \|U\|, & (3.18)\\
\kappa(U) &\approx \min\left\{\kappa(A), u^{-1}\right\}, & (3.19)\\
\|PA^T - LU\| &\approx u\|A\|, & (3.20)\\
\|L\| &\approx n. & (3.21)
\end{aligned}$$

From (3.15) and (3.19),

$$\kappa(U) = \|U\|\|U^{-1}\| \approx u^{-1}.$$

First, we consider an upper bound of $\|\Delta_R U^{-1}\|$. Since we use Dot2 to calculate $R := PA^T - LU$,

$$\|\Delta_R\| \approx u\|PA^T - LU\| + c_0 u^2 \|P|A^T| + |L||U|\| \quad (c_0 \approx n).$$

Here, according to [7], $c_0 \approx n^2$ in the sense of error bound. However, in fact an actual error often becomes $\mathcal{O}(n)$. Therefore $c_0 \approx n$ is more practical. Then (3.17) and (3.19) yield

$$\begin{aligned}
\|\Delta_R\| &\approx u^2\|A\| + c_0 u^2 \|A\| \\
&\approx c_1 u^2 \|A\| \quad (c_1 \approx n), & (3.22)
\end{aligned}$$

and

$$\|\Delta_R U^{-1}\| \le \|\Delta_R\| \cdot \|U^{-1}\| \approx c_1 u^2 \|A\| \cdot \frac{u^{-1}}{\|U\|} \lesssim c_1 u. \qquad (3.23)$$

Next, we consider an upper bound of $\|\Delta_S\|$. From (3.20) and (3.22),

$$\begin{aligned}
\|\widehat{R}\| &\le \|PA^T - LU\| + \|\Delta_R\| \\
&\lesssim u\|A\|.
\end{aligned}$$

Since Δ_S is a rounding error of $\widehat{R}U^{-1}$ in working precision, (3.18) yields

$$\begin{aligned} \|\Delta_S\| &\lesssim c_2 u \|\widehat{R}\| \|U^{-1}\| \approx c_2 u \cdot u\|A\| \cdot \frac{u^{-1}}{\|U\|} \\ &\lesssim c_2 u \quad (c_2 \approx n). \end{aligned} \tag{3.24}$$

Finally, we consider an upper bound of $\|\Delta_T\|$. Since Δ_T is a rounding error of $L+\widehat{E}$ in working precision,

$$\|\Delta_T\| \le u(\|L\| + \|\widehat{E}\|).$$

Here, using (3.24) we have

$$\begin{aligned} \|\widehat{E}\| &\le \|\widehat{R}U^{-1}\| + \|\Delta_S\| \le \|\widehat{R}\| \cdot \|U^{-1}\| + \|\Delta_S\| \lesssim u\|A\| \cdot \frac{u^{-1}}{\|U\|} + c_2 u \\ &\lesssim 1. \end{aligned} \tag{3.25}$$

From this and (3.21),

$$\|\Delta_T\| \lesssim c_3 u \quad (c_3 \approx n). \tag{3.26}$$

Inserting (3.23), (3.24) and (3.26) into (3.16), we have

$$\begin{aligned} \|\Delta_C\| &\lesssim (c_1 + c_2 + c_3)u \\ &\approx nu. \end{aligned} \tag{3.27}$$

Moreover, from (3.21), (3.25) and (3.26),

$$\begin{aligned} \|C\| &\approx \|L\| + \|\widehat{E}\| + \|\Delta_T\| \\ &\approx n. \end{aligned}$$

Therefore, it turns out that Δ_C is small enough to approximate $U^{-T}AP^T$ by C.

Step 4

This step is almost the same way as Step 2 and Section 3.3.1 except

$$r^{(k)} = U^{-T}\left(b - A\widehat{x}^{(k)}\right). \tag{3.28}$$

Note that (3.28) should be calculated by forward substitution using accurate computation based on `Dot2`.

