HAL
open science

Solving and Certifying the Solution of a Linear System

Hong Diep Nguyen, Nathalie Revol

To cite this version:

Hong Diep Nguyen, Nathalie Revol. Solving and Certifying the Solution of a Linear System. Reliable Computing, 2011, 15 (2), pp.120-131. inria-00546856

HAL Id: inria-00546856

https://inria.hal.science/inria-00546856v1

Submitted on 15 Dec 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Solving and Certifying the Solution of a Linear System

NGUYEN Hong Diep, Nathalie REVOL

INRIA
Université de Lyon
Laboratoire LIP (UMR 5668 CNRS - ENS Lyon - INRIA - UCBL)
École Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
Hong.Diep.Nguyen@ens-lyon.fr, Nathalie.Revol@ens-lyon.fr

Abstract. Using floating-point arithmetic to solve a linear system yields a computed result, which is an approximation of the exact solution because of roundoff errors. In this paper, we present an approach to certify the computed solution. Here, "certify" means computing a guaranteed enclosure of the error. Our method is an iterative refinement method and thus it also improves the computed result. The method we present is inspired from the `verifylss` function of the IntLab library, with a first step, using floating-point arithmetic, to solve the linear system, followed by interval computations to get and refine an enclosure of the error. The specificity of our method is to relax the requirement of tightness of the error, in order to gain in performance. Indeed, only the order of magnitude of the error is needed. Experiments show a gain in accuracy and in performance, for various condition number of the matrix of the linear system.

1 Introduction

In this paper, our approach is presented for solving a linear system and at the same time certifying the computed solution. "Certify" means to compute an enclosure of the error by switching from floating-point arithmetic to interval arithmetic to solve the residual system: this yields a guaranteed enclosure of the error on the exact result. This idea can be found in the `verifylss` function of the IntLab library [17]. `verifylss` first computes a floating-point approximation of the solution using *iterative refinement*, then it switches to interval arithmetic to compute an interval error bound using a method due to Neumaier [11]. Our proposal is to use alternately floating-point arithmetic and interval arithmetic to refine simultaneously the approximation and the error bound.

The use of the residual is the basis of iterative refinement methods [8, 19, 5]. An enclosure of the error can be computed, using interval arithmetic, by adapting one of these iterative refinement methods. These two building blocks,

i.e, the floating-point solution of a linear system and the iterative refinement of the error bounds using interval arithmetic, are combined to produce a more accurate solution along with a tight enclosure of the error. Furthermore, the error bound yields the number of correct digits of the approximate solution.

Another question naturally arises: it is well known that the accuracy of the iteratively refined solution relies for a large part on the computing precision used for the residual computation, but what is the best computing precision? Classically, the computing precision is (at least) doubled [1]. To explore this point, our algorithms are implemented using variable precisions.

This paper is organised as follows: the next section briefly introduces the classical floating-point iterative refinement methods, and its adaptation to interval arithmetic. The starting point of the interval iterative refinement method is an interval enclosure of the solution: its determination is explained in Section 3. Section 4 contains the specific iterative improvement method we use, namely the interval version of Gauss-Seidel iteration [10], and the relaxation we propose. In Section 5, all building blocks introduced in the previous sections are assembled and the complete version of our algorithm is given. Finally, experimental results are given: they demonstrate the efficiency of our method, and also the gain in terms of result accuracy when increasing the computing precisions.

2 Iterative refinement

In this section, we give a brief introduction to the iterative refinement method and we explain how it can be adapted to compute a guaranteed error bound.

2.1 Floating-point iterative refinement

Iterative refinement is a technique for improving a computed solution $\tilde{x}$ to a linear system $Ax = b$. First, the approximate solution $\tilde{x}$ is computed by some method. Then, the residual $r = b - A\tilde{x}$ of the system for this approximation is computed. The exact error e is the solution of a linear system involving the matrix A, with r as the right-hand side: $Ae = r$. By solving, again approximately, the residual system, a correction term $\tilde{e}$ is obtained and is used to update the floating-point approximation. This method is sketched in Algorithm 1 using MatLab notations. Hereafter, MatLab notations are used to describe algorithms: $A \setminus b$ means the solution to the linear system of equations $Ax = b$ computed by some method, and $=$ means assignment. Operations take place between scalars, vectors or matrices as long as their dimensions agree.

