Verified numerical computations for dense linear systems in supercomputing

Katsuhisa Ozaki

1 Associate Professor, Department of Mathematical Sciences, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan  
E-mail: ozaki@sic.shibaura-it.ac.jp

Abstract. The numerical computations with guaranteed accuracy called verified numerical computations are studied, herein. We focus on an accurate numerical solution for dense linear systems of the form \( Ax = b \) and error bound of \( \| \hat{x} - x \|_\infty \) in supercomputing, where \( \hat{x} \) is an approximate solution. First, we implemented accurate algorithms for a matrix-vector product for parallel and distributed computing, and then, applied them to iterative refinements. Next, we implemented verified numerical computations using the Parallel Basic Linear Algebra Subprograms (PBLAS) and the Scalable Linear Algebra Package library (ScaLAPACK). Numerical examples of the verified numerical computations for linear systems are presented using the Fujitsu FX100 supercomputer. They clearly illustrate the efficiency of the verified numerical computations.

1. Introduction
Floating-point arithmetic can be performed fast on modern computers, however, the number of bits in floating-point numbers is finite, which means that a rounding error may occur with each arithmetic operation. Therefore, the computed results may be inaccurate due to the accumulation of rounding errors. If the size of the calculation is large, the rounding error problem becomes critical. In worst case scenarios, the computed results don’t contain any correct digits, or its sign is opposite to that of the exact result.

Many scientific problems are boiling down to linear systems. Hence, it is important to obtain accurate numerical solutions for such linear systems. We define a linear system as \( Ax = b \), where \( A \) is a coefficient matrix, and \( b \) is a right-hand side vector. Let us assume that \( A \) is a non-singular matrix, and let \( \hat{x} \) be a computed result, i.e., a result obtained by employing a numerical method using floating-point arithmetic. The accuracy of the numerical solution \( \hat{x} \) may be checked by evaluating a norm of the residual \( \| b - A\hat{x} \| \). However, a small residual does not always indicate that the approximate solution is sufficiently accurate, which is demonstrated by the well-known inequality:

\[
\| \hat{x} - A^{-1}b \| = \| A^{-1}A\hat{x} - A^{-1}b \| = \| A^{-1}(A\hat{x} - b) \| \leq \| A^{-1} \| \cdot \| b - A\hat{x} \|.
\]

This inequality indicates that even if the residual \( \| b - A\hat{x} \| \) is small, the error in the result can be large, depending on the magnitude of \( \| A^{-1} \| \).

Verified numerical computations (e.g. [1]) produce numerical solutions for linear systems and their error bounds using only floating-point arithmetic. The goal is to produce an accurate
numerical solution for linear systems with a sharp error bound. In this study, we implemented the verified numerical computations [2] for linear systems for parallel and distributed computing and reported numerical examples using the Fujitsu FX100 supercomputer.

2. Accurate Numerical Solutions

In order to obtain an accurate solution for linear systems, iterative refinements are applied. Let $F$ be a set of floating-point numbers. Notation $fl(\cdot)$ indicates that each operation in the parenthesis is performed by using floating-point arithmetic. The rounding mode of $fl(\cdot)$, $fl_\wedge(\cdot)$ and $fl_\Delta(\cdot)$ is roundTiesToEven, roundTowardNegative and roundTowardPositive, respectively, as defined in IEEE 754. Let $u$ be the unit roundoff, e.g., $u = 2^{-53}$ for binary64, as defined in IEEE 754.

The iterative refinement is performed as follows:

Step 1. Solve the given linear system $Ax = b$ and obtain its approximate solution $\hat{x} \in F$.

Step 2. Compute $b - A\hat{x} \approx r \in F$.

Step 3. Solve the linear system $Ay = r$ and obtain its approximate solution $\hat{y} \in F$.

Step 4. Update $\hat{x}$ by $fl(\hat{x} + \hat{y})$.

Steps 2,3 and 4 are repeated until the accuracy of $\hat{x}$ becomes sufficient. Heavy cancellation can occur in the computation of the residual in Step 2, therefore, we need a routine for accurate matrix-vector product. We developed accurate algorithms for the matrix-vector product in parallel and distributed computing based on accurate algorithms for dot product in [4]. For $\alpha, \beta, z \in F$, $x, y \in F^n$, let res be the computed result of $\alpha x^T y + \beta z$ obtained by our implementation. $M$ is the number of nodes in a parallel and distributed computing system and $c = n/M$ is the size of the vectors $x$ and $y$ in a node. For simplicity, we assume that $n$ is divisible by $M$. Then, if $\gamma_n = \frac{nu}{1-nu}$ for $nu < 1$, the following expression is satisfied:

$$|\text{res} - (\alpha x^T y + \beta z)| \leq u|\alpha x^T y + \beta z| + P(c, M)(|\alpha||x^T||y| + |\beta||z|),$$

$$P(c, M) := \gamma_{c+2} + (1 + \gamma_c + \gamma_{M-1})\gamma_M \gamma_c + \gamma_M^2.$$

