Acceleration of a Preconditioning Method for Ill-Conditioned Dense Linear Systems by Use of a BLAS-based Method *

Yuka Kobayashi
Division of Mathematics, Graduate School of Science, Tokyo Woman's Christian University, Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
yukako.109b@gmail.com

Takeshi Ogita
Division of Mathematics, Graduate School of Science, Tokyo Woman's Christian University, Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
ogita@lab.twcu.ac.jp

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

Abstract

We are interested in accurate numerical solutions of ill-conditioned linear systems using floating-point arithmetic. Recently, we proposed a preconditioning method to reduce the condition numbers of coefficient matrices. The method utilizes an LU factorization obtained in working precision arithmetic and requires matrix multiplication in quadruple precision arithmetic. In this note, 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.

Keywords: ill-conditioned linear systems, preconditioning method, floating-point arithmetic
AMS subject classifications: 65F05, 65F35, 15A12

*Submitted: December 14, 2016; Revised: May 30, 2017; Accepted: May 31, 2017.

1 Introduction

We consider an accurate numerical solution of a linear system

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, \quad b \in \mathbb{R}^n \tag{1}$$

using floating-point arithmetic, where A is a dense matrix. The relative rounding error unit of floating-point arithmetic is denoted by u. We suppose that IEEE standard 754 binary64 (double precision) is working precision and $u = 2^{-53} \approx 10^{-16}$.

Let $\kappa(A)$ denote the condition number of A, i.e.,

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

where $\|\cdot\|$ denotes the spectral norm. If $\kappa(A)$ is large as $\kappa(A) > u^{-1}$, the problem becomes ill-conditioned. Then, a numerical solution of (1) tends to be inaccurate. For example, such ill-conditioned cases arise in inverse problems (cf. e.g. [1]). In this note, we deal with linear systems such that

$$\kappa(A) \leq (u^{-1})^2 \approx 10^{32},$$

which is solvable range by standard numerical methods in quadruple precision arithmetic.

There are two standard methods to solve (1) accurately. One is to use an LU factorization in working precision arithmetic with iterative refinement in higher-precision arithmetic. If given matrices are ill-conditioned, this method cannot work well. The other is to use an LU factorization in multiple-precision arithmetic, which can work to solve ill-conditioned linear systems. However, regardless of the condition number of a given coefficient matrix, the method using multiple-precision arithmetic needs significant computing time, which is typically more than 100 times as much as computing time of an LU factorization in working precision arithmetic.

In 1984, Rump [2] showed an interesting algorithm to obtain an approximate inverse of an ill-conditioned matrix. This algorithm is based on the multiplicative corrections of an approximate inverse using accurate dot product. Moreover, Rump [3] presented an algorithm for solving ill-conditioned linear systems. Rump's idea is as follows. Let R be an approximate inverse of A. As a preconditioner for A, we multiply R to the both sides of (1) and obtain

$$RAx = Rb. \tag{2}$$

Then, $\kappa(RA)$ is reduced by a factor u such that

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

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

With a similar idea to Rump's method, Ogita [4] presented more efficient preconditioning method using an approximate inverse of an LU factor of A instead of an approximate inverse of A. Suppose $A \approx LU$ with $\kappa(A) \approx \kappa(L)$. If we use L^{-1} as a left preconditioner, (1) is transformed into

$$L^{-1}Ax = L^{-1}b.$$

Then we can reduce $\kappa(A)$ by a factor u such that

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

Recently, we developed preconditioning methods [5] based on the method in [4]. The methods can cure the defects of the standard methods for solving linear systems. In the methods, an accurate dot product algorithm (`Dot2`) [6] is used for calculating matrix multiplication. With the methods, we can get an accurate approximate solution of (1) if $\kappa(A) \lesssim \kappa(u^{-1})^2$. The methods require about 30 times as much computing time of an LU factorization in working precision arithmetic in numerical experiments.

The purpose of this study is to accelerate our previous method [5] significantly. To do this, we adopt an matrix multiplication algorithm [7, 8] based on BLAS (Basic Linear Algebra Subprograms) routines instead of `Dot2` in the proposed method. The BLAS-based preconditioning method requires less computational cost than the method [5] using `Dot2`. Moreover, the proposed method benefits from high efficiency of BLAS routines. As a result, the proposed method requires within 10 times as much as computing time of an LU factorization in working precision arithmetic, which is shown by numerical experiments.

This note is organized as follows. In Section 2, we briefly explain the preconditioning method as our previous work. After that, we introduce an algorithm for accurate matrix multiplication based on BLAS. Finally, in Section 3, we demonstrate excellent performance of the proposed method by numerical experiments.

2 Acceleration of the Preconditioning Method

2.1 Preconditioning Method

We explain the preconditioning method in [5] to reduce $\kappa(A)$, which uses an approximate inverse of an LU factor. We assume that Doolittle's LU factorization is used because it is implemented in LAPACK. Let us consider Doolittle's LU factorization of A^T with partial pivoting such that $PA^T \approx LU$, where L is a unit lower triangular matrix. Then

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

by heuristics (cf. e.g. [11, p.130]). If we adopt

$$X_L \approx U^{-T} \tag{3}$$

as a left preconditioner, then

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

After the preconditioning, we solve

$$X_L A x = X_L b \tag{4}$$

and obtain an approximate solution $\widehat{x}$ with iterative refinement method.