Algorithm 1 *Classical iterative refinement*

$\tilde{x} = A \setminus b$;
while (stopping criterion non verified)
    $r = b - A\tilde{x}$;
    $\tilde{e} = A \setminus r$;
    $\tilde{x} = \tilde{x} + \tilde{e}$;
end

If r and $\tilde{e}$ could be computed exactly, then obviously $A(\tilde{x}+\tilde{e}) = A\tilde{x}+r = b$. Hence, the iteration would converge with just one step. Nevertheless, because of rounding errors, none of them are computed exactly.

In the first versions of this method, floating-point iterative refinement is used with Gaussian elimination. Its convergence conditions are provided in [8, 19]. Higham [4] gives a more thorough and general analysis for a generic solver and for both fixed and mixed computing precision. In fixed precision iterative refinement, the working precision is used for all computations. In mixed precision iterative refinement, residuals are computed in twice the working precision.

First, it is stated in [8, 4] that the rates of convergence of mixed and fixed iterative refinement are similar. However, the computing precision used for residual computations affects the accuracy of results. Indeed, given a matrix A which is not too ill-conditioned, the relative error, after convergence, of mixed precision iterative refinement is [4] of order $\|x-\tilde{x}\|_\infty/\|x\|_\infty \approx u$, with u being the relative machine error related to working precision. For fixed precision iterative refinement, the relative error, after convergence, is only of order $\|x-\tilde{x}\|_\infty/\|x\|_\infty \approx 2n \operatorname{cond}(A,x)u$: a relative error of order u is no longer ensured. However, this relative error bound is the best that can be obtained without using higher precision. Usually, fixed precision iterative refinement is used to get a stable solution to linear systems, such as in [7, 6]. Indeed, only one iteration of iterative refinement with only fixed precision accumulation of the residual suffices to make Gaussian elimination componentwise backward stable [19].

2.2 Interval iterative refinement

Notations: intervals are boldface, matrices are uppercase, vectors are lowercase. $[\cdot]$ denotes the result of an expression computed using interval arithmetic.

The idea of the interval version of iterative refinement is to compute an enclosure of the error term instead of approximating it. This enclosure of the error, added to the approximate solution, yields an enclosure of the exact result. The algorithm proceeds as follows: a floating-point approximation of the solution is computed first. Then, the residual is computed using interval arithmetic: it contains the exact residual. The residual system is now an interval linear system. Hence, the solution $\boldsymbol{e}$ to this interval linear system contains the exact correction of the floating-point approximation. Thus, $\tilde{x}+\boldsymbol{e}$ contains the exact solution to the original system. Finally, the floating-point approximate solution $\tilde{x}$ is updated by adding the midpoint of $\boldsymbol{e}$ to it, meanwhile $\boldsymbol{e}$ is centred to zero.

Algorithm 2 *Interval iterative refinement* $\tilde{x} = A \setminus b$;

while (stopping criterion not verified)

$\boldsymbol{r} = [b - A\tilde{x}]$;

$\boldsymbol{e} = A \setminus \boldsymbol{r}$;

$\tilde{x} = \tilde{x} + \operatorname{mid}(\boldsymbol{e})$;

$\boldsymbol{e} = \boldsymbol{e} - \operatorname{mid}(\boldsymbol{e})$;

end

Actually, when computing the initial solution, the residual system is usually preconditioned by R, a floating-point approximate inverse of A, and that multiplication is performed using interval arithmetic. This operation leads to another interval system:

$$\boldsymbol{K}\boldsymbol{e} = \boldsymbol{z}, \text{ where } \boldsymbol{K} = [RA], \boldsymbol{z} = [R\boldsymbol{r}]. \tag{1}$$

The goal is to make the iteration contractant and thus to ensure its convergence. However, the system now has an interval matrix and solving such a system is NP-hard [16, 15]

Algorithms for solving interval linear systems return a box containing the convex hull of the solution, denoted by $\square\Sigma(A, \boldsymbol{r})$, which is thus not the minimal enclosure. Direct algorithms to enclose the solution of an interval linear system exist [3, 13, 11]. Here we prefer an iterative refinement method, which reuses some of the previous floating-point computations. Indeed, as it will be shown in the next section, such an enclosure of that convex hull can be obtained at a cost of $\mathcal{O}(n^2)$ operations. This complexity is asymptotically negligible compared to the cost of solving the original system using floating-point arithmetic.

After each step the approximate solution is updated, and also the error bound on it. This updated error bound serves as the starting point for the next step, or as the initial solution for the next refinement: there's no need to recompute it. Still, to start the refinement, an initial solution to the residual system is needed. The following section explains how to compute an initial solution to an interval linear system.