3. Verified Numerical Computations

In this section we introduce verified numerical computations. For a linear system $Ax = b$ and the approximate solution $\hat{x}$, if there exists a matrix $R \in \mathbb{R}^{n \times n}$ such that $\|RA - I\| < 1$, where $I$ is the identity matrix, the following inequality is satisfied:

$$\|\hat{x} - A^{-1}b\|_\infty \leq \frac{\|R(A\hat{x} - b)\|_\infty}{1 - \|RA - I\|_\infty} \quad (1)$$

Several verification methods based on (1) can be implemented. We implemented the method reported in [2] by using the Parallel Basic Linear Algebra Subprograms (PBLAS), the Scalable Linear Algebra Package library (ScaLAPACK), and the accurate routine for the matrix-vector product.

In (1), we compute an upper bound of $\|RA - I\|_\infty$ using only floating-point arithmetic. Let $\alpha \in F$ be an upper bound of $\|RA - I\|_\infty$. A well-known method to obtain $\alpha$, written in the MATLAB language, is as follows:

```matlab
feature('setround',-Inf);
```

1 Thanks to Mr Ryota Ochiai, a former master’s student of the Shibaura Institute of Technology, for the implementation of the accurate matrix-vector product and its rounding error analysis [5].
\[ C = R \ast A - I; \]
\[
\text{feature('setround',Inf);} \]
\[ D = R \ast A - I; \]
\[ T = \max(|C|, |D|); \]
\[ \alpha = \text{norm}(T,\text{Inf}); \]

This approach is used in the verification method [2]. If \( \alpha < 1 \) is satisfied, the matrix \( A \) is non-singular.

On some supercomputers made by Fujitsu we can switch the rounding mode of floating-point arithmetic using the function \text{flibomp_fesetround}. We can select any one of roundTiesToEven, roundTowardPositive, roundTowardNegative, and roundTowardZero. Once the function is called, the rounding mode in all nodes is changed. Even if Open Multi Processing is used and multi-threading is employed in a node, changing the rounding mode remains valid for all threads. The pseudocode for obtaining the enclosure \([C, D]\) of \( RA - I \) written using PBLAS is as follows:

```c
//C and D are the identity matrix
flib_omp_fesetround_(&num_down); //num_down=3
pdgemm_(&NN, &NN, &n, &n, &n, &done, R, &one, &one, descMat,
        A, &one, &one, descMat, &dmone, C, &one, &one, descMat);  
flib_omp_fesetround_(&num_up); //num_up=2
pdgemm_(&NN, &NN, &n, &n, &n, &done, R, &one, &one, descMat,
        A, &one, &one, descMat, &dmone, D, &one, &one, descMat);
```

The function \text{flib_omp_fesetround} works as well as \text{feature('setround',-Inf);} for MATLAB. The function can be downloaded from [6].

In (1), we first require the enclosure of \( A\hat{x} - b \). We obtain vectors \( \text{mid}(1), \text{rad}(1) \in \mathbb{F}^n \) such that

\[ \text{mid}(1) - \text{rad}(1) \leq A\hat{x} - b \leq \text{mid}(1) + \text{rad}(1). \]

Here, all elements in the vector \( \text{rad}(1) \) are non-negative. We extended and implemented the algorithm Dot2Err in [4] for the matrix-vector product in parallel and distributed computing. Then, we expect that

\[ u \cdot |\text{mid}(1)| \approx \text{rad}(1). \]

We compute the upper bound of \( |R(A\hat{x} - b)| \) as follows

\[ |R(A\hat{x} - b)| \leq |\text{mid}(2)| + |\text{rad}(2)| + |\text{rad}(3)| \]

where

\[ |\text{mid}(2)| - |\text{rad}(2)| \leq R \cdot |\text{mid}(1)| \leq |\text{mid}(2)| + |\text{rad}(2)|, \quad |R| \cdot |\text{rad}(2)| \leq |\text{rad}(3)|. \]

The vectors \( \text{mid}(2) \) and \( \text{rad}(2) \) are obtained using the accurate matrix-vector product. The vector \( \text{rad}(3) \) can be obtained using \( f\Delta(|R| \cdot |\text{rad}(2)|) \). We obtain the upper bound of \( \|R(A\hat{x} - b)\|_{\infty} \) as

\[ \beta := f\Delta(\| \text{mid}(2) \| + |\text{rad}(2)| + |\text{rad}(3)|_{\infty}) \]