In practice, we compute $C \approx X_L A$ and $d \approx X_L b$ accurately and obtain a new linear system

$$Cx = d. \tag{5}$$

Then, we expect that C is not ill-conditioned and (5) can be solved accurately using an LU factorization in working precision arithmetic.

We present an algorithm for accurate solutions of linear systems using the preconditioning method.

Algorithm 1 (Kobayashi–Ogita [5]) *Accurate solutions of linear systems using the preconditioning method.*

Part 1: *Standard method to solve* $Ax = b$

Step 1. *Execute an LU factorization of* A^T *with partial pivoting by the Doolittle's method, so that* $PA^T \approx LU$. *Then, solve* $Ax = b$ *by forward and backward substitutions, i.e.,* $\widehat{x} \approx P^T(L^{-T}(U^{-T}b))$, *where* $\widehat{x}$ *is an approximate solution of* $Ax = b$.

Step 2. *Apply the iterative refinement method (cf. e.g. [11, 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: *Preconditioned method to reduce* $\kappa(A)$

Step 3. *Precondition* A *to reduce the condition number of* A *as* $C := U^{-T}A$, *where* U *is the LU factor obtained at Step 1. Then, solve* $Cx = d$ *where* $d := U^{-T}b$ *and obtain its approximate solution.*

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

In our previous work, we adopt an accurate dot product algorithm `Dot2` for calculating C. If we use `Dot2`, we can calculate a dot product as if computed in quadruple precision arithmetic. It is shown in [5] that 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 [9] and OpenBLAS [10]. With the BLAS-based method [7, 8] 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. It is expected to save the computing time of preconditioning method significantly.

2.2 BLAS-based Method

We briefly review the basic idea of the algorithms [7, 8] 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 fewer nonzero significand bits fewer than the number of significand bits in working precision (53 for binary64) so 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,

Table 1: Computational cost (flops) of the preconditioning method 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

Table 2: Computing Environment

CPU	Intel Xeon E5-4617 2.9GHz (6 cores × 4 CPUs)
Memory	1TB
Software	MATLAB R2015b
MEX Compiler	Intel C++ Compiler version 13.1.1
Working precision	IEEE 754 binary64 ($u = 2^{-53} \approx 10^{-16}$)

such as `DGEMM` and `DTRMM`. A obvious drawback of this method is that it requires 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 1 shows the computational cost of the preconditioning method together with the ratio to the computational cost of an LU factorization. In Table 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`.

3 Numerical Experiments

We apply the preconditioning method to some test matrices. Computing environment is shown in Table 2. We measure computing time and maximum relative errors of obtained approximate solutions. Let the exact solution be denoted by $x^* = A^{-1}b$. We define the maximum relative error as

$$\max_i \left| \frac{x_i^* - \widehat{x}_i}{x_i^*} \right|. \tag{6}$$

In the preconditioning method, we apply each of `Dot2` and the BLAS-based method for accurate matrix multiplication and compare numerical results. We set $\epsilon = 10^{-9}$

Table 3: 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	$1.4 \cdot 10^{-04}$

for iterative refinement of approximate solutions $\widehat{x}^{(k)}$ so that

$$|\widehat{x}_i^{(k+1)} - \widehat{x}_i^{(k)}| \leq \epsilon |\widehat{x}_i^{(k+1)}| \quad \text{for all } i.$$

To calculate (3), we apply the LAPACK routine `DTRTRI` using MEX function of MATLAB for better performance. The MEX function can call the functions written in C on MATLAB. Moreover, we implement matrix multiplication with `Dot2` in C with parallel computations by OpenMP and use MEX function.

For comparison, we also solve (1) with Advanpix Multiprecision Computing Toolbox version 3.8.5.9059 [13]. The toolbox uses GMP (GNU Multiple Prescision Arithmetic Library) [14] and the MPFR (Multiple Precision Floating-Point Reliable Library) [15] as fast and reliable multiple-precision arithmetic libraries. In particular, this toolbox becomes very fast if the number of computational digits d is set as $d = 34$, which is compliant with IEEE 754 binary128 (quadruple precision) arithmetic. We adopt this setting. We do not apply iterative refinement in this case.

As test matrices, we generate a variant of Rump's matrix `randmat` [16], which is explained in [5] in detail. Using `randmat`, we can generate a random matrix with specified matrix size and the condition number. A right-hand side vector b is set as $b := Ae$ where $e = (1, 1, \ldots, 1)^T$.

First, we show the maximum relative errors (6) 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 6, 7 and 8 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 6, 7 and 8 for $n = 5000$, $n = 10000$ and $n = 20000$, respectively. The 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 preconditioning method
- k_1: The number of iterations for iterative refinement in Part 1 of Algorithm 1

Table 4: 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	$5.3 \cdot 10^{-04}$

Table 5: 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	$1.4 \cdot 10^{-03}$

- k_2: The number of iterations for iterative refinement in Part 2 of Algorithm 1
- $\mathrm{R_{LU}}$: Ratio of $\mathrm{T_{total}}$ to $\mathrm{T_{LU}}$
- $\mathrm{T_{MP}}$: 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, while the multiple-precision approach requires more than 100 times as much as that. Therefore, it turns out that the preconditioning method, especially with the BLAS-based method, is very effective.

Table 6: 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

In conclusion, 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

Table 7: 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 8: 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

method to the preconditioning method is greatly effective regarding the acceleration of the preconditioning method.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers 16H03917, 16K12432, and CREST, JST.

References

[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] 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.

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

[7] 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.

[8] 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.

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

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

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

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

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

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

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

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