3 Initial solution

Determining an initial enclosure of the solution is a main issue in the refinement method. The entire real line could be considered as an initial enclosure of each component, but formulas shown in Section 4 would not be able to refine it.

The necessary condition to be able to compute an initial enclosure of the interval linear system $\boldsymbol{K}\boldsymbol{e} = \boldsymbol{z}$ is that $\boldsymbol{K}$ is invertible, i.e. each real matrix $K \in \boldsymbol{K}$ is invertible. Nevertheless, checking the invertibility of an interval matrix is NP-hard [14, 15]. However, if some sufficient condition is satisfied, an initial solution to an interval linear system can be computed. For this purpose, we apply the following proposition to the original, real, matrix A and to the interval right-hand side $\boldsymbol{r}$.

Proposition 3 ([10, Prop. 4.1.9, p. 121]) *Let $\boldsymbol{A} \in \mathbb{IR}^{n\times n}$ be an interval matrix of dimensions $n \times n$ and let $C, C' \in \mathbb{R}^{n\times n}$.*

1. *If $C\boldsymbol{A}C'$ is an H-matrix then, for all $\boldsymbol{b} \in \mathbb{IR}^n$, we have*

$$|\boldsymbol{A}^H\boldsymbol{b}| \leq |C'|\langle C\boldsymbol{A}C'\rangle^{-1}|C\boldsymbol{b}|,$$

2. if $\langle C\boldsymbol{A}C'\rangle u \geq v > 0$ for some $u > 0$ then

$$\begin{aligned} |\boldsymbol{A}^H\boldsymbol{b}| &\leq \|C\boldsymbol{b}\|_v |C'|u, \\ \boldsymbol{A}^H\boldsymbol{b} &\subseteq \|C\boldsymbol{b}\|_v |C'|[-u,u], \end{aligned}$$

where $\boldsymbol{A}^H$ is the convex hull of the inverse of $\boldsymbol{A}$, $\langle\boldsymbol{A}\rangle$ is the comparison matrix of $\boldsymbol{A}$, whose components are defined by:

$$\begin{aligned} \langle\boldsymbol{A}\rangle_{i,i} &= \min(|a_{i,i}|, a_{i,i} \in \boldsymbol{A}_{i,i}), \\ \langle\boldsymbol{A}\rangle_{i,j} &= -\max(|a_{i,j}|, a_{i,j} \in \boldsymbol{A}_{i,j}) \text{ for } j \neq i, \end{aligned}$$

and $\|\boldsymbol{b}\|_v$ is the scaled norm with respect to v: $\|\boldsymbol{b}\|_v = \max(|\boldsymbol{b}_i|/v_i) \quad i \in 1,\ldots,n$.

Following this proposition, a sufficient condition to be able to compute an initial solution is that we exhibit C and C' such that $C\boldsymbol{A}C'$ is an H-matrix. In our algorithm, we use $C = R$, with R an approximate inverse of A, and $C' = I$. If R is a good approximation of the inverse of A then $CAC' = RA \approx I$ is an H-matrix. We also need to exhibit u and v as in the second part of the proposition, to get $\boldsymbol{e} = \|R\boldsymbol{r}\|_v[-u,u]$ as an initial enclosure of the system $A\boldsymbol{e} = \boldsymbol{r}$.

Because all computations are performed using floating-point (as opposed to exact) arithmetic, it is necessary to compute RA using interval arithmetic in order to get a guaranteed result. So the previous proposition is modified as shown below. In the following, $\mathbb{F}$ denotes the set of floating-point numbers.

Proposition 4 *Let $A \in \mathbb{F}^{n\times n}$ and $R \in \mathbb{F}^{n\times n}$ be a floating-point approximate inverse of A. If $\langle[RA]\rangle u \geq v > 0$ for some $u > 0$ then:*

$$\begin{aligned} |A^{-1}\boldsymbol{r}| &\leq \|R\boldsymbol{r}\|_v u, \\ A^{-1}\boldsymbol{r} &\subseteq \|R\boldsymbol{r}\|_v[-u,u]. \end{aligned}$$

What is left now is to find a positive vector u so that $\langle[RA]\rangle u > 0$. In our case, A is a floating-point matrix. If A is well preconditioned, let's say RA is close to identity, or diagonally dominant, then it suffices to use the vector $(1,\ldots,1)^T$ as the value of u and the product $[RA]u$ is positive. If the test of positivity of $[RA]u$ fails, then our algorithm stops and issues a warning of failure.

