HAL
open science

Influence of the Condition Number on Interval Computations: Illustration on Some Examples

Nathalie Revol

To cite this version:

Nathalie Revol. Influence of the Condition Number on Interval Computations: Illustration on Some Examples. O. Kosheleva, S.P. Shary, G. Xiang, R. Zaparin. Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy etc. Methods and Their Applications, 835, Springer, pp.359–373, 2020, Studies in Computational Intelligence, 978-3-030-31041-7. 10.1007/978-3-030-31041-7_20 . hal-01588713

HAL Id: hal-01588713

https://inria.hal.science/hal-01588713v1

Submitted on 16 Sep 2017

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.

Influence of the Condition Number on Interval Computations: Illustration on Some Examples

Nathalie Revol

Abstract The *condition number* is a quantity that is well-known in "classical" numerical analysis, that is, where numerical computations are performed using floating-point numbers. This quantity appears much less frequently in interval numerical analysis, that is, where the computations are performed on intervals.

The goal of this paper is twofold. On the one hand, it is stressed that the notion of condition number already appears in the literature on interval analysis, even if it does not bear that name. On the other hand, three small examples are used to illustrate experimentally the impact of the condition number on interval computations. As expected, problems with a larger condition number are more difficult to solve: this means either that the solution is not very accurate (for moderate condition numbers) or that the method fails to solve the problem, even inaccurately (for larger condition numbers). Different strategies to counteract the impact of the condition number are discussed and experimented: use of a higher precision, iterative refinement, bisection of the input. More strategies are discussed as a conclusion.

1 Introduction

Condition number is a quantity that is commonly used in "classical" numerical analysis, that is, numerical analysis where computations are performed using floating-point arithmetic. Condition number is used to predict, or to explain, whether a problem is difficult to solve accurately or not. More precisely, the condition number indicates how sensitive the solution is to a perturbation of the input. If there is uncertainty on the input, or a small error such as a rounding error, this error is very likely to be amplified by a factor at most, but often close to, the condition number. This is known as the *rule of thumb* in [5, §1.6, p. 9].

Nathalie Revol
Inria, University of Lyon – LIP, ENS de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France, e-mail: Nathalie.Revol@inria.fr

In our experience with interval computations, we have noticed a similar behavior: problems with small condition number were easy to solve and problems with large condition number were not that easy – in a sense that we will comment on. However, the condition number is not a quantity one encounters frequently in works on interval computations. We will detail in Section 2 the formulas for the condition number and for a theorem given in [8]: these two formulas use a similar quantity as the amplification factor for the uncertainty in both contexts. Actually, one rather uses the condition number as the amplification factor for relative errors in classic numerical analysis and a quantity that is closer to the *sensitivity* as the amplification factor for absolute errors in interval computations. We will still use the denomination *condition number* for both, throughout the paper.

The goal of this paper is to put into light, through three small illustrative examples, the impact of the condition number on interval computations. These examples are first, the summation of n numbers, then the solution of a linear system of dimension n and eventually the solution of a univariate, but nonlinear, equation. These examples are chosen amoong the most classical problems discussed in numerical analysis, still they exhibit interesting features. They are introduced here in increasing order of difficulty. Indeed, summation involves only addition, and each variable is used only once. Linear system solving involves also multiplication and division, and variables are used more than once, which is relevant for interval computations, where the so-called *dependency problem* is one of the main causes of overestimation. The last problem is not only nonlinear, it also involves more elaborate functions (such as the logarithm in our example). In Section 3, we will detail the vectors, with varying condition number for the summation problem and the accuracy of their sum, depending on this condition number. In Section 4, we will describe the method used to solve linear systems and we will present experimentally the influence of the condition number, either on the accuracy of the solution or on the ability of the method to solve the problem. In Section 5, we will introduce an example of ill-conditioned (for the determination of zeros) nonlinear equation and, again, illustrate experimentally with interval Newton's method, what happens when the condition number increases.

In all three cases, the impact of the condition number is visible and as expected. In all three cases, we experimented some strategies to counteract this impact. For the linear problems, the use of a higher precision can obviate the impact of the condition number. For the summation problem, an increase of the computing precision is tested. Regarding the solution of linear systems: we will illustrate how combining the use of iterative refinement with the choice of the computing precision allows one to get a fully accurate solution... when the method succeeds in computing the solution. The key point is to restrict the higher precision to the most sensitive parts of the computation. For nonlinear systems, again it is not difficult to target the parts that are most sensitive to the computing precision, but it is not always obvious to so without resorting to a dedicated library for high precision arithmetic. In this case, our experiments focus instead on another, naive but always applicable, strategy: the bisection of the input interval to get a narrower enclosure of the sought zero as output.

For the summation and the nonlinear equation solving, our experiments were performed using the `interval` package of Octave, version 2.1.0 [4]. The linear system solving algorithms and experiments are taken from Nguyen's PhD thesis [10], they have been conducted using IntLab in MATLAB.