Finally, we obtain

\[ \|\hat{x} - A^{-1}b\|_{\infty} \leq \frac{\|R(A\hat{x} - b)\|_{\infty}}{1 - \|RA - I\|_{\infty}} \leq f\Delta(\frac{\beta}{f\Delta(1 - \alpha)}). \quad (2) \]
Table 1. Specification of a node in Fujitsu FX100.

| CPU      | Peak performance | Cores | Memory |
|----------|------------------|-------|--------|
| SPARC64 XIfx | more than 1 TFLOPS | 32    | 32GB   |

Table 2. Error bounds of the approximate solution.

| Dimension | Error Bound |
|-----------|-------------|
| 100 × 10^3 | 1.11e-16   |
| 200 × 10^3 | 1.14e-16   |
| 300 × 10^3 | 1.11e-16   |
| 400 × 10^3 | 1.12e-16   |
| 500 × 10^3 | 1.12e-16   |

Table 3. Comparison of computing times.

| Dimension (×10^3) | Nodes | Approximation | Verification | Ratio |
|-------------------|-------|---------------|--------------|-------|
| 100               | 25    | 9.52e+01      | 4.97e+02     | 5.22  |
| 200               | 100   | 1.94e+02      | 9.70e+02     | 4.99  |
| 300               | 225   | 3.15e+02      | 1.46e+03     | 4.64  |
| 400               | 400   | 4.49e+02      | 1.96e+03     | 4.37  |
| 500               | 625   | 6.70e+02      | 2.50e+03     | 3.73  |

4. Numerical Results

Herein, we show the numerical results for the method introduced in the previous section. We used the Fujitsu FX100 supercomputer at Nagoya University. The specifications of the node in the Fujitsu FX100 are listed in Table 1.

All elements of the matrix $A$ are generated using pseudo-random numbers. For the vector $e = (1,1,\ldots,1)^T$, the right-hand side vector is obtained as $b = f_1(A*e)$. Note that we cannot expect $x = (1,1,\ldots,1)^T$ because of the rounding errors in $f_1(A*e)$, but we can expect $x \approx (1,1,\ldots,1)^T$. We obtained the approximation $\hat{x}$ by using the iterative refinements thrice. Table 2 shows the error bound in (2). Since $x \approx (1,1,\ldots,1)^T$ and the error bound is nearly $10^{-16}$, it is guaranteed that the accuracy of $\hat{x}$ is sufficient because of the relative error being around $10^{-16}$.

Note that the costs of pdgesv and the verification methods are $\frac{2}{3}n^3 + O(n^2)$ and $6n^3 + O(n^2)$ floating-point operations, respectively. This indicates that theoretical ratio of their cost is approximately equal to nine. However, the ratio in Table 3 is much less than nine. The reason for this discrepancy is the difference of the performance between routines in PBLAS and ScaLAPACK. Table 3 shows the computing times necessary to solve the linear systems (by using the pdgesv function), obtaining the matrix inverse (by using the pdgesv function for multiple right-hand side), and computing matrix multiplication (by using the pdgemm function). The flops values indicate the number of floating-point operations. Compared to the pdgesv function, the cost of pdgemm is high in terms of the number of floating-point operations. However, the computing time of pdgemm is shorter than that of pdgesv, which explains why the computing time ratio is much better than the theoretical ratio.
Table 4. Computing times for each function.

| Dimension ($10^3$) | pdgesv $2/3n^3 + O(n^2)$ | inv $2n^3 + O(n^2)$ | pdgemm $2n^3 + O(n^2)$ |
|---------------------|--------------------------|---------------------|------------------------|
| 100                 | 9.52e+01 (1)             | 3.04e+02 (3.19)     | 9.37e+01 (0.98)        |
| 200                 | 1.94e+02 (1)             | 5.92e+02 (3.04)     | 1.85e+02 (0.95)        |
| 300                 | 3.15e+02 (1)             | 8.93e+02 (2.83)     | 2.80e+02 (0.88)        |
| 400                 | 4.49e+02 (1)             | 1.27e+03 (2.84)     | 3.86e+02 (0.85)        |
| 500                 | 6.70e+02 (1)             | 1.60e+03 (2.39)     | 4.78e+02 (0.71)        |

Acknowledgment
This research is supported by the project: Development of Verified Numerical Computations and Super High-performance Computing Environment for Extreme Researches, Ministry of Education, Culture, Sports, Science and Technology. We express our sincere thanks to the Information Technology Center of Nagoya University. The function flib_omp_fesetround_ is developed by Fujitsu.

References
[1] Rump S.M 2010 Acta Numerica 19 449
[2] Oishi S and Rump S.M 2002 Numer. Math. 90 773
[3] IEEE Std. 754, 2008
[4] Ogita T, Rump S.M. and Oishi S 2005 SIAM J. Sci. Comput. 26 1988
[5] Ochiai R 2018 Master Thesis Shibaura Institute of Technology
[6] http://www.math.twcu.ac.jp/ogita/post-k/index.html.