4 Relaxed interval iterative refinement

There exist several methods of interval iterative refinement, such as the methods of Jacobi and of Gauss-Seidel (introduced below), or the method of Krawczyk. Krawczyk method converges quadratically, but Gauss-Seidel iteration always yields tighter intervals than Krawczyk iteration, when applied to a preconditioned system [10, Theorem 4.3.5]. This section details our method for the iterative improvement of an initial enclosure $\boldsymbol{e}$ of the solution to the system $\boldsymbol{K}\boldsymbol{e} = \boldsymbol{z}$: it is a relaxed interval version of the Gauss-Seidel iteration.

4.1 Jacobi and Gauss-Seidel methods

Given an initial enclosure $\boldsymbol{e}$ to the interval linear system $\boldsymbol{K}\boldsymbol{e} = \boldsymbol{z}$, an improved approximate enclosure is obtained by writing the linear system satisfied by $\tilde{e} \in \boldsymbol{e}$: $\exists \tilde{K} \in \boldsymbol{K}, \exists \tilde{z} \in \boldsymbol{z} \ : \ \tilde{K}\tilde{e} = \tilde{z}$. Developing the i-th line and separating the i-th component, one gets:

$$\tilde{e}_i = \left(\tilde{z}_i - \sum_{j=1}^{i-1} \tilde{K}_{i,j}\tilde{e}_j - \sum_{j=j+1}^{n} \tilde{K}_{i,j}e_j \right) / \tilde{K}_{i,i} \text{ and } \tilde{e}_i \in \boldsymbol{e}_i.$$

Replacing punctual terms by the corresponding interval terms yields the formula of the interval Jacobi iteration:

$$\tilde{e}_i \in \boldsymbol{e}'_i := \left(\boldsymbol{z}_i - \sum_{j=1}^{i-1} \boldsymbol{K}_{i,j}\boldsymbol{e}_j - \sum_{j=j+1}^{n} \boldsymbol{K}_{i,j}\boldsymbol{e}_j \right) / \boldsymbol{K}_{i,i} \cap \boldsymbol{e}_i.$$

Taking into account that for components that have already been refined, $\tilde{e_j}$ belongs to both original value $\boldsymbol{e}_j$ and refined valued $\boldsymbol{e}'_j$, $\boldsymbol{e}_j$ can be replaced by $\boldsymbol{e}'_j$ for $j < i$ to obtain the Gauss-Seidel iteration.

$$\boldsymbol{e}'_i = \left(\boldsymbol{z}_i - \sum_{j=1}^{i-1} \boldsymbol{K}_{i,j}\boldsymbol{e}'_j - \sum_{j=j+1}^{n} \boldsymbol{K}_{i,j}\boldsymbol{e}_j \right) / \boldsymbol{K}_{i,i} \cap \boldsymbol{e}_i. \tag{2}$$

Taking the intersection with the former iterate $\boldsymbol{e}_i$ implies the contracting property of both Jacobi and Gauss-Seidel iterations. Hence, both iterations converge. Nevertheless, making full use of the refined values, Gauss-Seidel iterations converge much more quickly than Jacobi iterations. As mentioned in Section 3, our sufficient condition to compute an initial solution to the interval residual system is that $[RA]$ is an H-matrix. Under this condition, Gauss-Seidel iterations converge very quickly. Actually, in the experiments presented in Section 6, five iterations suffice to obtain accurate results.

Let us now detail the complexity of an iteration. The refinement of each component requires $n-1$ interval multiplications, n interval additions and one interval division. Thus, in total, each Gauss-Seidel iteration costs $\mathcal{O}(n^2)$ interval operations. Hence, theoretically, the refinement stage should not affect the overall cost of the method. In practice however, as the number of iterations increases, the execution time of the refinement increases significantly because of interval computations. Indeed, interval operations are usually slower than floating-point operations, within a factor up to 20. In the next section, we propose a relaxation technique to reduce the cost of the refinement step. The idea is to use floating-point operations rather than interval ones when possible.

4.2 Relaxation

The refinement step is used to improve the error bound upon a computed approximation. This error bound should correspond to lost bits in the approximation. Hence, it is not necessary to compute an error bound with high accuracy.

Thus we relax the tightness requirement on this error bound to speed up the program. To gain in performance, the matrix of the system is enlarged, so as to have its off-diagonal elements centred in 0. Formulas for subsequent operations are thus simplified: the computed intervals are symmetrical around 0 and only one endpoint is computed, using floating-point arithmetic.