Chapter 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 4.1. Theoretical peak performance of the laptop PC is 32 GFLOPS (floating-point operations per sec $\times 10^{-9}$). We apply the LAPACK routine `DTRTRI` for (3.5) 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 (3.6), (3.8), (3.10), (3.13) and (3.28). We set $\epsilon = 10^{-9}$ and $k_{\max} = 16$ 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 [8], which utilizes well-known, fast, and reliable multiple-precision arithmetic libraries using GMP [9] 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

Table 4.1: Computing Environment (laptop PC)

CPU	Intel Core i7 2 GHz 2 Cores
Memory	8GB
Software	MATLAB R2013b
Compiler	gcc version 4.4.7
	IEEE 754 binary64 ($u = 2^{-53} \approx 10^{-16}$)
Theoretical peak performance	32 GFLOPS

relative error as

$$\max_i \left| \frac{x_i^* - \widehat{x}_i}{x_i^*} \right| .$$

We check whether $\widehat{C}_{\mathrm{A}} \approx X_L A$ in Method A and $\widehat{C}_{\mathrm{B}} \approx (L+E)^T$ in Method B are actually good approximations or not. Now, $C_{\mathrm{A}} = X_L A$ and $C_{\mathrm{B}} = U^{-T} A$ are calculated by MPFR. We confirm these relative errors E_{A} and E_{B} as follows:

$$E_{\mathrm{A}} = \frac{\|C_{\mathrm{A}} - \widehat{C}_{\mathrm{A}}\|}{\|C_{\mathrm{A}}\|},$$

$$E_{\mathrm{B}} = \frac{\|C_{\mathrm{B}} - \widehat{C}_{\mathrm{B}}\|}{\|C_{\mathrm{B}}\|}.$$

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 `randsvd` from Higham's test matrices [6]:

$$A := \texttt{gallery('randsvd',n,cnd,3,n,n,1)} .$$

Here n is the order of the matrix A and cnd is an expected condition number of A. Besides, if the condition number is greater than 10^{16}, it is difficult to generate such ill-conditioned matrices using **randsvd**. Then we use Rump's algorithm **randmat** [11] to generate ill-conditioned matrices. Since the 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**:

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

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:

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

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

$$A' := \left.\begin{pmatrix} \begin{array}{c|c} A_{11} & O \\ \hline O & A_{22} \end{array} \end{pmatrix} \begin{matrix} \}m \\ \\ \end{matrix}\right\} n \ .$$

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

$$b := A * \texttt{ones(n,1)} \ .$$

Table 4.2: Maximum Relative Error (laptop PC), $n = 2000$

cnd	$A\backslash b$	Method A	Method B	$\mathrm{MP}_{d=34}$	E_{A}	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}	$2.6 \cdot 10^{+01}$	$7.1 \cdot 10^{+00}$	$8.5 \cdot 10^{-02}$	$1.8 \cdot 10^{-03}$	$2.6 \cdot 10^{-17}$	$2.0 \cdot 10^{-15}$

4.2 Numerical Results

First, we display the maximum relative error for $n = 2000$ in Table 4.2. The item 'cnd' is a specified condition number of A ($\kappa(A) \approx$ 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 4.3. The meanings of the items in the tables are shown below.

- cnd: Specified condition number of A ($\kappa(A) \approx$ cnd)
- LU: Computing time for an LU factorization of A^T in binary64 arithmetic

