A fast and efficient algorithm for solving ill-conditioned linear systems

Yuka Kobayashi and Takeshi Ogita

1 Division of Mathematics, Graduate School of Science, Tokyo Woman’s Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
2 CREST, JST
E-mail twcu.d13m201@gmail.com

Received October 21, 2014, Accepted November 12, 2014

Abstract
In this paper, a fast and accurate algorithm for solving ill-conditioned linear systems is proposed. The proposed algorithm is based on a preconditioned technique using a result of an LU factorization, which requires less computational cost than a previous method using an approximate inverse. The algorithm can provide accurate numerical solutions for ill-conditioned problems beyond the limit of the working precision. Results of numerical experiments are presented for confirming the effectiveness of the proposed algorithm.

Keywords accurate algorithm, solutions of linear systems, preconditioned technique, floating-point arithmetic

Research Activity Group Quality of Computations

1. Introduction
This paper is concerned with a numerical solution of a 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 \( u \), in particular \( u = 2^{-53} \approx 10^{-16} \) for IEEE standard 754 binary64.

Let \( \kappa(A) \) denote the condition number of \( A \):

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

where \( \| \cdot \| \) stands for the spectral norm. If \( \kappa(A) \) is too large to solve (1), then \( A \) is ill-conditioned and a numerical solution \( x \) of (1) tends to be inaccurate. For example, such ill-conditioned (or ill-posed) problems often arise in inverse problems (cf. e.g. [1]). We propose a fast and accurate algorithm for solving (1), which can deal with ill-conditioned problems such as \( \kappa(A) \approx u^{-1} \).

The standard method of solving (1) accurately is to use an LU factorization with the iterative refinement method. The method is effective if \( \kappa(A) \) is not too large. However, the method cannot work well if \( A \) is ill-conditioned. Some multiple-precision arithmetic can cure this problem, though it requires significant computing time in spite of the magnitude of \( \kappa(A) \). To remedy the defects of the standard method, we adopt a preconditioned technique using the results of the LU factorization of \( A \).

In about 1984, Rump [2,3] derived a very interesting algorithm for accurately inverting such ill-conditioned matrices. The algorithm is based on the multiplicative corrections of an approximate inverse using accurate dot product. Moreover, Rump [4] also presented an algorithm for solving ill-conditioned linear systems by using his algorithm for accurate matrix inversion. Let \( R \) denote an approximate inverse of \( A \). By using \( R \) as a preconditioner, (1) can be transformed into

\[ RAX = Rb. \]  

Then it is observed [3] that

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

Therefore, (2) is expected to be more well-conditioned than (1).

On the other hand, Ogita [5] has shown that an approximate inverse of an LU factor of \( A \) can be used in a similar way of Rump’s method in [2,3]. Assume \( A \approx LU \) with \( \kappa(A) \approx \kappa(L) \). Let \( X_L \) denote an approximate inverse of \( L \). By using \( X_L \) as a preconditioner, the linear system (1) can be transformed into

\[ X_LAX = X_Lb. \]

Then it is also observed [5] that

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

For the purpose of this paper, we adopt the method in [5]. Then we can reduce the computational cost for the preconditioning of \( A \) compared with Rump’s method in [4] since calculating the approximate inverse of \( A \) becomes unnecessary.

The rest of the paper is organized as follows: Section 2 explains a preconditioned technique using the LU factorization. In Section 3, the proposed algorithm is presented. In Section 4, results of numerical experiments are shown for illustrating the performance of the proposed algorithm.
2. Preconditioned technique

Let \( M \in \mathbb{R}^{n \times n} \) denote some left preconditioner of \( A \) to reduce the condition number of \( A \). Then (1) can be transformed into

\[
MAx = Mb,
\]

where it is expected that

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

We can choose any 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 \( A \) such that \( A \approx LU \). By Crout’s method, \( U \) becomes a unit upper triangular matrix. Then heuristics (cf. e.g. [6, p.130]) tell us that

\[
\kappa(A) \approx \kappa(U)
\]

and \( L^{-1} \) can work as a left preconditioner such that

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

Namely, the condition number of \( A \) can be reduced by a factor around \( u \) down to 1 using \( L \).

On the other hand, LAPACK [7] adopts Doolittle’s method for an LU factorization. By Doolittle’s method, \( L \) becomes a unit lower triangular matrix. Then heuristics also tell us that

\[
\kappa(A) \approx \kappa(U)
\]

and \( U^{-1} \) can work as a right preconditioner such that

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

Therefore, the condition number of \( A \) can be reduced by a factor around \( u \) down to 1 using \( U \). LAPACK routines are well-optimized for today’s computers and therefore very fast. To utilize such LAPACK routines, we execute an LU factorization of \( AT \) for obtaining a left preconditioner for \( A \). Namely

\[
AT \approx LU \iff A \approx U^T L^T
\]

and

\[
\kappa(A) = \kappa(AT) \approx \kappa(U).
\]

By using \( X_L \approx U^{-T} \) as a left preconditioner of \( A \), we have

\[
X_LAx = X_Lb.
\]

Then it makes possible to calculate an accurate numerical solution of \( Ax = b \) while decreasing the condition number of \( A \).