Let $\boldsymbol{D} \in \mathbb{IR}^{n\times n}$ and $\boldsymbol{M} \in \mathbb{IR}^{n\times n}$ be defined by

$$\begin{aligned}
\boldsymbol{D}_{i,j} &= \begin{cases} \boldsymbol{K}_{i,j} & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases} \\
\boldsymbol{M}_{i,j} &= \begin{cases} 0 & \text{if} \quad i = j \\ [-\text{mag}\,(\boldsymbol{K}_{i,j}), \text{mag}\,(\boldsymbol{K}_{i,j})] & \text{if} \quad i \neq j \end{cases}
\end{aligned}$$

Because $\boldsymbol{x} \in [-\text{mag}\,(\boldsymbol{x}), \text{mag}\,(\boldsymbol{x})]$ and $\text{mag}\,(\boldsymbol{x}) = \text{mag}\,([-\text{mag}\,(\boldsymbol{x}), \text{mag}\,(\boldsymbol{x})])$ for all $\boldsymbol{x} \in \mathbb{IR}$, we have:

$$\boldsymbol{K} \subseteq \boldsymbol{D} + \boldsymbol{M} \tag{3}$$

$$\langle \boldsymbol{K} \rangle = \langle \boldsymbol{D} + \boldsymbol{M} \rangle. \tag{4}$$

From (3), we deduce that the solution set of system (1) is included in the solution set of the system:

$$(\boldsymbol{D} + \boldsymbol{M})\boldsymbol{e} = \boldsymbol{z} \tag{5}$$

In particular, if $\boldsymbol{K}$ is centred about the identity matrix (ideally $RA = I$), then the equality in (3) holds, and the two systems have the same solution set.

Moreover, (4) means that if $\boldsymbol{K}$ is an H-matrix then $(\boldsymbol{D} + \boldsymbol{M})$ is also an H-matrix. Hence both systems have the same convergence property. Let us apply the Gauss-Seidel iteration for system (5) to get a relaxed solution to (1):

$$\boldsymbol{e}'_i = \left(\boldsymbol{z}_i - \sum_{j=1}^{i-1} \boldsymbol{M}_{i,j}\boldsymbol{e}'_j - \sum_{j=i+1}^{n} \boldsymbol{M}_{i,j}\boldsymbol{e}_j \right) / \boldsymbol{D}_{i,i} \cap \boldsymbol{e}_i. \tag{6}$$

Since $\boldsymbol{M}_{i,i} = 0$ and $\boldsymbol{M}_{i,j}$, for $i \neq j$, is symmetrical around 0, each product $\boldsymbol{M}_{I,j}\boldsymbol{y}_j$, for any $\boldsymbol{y}_j \in \mathbb{IR}$, can be written as:

$$\boldsymbol{M}_{i,j}\boldsymbol{y}_j = \left[-\text{mag}\,(\boldsymbol{M}_{i,j})\text{mag}\,(\boldsymbol{y}_j), \text{mag}\,(\boldsymbol{M}_{i,j})\text{mag}\,(\boldsymbol{y}_j)\right]$$

Thus, if the Gauss-Seidel iteration is performed in place (i.e. the new iterate $\boldsymbol{e}'$ is stored at the memory location of the previous iterate $\boldsymbol{e}$), then there is no need to distinguish between $\boldsymbol{e}'$ and $\boldsymbol{e}$ and the iteration can be written as follows. Denote by $\boldsymbol{M}_i$ the i-th row of $\boldsymbol{M}$, the truncated solution of (5) is computed by

$$\boldsymbol{e}_i = \left(\boldsymbol{z}_i - \left[-\text{mag}\,(\boldsymbol{M}_i)^T\text{mag}\,(\boldsymbol{e}), \text{mag}\,(\boldsymbol{M}_i)^T\text{mag}\,(\boldsymbol{e})\right]\right) / \boldsymbol{D}_{i,i} \cap \boldsymbol{e}_i \tag{7}$$

In comparison with the interval iterative refinement (2) of the original residual system, there is no computation of an interval dot product, the interval refinement of relaxed system only requires a floating-point dot product, $\text{mag}\,(\boldsymbol{M})^T\text{mag}\,(\boldsymbol{e}^i)$, which helps to reduce a lot the execution time. Note that, to control rounding errors, this floating-point dot product must be computed using upward rounding mode.

5 Algorithm

The complete algorithm we propose is given below, it uses all building blocks introduced above.