Table 4.3: Computing time(sec) and Ratio (laptop PC), $n = 2000$

		Method A				Method B				$\mathrm{MP}_{d=34}$
cnd	LU	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	T_{MP}
10^8	0.22	0.27	$2_{(\mathrm{S})}$	–	1.23	0.27	$2_{(\mathrm{S})}$	–	1.23	78.78
10^{15}	0.23	0.35	$6_{(\mathrm{S})}$	–	1.52	0.35	$6_{(\mathrm{S})}$	–	1.52	78.86
10^{16}	0.25	13.44	$1_{(\mathrm{F1})}$	$1_{(\mathrm{S})}$	53.76	13.30	$1_{(\mathrm{F1})}$	$1_{(\mathrm{S})}$	53.20	78.53
10^{24}	0.22	13.06	$1_{(\mathrm{F1})}$	$2_{(\mathrm{S})}$	59.36	13.36	$1_{(\mathrm{F1})}$	$2_{(\mathrm{S})}$	60.73	73.12
10^{30}	0.21	13.12	$2_{(\mathrm{F1})}$	$4_{(\mathrm{S})}$	62.48	13.55	$2_{(\mathrm{F1})}$	$5_{(\mathrm{S})}$	64.52	73.09
10^{32}	0.22	13.06	$2_{(\mathrm{F1})}$	$1_{(\mathrm{F1})}$	59.36	13.30	$2_{(\mathrm{F1})}$	$1_{(\mathrm{F1})}$	60.45	73.16

Table 4.4: Computing Performance (laptop PC, GFLOPS)

	LU	Dense×Dense	Tri×Dense	Tri×Tri
n	MATLAB	MATLAB	`Dot2`	`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

- $\mathrm{T}_{\mathrm{total}}$: 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 3.2
- 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 4.3 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 $\mathrm{R_{LU}}$ of Methods A and B are 22 and 16, respectively, the numerical results show that the values of $\mathrm{R_{LU}}$ 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 4.4 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
- Dense×Dense: Dense-dense matrix multiplication by MATLAB
- Tri×Dense: Triangular-dense matrix multiplication by `Dot2`
- Tri×Tri: Triangular-triangular matrix multiplication by `Dot2`

In this table, it can be seen that the performance of triangular matrix multiplications (Tri×Dense and Tri×Tri) using `Dot2` is considerably worse than those of LU. In the proposed algorithm, Method A has a triangular-dense matrix multiplication for calculating $X_L A$. Moreover, Method B has a triangular-triangular matrix multiplication using `Dot2` for calculating $PA^T - LU$. This is the reason why $\mathrm{R_{LU}}$ of Methods A and B becomes greater than the theoretical values. On the other hand, computing

Table 4.5: Computing Environment (workstation)

CPU	Intel Xeon E5-4617 2.9 GHz 24 Cores
Memory	1TB
Software	MATLAB R2015b
Compiler	Intel C++ Compiler version 13.1.1
	IEEE 754 binary64 ($u = 2^{-53} \approx 10^{-16}$)
Theoretical peak performance	556.8 GFLOPS

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 the workstation in the same way. This computing environment is shown in Table 4.5. Theoretical peak performance of the workstation is 556.8 GFLOPS. We display the maximum relative error for $n = 5000$ and $n = 10000$ in Tables 4.7 and 4.6, respectively. Computing times for $n = 5000$ and $n = 10000$ is shown in Tables 4.8 and 4.9. Moreover, computing performance using the workstation is presented in Table 4.10. 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 4.6: Maximum Relative Error (workstation), $n = 5000$

cnd	$A\backslash b$	Method A	Method B	$\text{MP}_{d=34}$
10^{8}	$8.6 \cdot 10^{-08}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{14}	$2.9 \cdot 10^{-02}$	$2.9 \cdot 10^{-14}$	$2.9 \cdot 10^{-14}$	$2.2 \cdot 10^{-16}$
10^{15}	$2.9 \cdot 10^{-01}$	$4.3 \cdot 10^{-16}$	$4.4 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{24}	$2.6 \cdot 10^{+01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.3 \cdot 10^{-11}$
10^{30}	$2.7 \cdot 10^{+01}$	$4.6 \cdot 10^{-14}$	$8.8 \cdot 10^{-13}$	$1.1 \cdot 10^{-05}$
10^{32}	$3.4 \cdot 10^{+01}$	$2.9 \cdot 10^{+00}$	$8.4 \cdot 10^{-02}$	$1.4 \cdot 10^{-04}$