3. Proposed algorithm

The proposed algorithm is as follows.

**Algorithm 1** The proposed algorithm for accurate solutions of linear systems.

1. Execute an LU factorization of \( A \) (Crout’s method for \( A \) or Doolittle’s method for \( AT \)) with partial pivoting. Then solve \( Ax = b \) by forward and back substitutions for obtaining its approximate solution.

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 Step 3.

3. Construct a left preconditioner \( X_L \) from an LU factor of \( A \) or \( AT \), and precondition \( A \) by \( X_L \) to reduce the condition number of \( A \). Then solve (3) by an LU factorization.

4. Apply the iterative refinement method to the approximate solution at Step 3.

In order to avoid extra computational cost, we intend to execute Steps 3 and 4 only if \( A \) is ill-conditioned.

In the following, we explain the details of the algorithm. Assume that Doolittle’s method for \( AT \) is used. We can also apply a similar way when we use Crout’s method for \( A \).

3.1 Step 1

To solve the linear system \( Ax = b \), we execute an LU factorization (Doolittle’s method) of \( AT \) with partial pivoting:

\[
PA^T \approx LU.
\]

After that, we solve

\[
U^T L^T Px = b,
\]

and obtain

\[
\tilde{x} \approx P^T L^{-T} U^{-T} b.
\]

3.2 Step 2

By repeating the following steps, an accurate solution of \( Ax = b \) can be obtained if \( A \) is not ill-conditioned. First, we calculate the residual (4) precisely by the algorithm **Dot2** in [8]:

\[
r_k \approx b - A\tilde{x}_k.
\]

Next, we solve

\[
Ay = r_k
\]

by using the LU factors obtained at Step 1. Let \( \tilde{y}_k \) be an approximate solution of (5), then we update \( \tilde{x}_k \) by

\[
\tilde{x}_{k+1} = \tilde{x}_k + \tilde{y}_k.
\]

The more these steps repeat, the more accuracy of \( \tilde{x} \) gains. The iterations can stop for the following two reasons:

- Let \( \epsilon \) denote a tolerance. If

\[
\frac{||\tilde{x}_{k+1} - \tilde{x}_k||}{||\tilde{x}_{k+1}||} \leq \epsilon
\]

is satisfied, then this algorithm successfully stops.

- Let \( k_{\text{max}} \) be the maximum number of iterations. If the number of iterations reaches \( k_{\text{max}} \), then we go to Step 3.

- If

\[
||\tilde{y}_k|| \geq \alpha||\tilde{y}_{k-1}||, \quad 0 < \alpha < 1
\]
is satisfied, then we also go to Step 3. We recommend that $\alpha = 0.1$. If (7) is satisfied with $\alpha = 0.1$, we guess $\tilde{x}_k$ cannot be improved any more due to the ill-conditionedness of $A$.

3.3 Step 3

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

$$X_L \approx U^{-T}. \tag{8}$$

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

$$C \approx X_LA \quad \text{and} \quad d \approx X_Lb. \tag{9}$$

Here, $X_LA$ and $X_Lb$ should be calculated accurately. To do this, we adopt the algorithm in [9] to calculate $X_LA$ and the algorithm Dot2 in [8] to calculate $X_Lb$, respectively. When using the algorithm in [9], we split both $X_L$ and $A$ into two parts, and

$$X_LA = ((X_L)_1 + (X_L)_2)(A_1 + A_2) = (X_L)_1A_1 + (X_L)_1A_2 + (X_L)_2A. \tag{10}$$

which involves three triangular matrix-matrix multiplications. The results using this method are normally more accurate than those by pure floating-point arithmetic in the working precision, though it depends on the matrices [9]. See [9] for details.

Remark 2 An alternative to the algorithm in [9] is to apply Dot2 for calculating $X_LA$. In this case, it is ensured that a result of $X_LA$ is obtained as if computed in twice the working precision. However, it requires more computational cost, which becomes $(25/2)n^3$ flops without FMA (Fused Multiply-Add) or $5n^3$ flops with FMA. Moreover, it is preferable to optimize Dot2 for matrix multiplication in terms of computational speed.

Here we now consider

$$CX = d. \tag{11}$$

We expect $C$ is not ill-conditioned. If that is the case, (11) can be solved using a standard way with an LU factorization and forward and back substitutions.

3.4 Step 4

This Step is almost the same as Step 2 except

$$r_k = X_L(b - A\tilde{x}_k). \tag{12}$$

3.5 Computational cost of the proposed algorithm

Main parts of taking $O(n^3)$ flops to calculate are as follows:

- LU factorization of $A$ (or $A^T$) and $C$ involve $(4/3)n^3$ flops.
- To calculate an approximate inverse $X_L$ of a lower triangular matrix $L$ (or $U^T$) involves $(1/3)n^3$ flops.
- To calculate lower triangular matrix-matrix multiplication $X_LA$ as in (10) involves $3n^3$ flops.