Algorithm 5 *Solve and certify a linear system*

Compute the LU decomposition of A
Compute an approximation $\tilde{x}$ with a forward and a backward substitution using L and U
Compute an approximative inverse of A, by solving $RL = \texttt{inv}(U)$ [5, Chapter 14]
Precondition the system: $\boldsymbol{K} = [RA]$
Test if $\boldsymbol{K}$ is an H-matrix by computing a non-negative vector u such that $\langle \boldsymbol{K} \rangle u \geq v > 0$.
If fail to compute u, display a warning "`Failed to certify the solution. The system may be either singular or too ill-conditioned`" and exit
Compute the residual $\boldsymbol{r} = [b - A\tilde{x}]$ in double the working precision
Precondition the residual by R: $\boldsymbol{z} = R\boldsymbol{r}$
Compute an initial error bound $\boldsymbol{e} = \|\boldsymbol{z}\|_v[-u, u]$
While (not converged)
Apply five Gauss-Seidel iterations on $\boldsymbol{K}, \boldsymbol{z}$, and $\boldsymbol{e}$
Update $\tilde{x}$ and $\boldsymbol{e}$
Recompute $\boldsymbol{r}$ and $\boldsymbol{z}$
End

As mentioned in Section 2.2, the width of the interval error decreases after each step. So it is a non-negative non-increasing series. This property leads to two stopping criteria (for a study of stopping criteria, see [1]).

Firstly, we stop the computation whenever reaching a required number of accurate bits. For example, working with double floating-point numbers, there is no point of getting a result which is accurate to more than 52 bits. Using interval arithmetic, it is quite easy to compute the number of exact bits in the result via componentwise relative errors:

$$\text{nb_bits} = -\log_2 \left(\max \left(\frac{\text{wid}\,(\boldsymbol{e}_i)}{|\tilde{x}_i|} \right) \right). \tag{8}$$

Nevertheless, the program can end up without reaching the required number of exact bits. Hence, we have to detect whether the iteration yields extra correct digits or stagnates. Let $\boldsymbol{e}$ and $\boldsymbol{e}'$ be two successive iterates, a second stopping criterion is that no more correct digit is obtained in any component of the approximation:

$$\max_j \left(\frac{\left|\text{wid}\,\left(\boldsymbol{e}'_j\right) - \text{wid}\,\left(\boldsymbol{e}_j\right)\right|}{|\tilde{x}_j^{(i)}|} \right) < u. \tag{9}$$

In cases where none of these two criteria above is matched, it is necessary to stop the computation when one has tried enough. Practically, in our implementations, the maximum number of iterations is set to 10.

6 Experiments

In this section we present some experimental results to demonstrate the efficiency of our method, as well as to study the effect of computing precisions.

6.1 Implementation in MatLab

In what follows, the implementations in MatLab, using the interval library IntLab, of Algorithm 2, and Algorithm 5 using Equation (7), are called respectively `certifylss` and `certifylss_relaxed`. The computing precision used for all computations is the IEEE double floating-point precision, except for the residual computation, which is computed in twice the working precision.

Figure 1 depicts results computed by these two functions, by the function `verifylss` of the IntLab library, and non-certified results computed by MatLab. Certified results provided by all certified functions: `certifylss`, `certifylss_relaxed` and `verifylss` are much more accurate than non-verified results provided by MatLab, at the price of a higher execution time.

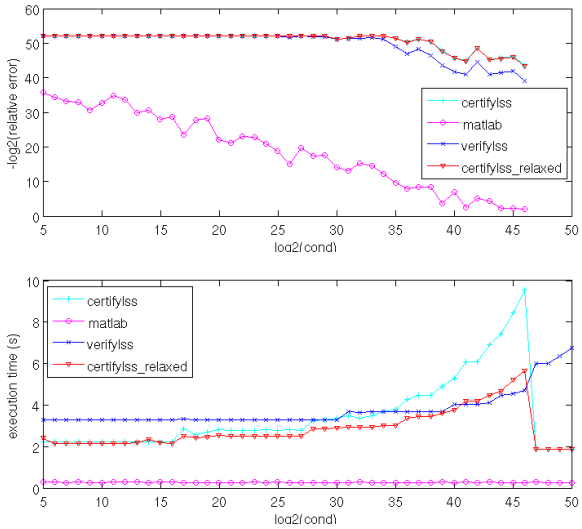


Figure 1: MatLab implementation results for 1000×1000 matrices.

When the coefficient matrix is not too ill-conditioned, all three certified functions provide the same accuracy (52 bits). However, `certifylss` runs faster than `verifylss`, because `certifylss` does not compute a tight error bound. As expected, `certifylss_relaxed` runs even faster. As for the `verifylss` function, it first applies the iterative refinement on the floating-point approximation. Then it computes a tight error bound for the refined approximation using a method of Neumaier [11], which computes an upper bound of the inverse of $\langle [RA] \rangle$: this explains the overhead on the execution time.