Table 4.7: Maximum Relative Error (workstation), $n = 10000$

cnd	$A\backslash b$	Method A	Method B	$\text{MP}_{d=34}$
10^{8}	$8.6 \cdot 10^{-08}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{14}	$2.9 \cdot 10^{-02}$	$1.8 \cdot 10^{-12}$	$1.8 \cdot 10^{-12}$	$2.2 \cdot 10^{-16}$
10^{16}	$3.3 \cdot 10^{+03}$	$8.5 \cdot 10^{-16}$	$7.4 \cdot 10^{-16}$	$3.8 \cdot 10^{-15}$
10^{24}	$2.6 \cdot 10^{+01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$3.1 \cdot 10^{-11}$
10^{30}	$2.7 \cdot 10^{+01}$	$1.4 \cdot 10^{-14}$	$2.6 \cdot 10^{-14}$	$7.3 \cdot 10^{-07}$
10^{32}	$3.4 \cdot 10^{+01}$	$1.8 \cdot 10^{+03}$	$1.9 \cdot 10^{-03}$	$5.3 \cdot 10^{-04}$

Table 4.8: Computing time(sec) and Ratio (workstation), $n = 5000$

		Method A				Method B				$\mathrm{MP}_{d=34}$
cnd	LU	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	T_{MP}
10^8	1.0	1.0	$2_{(S)}$	–	1.0	1.0	$2_{(S)}$	–	1.0	160.9
10^{14}	1.0	1.2	$5_{(S)}$	–	1.2	1.2	$5_{(S)}$	–	1.2	174.5
10^{15}	1.0	22.6	$1_{(F1)}$	$1_{(S)}$	22.6	24.3	$1_{(F1)}$	$1_{(S)}$	24.3	159.0
10^{24}	1.0	22.9	$2_{(F1)}$	$2_{(S)}$	22.9	25.2	$2_{(F1)}$	$2_{(S)}$	25.2	154.4
10^{30}	1.0	23.7	$1_{(F1)}$	$5_{(S)}$	23.7	24.1	$1_{(F1)}$	$4_{(S)}$	24.1	154.1
10^{32}	1.0	23.4	$1_{(F1)}$	$2_{(F1)}$	23.4	25.1	$1_{(F1)}$	$2_{(F1)}$	25.1	154.5

Table 4.9: Computing Time (sec) and Ratio (workstation), $n = 10000$

		Method A				Method B				$\mathrm{MP}_{d=34}$
cnd	LU	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	$\mathrm{T}_{\mathrm{total}}$	k_1	k_2	R_{LU}	T_{MP}
10^8	5.3	5.6	$2_{(S)}$	–	1.1	5.6	$2_{(S)}$	–	1.1	1072.4
10^{14}	5.4	5.9	$5_{(S)}$	–	1.1	5.9	$5_{(S)}$	–	1.1	1071.3
10^{15}	5.4	193.7	$1_{(F1)}$	$1_{(S)}$	35.9	203.9	$1_{(F1)}$	$1_{(S)}$	37.8	1070.3
10^{24}	5.4	196.1	$1_{(F1)}$	$2_{(S)}$	36.3	198.5	$1_{(F1)}$	$2_{(S)}$	36.8	1055.9
10^{30}	5.5	197.6	$2_{(F1)}$	$4_{(S)}$	35.9	208.5	$2_{(F1)}$	$4_{(S)}$	37.9	1055.9
10^{32}	5.4	193.4	$2_{(F1)}$	$1_{(F1)}$	35.8	206.1	$2_{(F1)}$	$3_{(F1)}$	38.2	1055.9

Table 4.10: Computing Performance (workstation, GFLOPS)

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

Chapter 5

Acceleration of the Method A

If we use Dot2, we can calculate a dot product as if computed in quadruple precision arithmetic. This algorithm works well for the preconditioning method in terms of the accuracy of the results. However, it is not so easy to implement the algorithm in every computer environment with the optimization level as good as optimized BLAS routines, such as Intel MKL [12] and OpenBLAS [13]. With the BLAS-based method [14, 15] of accurate matrix multiplication, we can calculate matrix products much faster than Dot2 with a naive implementation. Therefore, we apply the BLAS-based method to calculate C in (3.6). It is expected to save the computing time of the Method A significantly.