In total, the computational cost of the proposed algorithm comes to $(14/3)n^3$ flops, which means the algorithm costs 7 times as much as an LU factorization in case that $A$ is ill-conditioned. On the other hand, applying a similar way to Rump’s method in [4] costs 13 times as much as an LU factorization.

4. Numerical experiments

We conducted numerical experiments using our proposed algorithm. Computing environment is shown in Table 1.

We used the BLAS routine DTRMM for (10) and the LAPACK routine DTRTRI for (8) using the MATLAB’s MEX function, which enables us to call functions written in C on MATLAB. Moreover, we implemented Dot2 in [8] using C with parallel computations by OpenMP and MEX for (4) and (12).

We generated random matrices and right-hand side vectors with specified matrix sizes and condition numbers using randsvd from Higham’s test matrices [10]:

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

$$b = A*\text{ones}(n,1);$$

Here $n$ is an order of the matrix $A$ and cnd is an expected condition number of $A$. Moreover, we set $\epsilon = 10^{-12}$ for the iterative refinement method at Steps 2 and 4 and $k_{\text{max}} = 16$.

Results of the experiments are displayed in Tables 2, 3 and 4 for $n = 10000, 20000, 40000$, respectively. The meanings of the items in the tables are shown below.

- cond: Condition number of $A$.
- LU: Computing time(sec) for an LU factorization of $A^T$.
- Alg. 1: Computing time(sec) for the proposed algorithm.
- $k_1 & k_2$: The number of iterations at Steps 2 and 4.
- P/LU: Ratio of computing time for LU to Alg. 1.

From Tables 2, 3 and 4, it can be seen that computing time for the proposed algorithm is less than 7 times that for an LU factorization. If $n$ increases, P/LU approaches 7. This is almost identical to the theoretical value. Moreover, it is possible to solve the linear systems in the same time as the standard method using an LU factorization with the iterative refinement method if $A$ is not ill-conditioned.

Moreover, the maximum relative error of the approximate solution $\tilde{x}$ is displayed in Tables 5, 6 and 7. The maximum relative error of $\tilde{x}$ is defined as

$$\max_{i} \frac{|x_i - \tilde{x}_i|}{x_i},$$

Table 1. Computing environment.

| CPU             | Intel Xeon E5-2617 2.9GHz (24 cores) |
|-----------------|--------------------------------------|
| Memory          | 512GB                                 |
| Software        | MATLAB R2014a                         |
| IEEE 754 binary64 | ($u = 2^{-53} \approx 10^{-16}$)       |

- cond: Condition number of $A$.
- LU: Computing time(sec) for an LU factorization of $A^T$.
- Alg. 1: Computing time(sec) for the proposed algorithm.
- $k_1 & k_2$: The number of iterations at Steps 2 and 4.
- P/LU: Ratio of computing time for LU to Alg. 1.
where \( x^* = A^{-1} b \). In Tables 5, 6 and 7, the items \( A\backslash b \) and ‘Proposed’ show the maximum relative error of \( \bar{x} \) obtained by the MATLAB’s command and the proposed algorithm, respectively. Note that an LU factorization of \( A \) with partial pivoting is performed in \( A\backslash b \) without the iterative refinement.

If we use MATLAB’s command \( A\backslash b \), the accuracy of \( \bar{x} \) is getting worse depending on \( \kappa(A) \), in particular it becomes meaningless when \( A \) is ill-conditioned. On the other hand, the proposed algorithm can provide an accurate result even if \( A \) is ill-conditioned.

When \( \kappa(A) \leq 10^{13} \), the approximate solutions satisfying (6) were obtained by Steps 1 and 2 in the proposed algorithm. On the other hand, when \( \kappa(A) \geq 10^{14} \), the iterative refinement method stops by (7). After that, the approximate solutions satisfying (6) were obtained by Steps 3 and 4.

### References

[1] R. C. Aster, B. Borchers and C. H. Thurber, Parameter Estimation and Inverse Problems, 2nd ed., Academic Press, New York, 2012.

[2] S. M. Rump, Approximate inverses of almost singular matrices still contain useful information, Forschungschwerpunktes Informations- und Kommunikationstechnik, Technical Report 90.1, Hamburg University of Technology, Hamburg, Germany, 1990.

[3] S. M. Rump, Inversion of extremely ill-conditioned matrices in floating-point, Japan J. Indust. Appl. Math., 26 (2009), 249–277.

[4] S. M. Rump, Accurate solution of dense linear systems, part I:

Algorithms in rounding to nearest, J. Comput. Appl. Math., 242 (2013), 157–184.

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

[6] G. H. Golub and C. F. van Loan, Matrix Computations, 3rd ed., The Johns Hopkins University Press, Baltimore, 1996.

[7] E. Anderson et al., LAPACK User’s Guide, 3rd ed., SIAM, Philadelphia, 1999.

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

[9] K. Ozaki, T. Ogita and S. Oishi, Tight and efficient enclosure of matrix multiplication by using optimized BLAS, Numer. Linear Algebra Appl., 18 (2011), 237–248.

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