When the condition number increases, an accuracy of 52 bits cannot be obtained any more. In that case, `certifylss` and `certifylss_relaxed` provide slightly more accurate results. However, the execution time for `certify` becomes higher than the execution time for `verifylss`, and for higher condition number this becomes true even for `certifylss_relaxed`.

When the condition number gets close to $1/u$, all three functions fail to certify the solution. Indeed, a good approximation of the inverse of A cannot be obtained, so the sufficient condition that RA is an H-matrix does not hold.

It is noticeable that `certifylss_relaxed` achieves the same accuracy as `certify`, even if, by design, it relaxes the tightness constraint. Indeed, it obtains the most significant bits of the error bound, which are the only ones needed.

6.2 Implementation using variable computing precisions

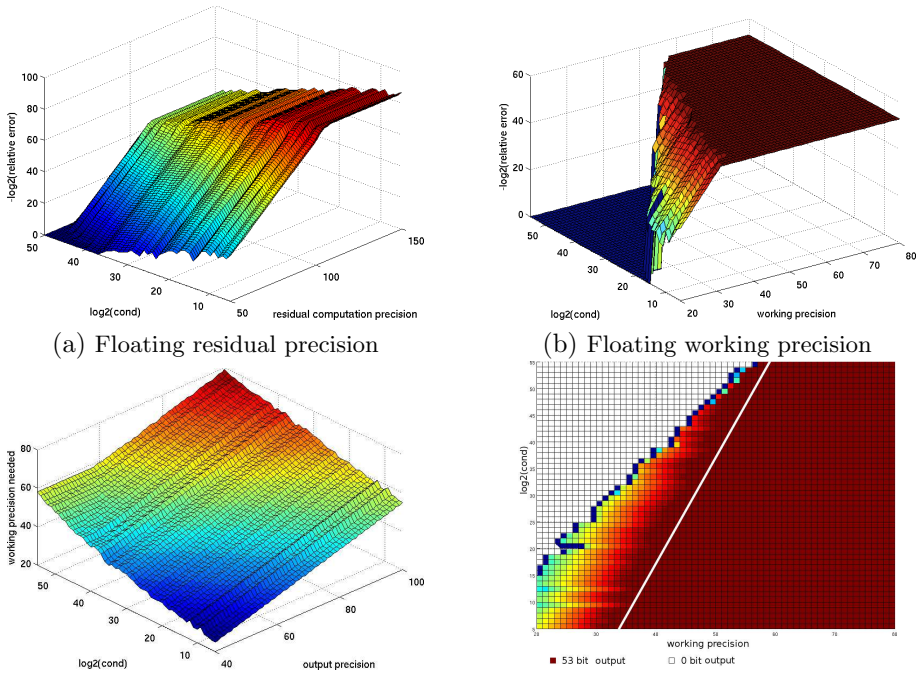
MPFR [2] and MPFI [12] are libraries that offer arbitrary precision for floating-point arithmetic and interval arithmetic, respectively. Using these two libraries to implement our algorithms, computing precision can be tuned at each step in order to study the effect of these precisions on the result accuracy.

Matrices used in the following tests are generated by function `gallery` $('randsvd', dim, cond)$, where dim is matrix dimension (taken as 1000), and $cond$ is the condition number. Coefficients of generated matrices are IEEE double floating-point numbers. The condition number varies between 2^5 and 2^{50}.

First, the precision for all the computations is fixed, except for the residual computation, to study the effect on the result accuracy. Figure 2(a) depicts results obtained with a fixed working precision of 53 bits. When the condition number decreases, the number of guaranteed correct bits (computed as in Eq.(8)) increases nearly linearly. When the residual computation precision increases, the number of guaranteed correct bits also increases linearly. However, when the residual computation precision gets a bit higher than twice the working precision, the number of correct bits stops increasing: this precision becomes too high and it does not have any effect when the other precisions remain low. Practically, this phenomenon justifies the use of twice the working precision for residual computations in iterative methods. From now on, the computing precision for the residual will be set to twice the working precision.

Next the working precision varies and the results are shown in figure 2(b): it is not necessary to use a precision equal to the output precision to get a result which is correct to the last bit. For a fixed output precision, the needed working precision depends on the condition number, and this dependency is again linear. As revealed in figure 2(d) (view from top of 2(b)), when the condition number increases by 0.5 bit, the computing precision should be increased by 1 bit in order to get a result which is correct to the last bit.