5.1 BLAS-based Method

We briefly review the basic idea of the algorithms [14, 15] for accurate matrix multiplication based on Level 3 BLAS. For two floating-point matrices A and B with consistent inner dimension, let us divide A and B into three floating-point matrices each such that

$$A = A^{(1)} + A^{(2)} + A^{(3)}, \quad B = B^{(1)} + B^{(2)} + B^{(3)}.$$

These transformations are error-free. Each element of $A^{(k)}$ and $B^{(k)}$, $k = 1, 2$ has nonzero significand bits fewer than the number of significand bits in working precision (53 for binary64) such that $A^{(1)}B^{(1)}$, $A^{(1)}B^{(2)}$ and $A^{(2)}B^{(1)}$ can be computed in working precision arithmetic without roundoff error. Then, the matrix product AB is expressed as

$$AB = (A^{(1)} + A^{(2)} + A^{(3)})(B^{(1)} + B^{(2)} + B^{(3)}),$$

and

$$AB = A^{(1)}B^{(1)} + ((A^{(1)}B^{(2)} + A^{(2)}B^{(1)}) + (A^{(1)}B^{(3)} + A^{(2)}B^{(2)} + A^{(3)}B)).$$

Therefore, we can simulate a higher-precision matrix multiplication by calculating six matrix multiplications in working precision arithmetic with Level 3 BLAS routines, such as DGEMM and DTRMM. A obvious drawback of this method is to require more working space for storing intermediate results. Although this method does not necessarily achieve quadruple precision arithmetic, heuristics suggest that it is usually sufficient in practice.

Table 5.1 shows the computational cost of the preconditioning method together with the ratio to the computational cost of an LU factorization. In Table 5.1, the cost of calculating C with Dot2 is $\frac{25}{2}n^3$ flops (floating-point operations), while the cost of calculating C with the BLAS-based method is $6n^3$ flops. Because of this difference, the total cost of the preconditioning method with the BLAS-based method is less than that with Dot2.

5.2 Numerical Results

In the Method A, we apply each of Dot2 and the BLAS-based method for accurate matrix multiplication and compare numerical results on the workstation. The experiment method is the same as section 4.

Table 5.1: Computational cost (flops) of Method A and ratio to LU

Operations requiring $\mathcal{O}(n^3)$ flops	With Dot2	BLAS-based
LU factorization of A^T	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
$X_L := U^{-T}$	$\frac{1}{3}n^3$	$\frac{1}{3}n^3$
$C := X_L A$	$\frac{25}{2}n^3$	$6n^3$
LU factorization of C	$\frac{2}{3}n^3$	$\frac{2}{3}n^3$
Total cost	$\frac{85}{6}n^3$	$\frac{23}{3}n^3$
Ratio to LU	21.25	11.5

First, we show the maximum relative errors (4.1) of approximate solutions obtained by both of the preconditioning methods with Dot2 and the BLAS-based method. We also show the results of the standard method to solve $Ax = b$ with an LU factorization in multiple precision arithmetic. To estimate the maximum relative errors, we solve the linear systems in sufficiently long precision arithmetic and regard the results as the exact solutions. Numerical results are shown in Tables 5.5, 5.6 and 5.7 for $n = 5000$, $n = 10000$ and $n = 20000$, respectively. As can be seen, both of the preconditioning methods with Dot2 and the BLAS-based method work well for cnd $\leq 10^{30}$. In the case of cnd $= 10^{32}$, the preconditioning method fails because the generated matrix is too ill-conditioned and it is not sufficient for the preconditioning to reduce the condition number.