In the final set of experiments, the demanded number of correct bits varies, and the minimal working precision that reaches the prescribed accuracy is determined. As shown in Figure 2(c), the minimal working precision depends nearly linearly on the condition number and the number of correct bits demanded.



(a) Floating residual precision

(b) Floating working precision

(c) Minimal working precision needed

(d) Floating working precision (seen from top)

Figure 2: Effects of the working precision and of the residual precision.

Conclusions and future work

Our method is based on an iterative refinement and on interval arithmetic for guaranteeing the results. Relaxing the residual system enabled to gain in performance without loosing in accuracy.

Experiments on computing precisions illustrate the linear relation between the computing precision and the result accuracy. They also confirm that in most cases, doubling the working precision suffices to get the best results.

The method presented here could also be used to solve interval linear systems of small width, with the preconditioning matrix being the approximate inverse of the midpoint coefficient matrix. In the future, the same philosophy and techniques could be extended to other problems. For instance, linear systems of inequalities could be solved using slack variables. Another example is the solution of nonlinear systems by transforming them into linear systems using Taylor models [9] of order 1. The idea here is to replace a nonlinear expression by a Taylor order expansion of order 1, ie. by a linear form plus a constant interval (which contains all truncation errors).

Acknowledgements. This work was supported by the ANR project EVA-Flo and by "Pôle de compétitivité mondial" Minalogic.

References

[1] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy, *Error bounds from extra-precise iterative refinement*, ACM Trans. Mathematical Software **32** (2006), no. 2, 325–351.

[2] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann. *MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding*, ACM Trans. Mathematical Software **33** (2007), no. 2, article 13. `http://www.mpfr.org/`

[3] E. R. Hansen, *Bounding the Solution of Interval Linear Equations*, SIAM Journal on Numerical Analysis **29** (1992), no. 5, 1493–1503.

[4] N.J. Higham, *Iterative refinement for linear systems and LAPACK*, IMA Journal of Numerical Analysis **17** (1997), no. 4, 495–509.

[5] ———, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM Press, 2002.

[6] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra, *Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems) - Article 113 (17 pages)*, Proc. ACM/IEEE conf. on Supercomputing, 2006.

[7] X.S. Li, and J Demmel. *SuperLu_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Mathematical Software **29** (2003), no. 2, 110–140.

[8] C.B. Moler, *Iterative Refinement in Floating Point*, J. ACM **14** (1967), no. 2, 316–321.

[9] M. Neher, *From Interval Analysis to Taylor Models - An Overview (8 pages)*, IMACS'05, 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, 2005.

[10] A. Neumaier, *Interval Methods for Systems of equations*, Cambridge University Press, 1990.

[11] ———, *A Simple Derivation of the Hansen-Bliek-Rohn-Ning-Kearfott Enclosure for Linear Interval Equations*, Reliable Computing **5** (1999), no. 2, 131–136, Erratum in Reliable Computing **6** (2000), no. 2, p. 227.

[12] N. Revol and F. Rouillier. *Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library*, Reliable Computing **11** (2005), no. 4, 275–290. `http://gforge.inria.fr/projects/mpfi/`

[13] J. Rohn, *Cheap and Tight Bounds: The Recent Result by E. Hansen Can Be Made More Efficient*, Interval Computations (1993), no. 4, 13–21.

[14] _____, *Checking Properties of Interval Matrices*, Tech. Report 686, Czech Academy of Sciences, 1996.

[15] _____, *A Handbook of Results on Interval Linear Problems*, Czech Academy of Sciences, 2005, http://www.cs.cas.cz/ rohn/publist/handbook.

[16] J. Rohn and V. Kreinovich, *Computing Exact Componentwise Bounds on Solutions of Linear Systems with Interval Data is NP-hard*, SIAM Journal on Matrix Analysis and Applications **16** (1995), no. 2, 415–420.

[17] S.M. Rump, *INTLAB - INTerval LABoratory*, `http://www.ti3.tu-hamburg.de/rump/intlab`.

[18] _____, *Handbook on Accuracy and Reliability in Scientific Computation (edited by Bo Einarsson)*, ch. Computer-assisted Proofs and Self-validating Methods, pp. 195–240, SIAM, 2005.

[19] R.D. Skeel, *Iterative Refinement Implies Numerical Stability for Gaussian Elimination*, Mathematics of Computation **35** (1980), no. 151, 817–832.