Next, we compare computing times of the above methods. The results are shown in Tables 5.5, 5.6 and 5.7 for $n = 5000$, $n = 10000$ and $n = 20000$, respectively. The

Table 5.2: Maximum Relative Error, $n = 5000$

cnd	$A\backslash b$ (MATLAB)	With Dot2	BLAS-based	MP ($d = 34$)
10^{18}	$3.7 \cdot 10^{+00}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{24}	$2.6 \cdot 10^{+01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.3 \cdot 10^{-11}$
10^{30}	$2.7 \cdot 10^{+01}$	$4.6 \cdot 10^{-14}$	$4.6 \cdot 10^{-14}$	$1.1 \cdot 10^{-05}$
10^{32}	$3.4 \cdot 10^{+01}$	failed	failed	$1.4 \cdot 10^{-04}$

Table 5.3: Maximum relative error, $n = 10000$

cnd	$A\backslash b$ (MATLAB)	With Dot2	BLAS-based	MP ($d = 34$)
10^{18}	$5.6 \cdot 10^{-01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{24}	$2.6 \cdot 10^{+01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$3.1 \cdot 10^{-11}$
10^{30}	$2.7 \cdot 10^{+01}$	$1.4 \cdot 10^{-14}$	$1.4 \cdot 10^{-14}$	$7.3 \cdot 10^{-07}$
10^{32}	$3.4 \cdot 10^{+01}$	failed	failed	$5.3 \cdot 10^{-04}$

meanings of the items in the tables are as follows.

- cnd: Specified condition number of A ($\kappa(A) \approx$ cnd)
- $\mathrm{T_{LU}}$: Computing time for an LU factorization of A^T in binary64 arithmetic
- $\mathrm{T_{total}}$: Total computing time for the Method A
- k_1: The number of iterations for iterative refinement in Part 1 of Algorithm 3
- k_2: The number of iterations for iterative refinement in Part 2 of Algorithm 3
- $\mathrm{R_{LU}}$: Ratio of $\mathrm{T_{total}}$ to $\mathrm{T_{LU}}$

Table 5.4: Maximum relative error, $n = 20000$

cnd	$A\backslash b$ (MATLAB)	With Dot2	BLAS-based	MP ($d = 34$)
10^{18}	$2.0 \cdot 10^{+00}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{24}	$5.3 \cdot 10^{+01}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$	$7.3 \cdot 10^{-12}$
10^{30}	$1.2 \cdot 10^{+02}$	$2.0 \cdot 10^{-13}$	$2.0 \cdot 10^{-13}$	$7.5 \cdot 10^{-06}$
10^{32}	$1.9 \cdot 10^{+01}$	failed	failed	$1.4 \cdot 10^{-03}$

Table 5.5: Computing time (sec.) and ratio, $n = 5000$

		With Dot2				BLAS-based				MP ($d = 34$)
cnd	$\mathrm{T_{LU}}$	$\mathrm{T_{total}}$	k_1	k_2	$\mathrm{R_{LU}}$	$\mathrm{T_{total}}$	k_1	k_2	$\mathrm{R_{LU}}$	$\mathrm{T_{MP}}$
10^{18}	1.0	20.0	1	1	20.0	6.8	1	1	6.8	128.3
10^{24}	1.0	22.9	2	2	22.9	6.8	2	2	6.8	129.6
10^{30}	1.0	23.7	1	5	23.7	6.9	1	5	6.9	129.0

- $\mathrm{T_{MP}}$: Total computing time to solve $Ax = b$ with the multiple-precision toolbox with $d = 34$ (compliant with IEEE 754 binary128)

As can be seen from the tables, the preconditioning method with the BLAS-based method is much faster than that with Dot2 in all cases. It is remarkable that the preconditioning method with the BLAS-based method requires less than 10 times as much as computing time of an LU factorization in working precision arithmetic.Therefore, it turns out that the Method A with the BLAS-based method is very effective.

Table 5.6: Computing time (sec.) and ratio, $n = 10000$

		With `Dot2`				BLAS-based				MP ($d = 34$)
cnd	T_{LU}	T_{total}	k_1	k_2	R_{LU}	T_{total}	k_1	k_2	R_{LU}	T_{MP}
10^{18}	5.4	148.3	1	2	27.5	42.5	1	1	7.9	923.0
10^{24}	5.4	148.1	1	2	27.4	42.7	1	2	7.9	927.5
10^{30}	5.4	148.9	2	4	27.6	43.6	2	4	8.1	924.9

Table 5.7: Computing time (sec.) and ratio, $n = 20000$

		With `Dot2`				BLAS-based				MP ($d = 34$)
cnd	T_{LU}	T_{total}	k_1	k_2	R_{LU}	T_{total}	k_1	k_2	R_{LU}	T_{MP}
10^{18}	36.7	1192.2	1	2	32.5	291.8	1	1	8.0	7078.4
10^{24}	36.7	1192.7	1	2	32.5	291.4	1	2	7.9	7060.8
10^{30}	36.7	1196.8	2	3	32.6	291.7	2	3	7.9	7092.1

Chapter 6

Conclusion

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 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.

If we use the BLAS-based method instead of `Dot2` to achieve accurate matrix multiplication, we can save much computing time due to high efficiency of BLAS routines, although the BLAS-based method requires more working space than the method with `Dot2`. Numerical results demonstrate that applying the BLAS-based method to the preconditioning method is greatly effective regarding the acceleration of the Method A. If we apply the FMA instruction to `Dot2`, we can reduce computational cost. We wish to use FMA in our future works.

Acknowledgements

First of all, I deeply appreciate the help received from Professor Takeshi Ogita at Tokyo Woman's Christian University, my supervisor, with valuable discussion, comments on this paper, and reading the manuscript. I would like to express my gratitude to Professor Katsuhisa Ozaki at Shibaura Institute of Technology for generous support, helpful advice and comments about the BLAS-based method. I also thank Professor Yuka Kato at Tokyo Woman's Christian University and Professor Toshinori Oaku at Tokyo Woman's Christian University for generous support during my doctoral course.

Bibliography

[1] R. C. Aster, B. Borchersa, 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., 26 (2009), 249–277.

[3] S. M. Rump, Accurate solution of dense linear systems, part I: Algorithms in rounding to nearest, J. Comp. Appl. Math., 242 (2013), 157–184.

[4] T. Ogita, Accurate matrix factorization: inverse LU and inverse QR factorizations, SIAM J. Matrix Anal. Appl., 31 (2010), 2477–2497.

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

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

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

[8] Advanpix, Multiprecision Computing Toolbox for MATLAB, http://www.advanpix.com/

[9] The GNU Multiple Precision Arithmetic Library, https://gmplib.org

[10] The GNU MPFR Library, http://www.mpfr.org

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

[12] Intel Math Kernel Library, `https://software.intel.com/en-us/intel-mkl`

[13] OpenBLAS, an optimized BLAS library, `http://www.openblas.net/`

[14] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump, Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications, Numerical Algorithms, 59 (2012), 95–118.

[15] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump, Error-free transformation of matrix multiplication with a posteriori validation, Numer. Linear Algebra Appl., 23 (2016), 931–946.

Publication Related to this Dissertation

1. Y. Kobayashi, T.Ogita, Accurate and efficient algorithm for solving ill-conditioned linear systems by preconditioning methods, Nonlinear Theory and Its Applications, IEICE, 7 (2016), 374–385.
   (The content of Chapters 1, 3, 4, and 6)

2. Y. Kobayashi, T.Ogita, K.Ozaki, Acceleration of a preconditioning method for ill-conditioned dense linear systems by use of a BLAS-based method, Reliable Computing, 25 (2017), 15–23.
   (The content of Chapter 5 and 6)