2 Condition Number and Interval Computations

Let us start by recalling the notion of condition number of a problem in classic numerical analysis in Section 2.1. How an error on the input is amplified, how it results in an error on the output, gives rise to this notion of condition number: it is the amplification factor of the relative errors. In Section 2.2, computations are performed using interval arithmetic. A similar study on the effect, on the output, of an error on the input gives rise to a theorem about the amplification factor in this case. Section 2 also contains the definitions and notations used in this paper. The main references for this section are Higham [5, §1.6, p. 9] for Section 2.1 and Neumaier [8, §2.1] for Section 2.2.

2.1 Condition Number of a Problem

Let us denote by $x \in \mathbb{R}$ the input and by $y = f(x) \in \mathbb{R}$ the solution of a considered problem, or its output. We are interested in the variations of x and y: when the variation of the input is Δx, the output of the new problem is $y + \Delta y = f(x + \Delta x)$ and the variation of the output is Δy. If f is twice continuously differentiable,

$$\begin{aligned} y + \Delta y = f(x + \Delta x) &= f(x) + f'(x)\Delta x + \mathcal{O}(\Delta x^2) \\ \Rightarrow \Delta y = f(x + \Delta x) - f(x) &= f'(x)\Delta x + \mathcal{O}(\Delta x^2). \end{aligned}$$

If Δx is an error on x, then Δy is the error on the solution, due to this error on the input. The absolute error Δx on the input is amplified by a factor close to $|f'(x)|$:

$$\Delta y \simeq f'(x)\Delta x \Rightarrow |\Delta y| \simeq |f'(x)|.|\Delta x|. \tag{1}$$

The amplification factor for absolute errors is sometimes referred to as *sensitivity*, especially for multidimensional inputs.

The relative error on the ouput is $\Delta y / y$ if $y \neq 0$, or $|\Delta y| / |y|$. The previous equality yields

$$\frac{\Delta y}{y} = \frac{f'(x)\Delta x}{f(x)} + \mathcal{O}(\Delta x^2)$$

and, if $x \neq 0$, the ratio of the relative error on the output by the relative error on the input is

$$\frac{\Delta y/y}{\Delta x/x} = \frac{f'(x)\Delta x/f(x)}{\Delta x/x} + \mathscr{O}(\Delta x/x) = \frac{xf'(x)}{f(x)} + \mathscr{O}(\Delta x/x) \simeq \frac{xf'(x)}{f(x)}.$$

The amplification factor of the relative error is thus

$$c_f(x) = \left|\frac{\Delta y/y}{\Delta x/x}\right| = \frac{|f'(x)|.|x|}{|f(x)|}. \tag{2}$$

The quantity $c_f(x)$ is called the *condition number* of the problem f at x.

For problems with higher dimensions: $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, a similar reasoning yields

$$\Delta y = f(x+\Delta x) - f(x) = Jf(x).\Delta x + \mathscr{O}(\|\Delta x\|_x^2)$$

where $Jf(x)$ is the Jacobian of f in x and the norm $\|.\|_x$ applies to vectors in $\mathbb{R}^n$. In what follows, the norm $\|.\|_y$ applies to vectors in $\mathbb{R}^m$ and the matrix norm $\|.\|_{x,y}$ is the matrix norm induced by these vector norms. The ratio of the relative error on the output, if $y \neq 0$, on the relative error on the input, if $x \neq 0$, satisfies

$$\frac{\|\Delta y\|_y/\|y\|_y}{\|\Delta x\|_x/\|x\|_x} \leq \frac{\|\,|Jf(x)|.|x|\,\|_y}{\|f(x)\|_y} \leq \frac{\|Jf(x)\|_{x,y}.\|x\|_x}{\|f(x)\|_y}.$$

Again, the *condition number* $c_f(x)$ of the problem f at x is an upper bound on the amplification factor of the relative error:

$$c_f(x) = \frac{\|\,|Jf(x)|.|x|\,\|_y}{\|f(x)\|_y} \text{ or } c_f(x) = \frac{\|Jf(x)\|_{x,y}.\|x\|_x}{\|f(x)\|_y}. \tag{3}$$

2.2 Amplification Factor for Interval Computations

Let us denote again by $x \in \mathbb{R}$ the real input of the problem and $y = f(x) \in \mathbb{R}$ the real output. Let us assume that f is smooth enough: being $\mathscr{C}^1$ (or sometimes $\mathscr{C}^2$) usually suffices.

Let us now consider the case of interval computations. Intervals are denoted in boldface, as in $\mathbf{x}$, $\mathbf{y}$. Let x vary in an interval $\mathbf{x}$, the output varies in an interval $f(\mathbf{x})$ the range of f over $\mathbf{x}$. Let us assume that f is given by an arithmetic expression and that f is Lipschitz-continuous in $\mathbf{x}$ (in the sense defined in [8, §2.1, p.33]). The evaluation of f over $\mathbf{x}$ using interval arithmetic usually does not produce $f(\mathbf{x})$, but a larger (in the sense of inclusion) interval that will be denoted by $\mathbf{f}(\mathbf{x})$.

Similarly, the evaluation of $f'(\mathbf{x})$ using the arithmetic expression for f and the rules for the derivation of each operation, such as the chainrule for the derivation of a product, without any simplification, yields $\mathbf{f}'(\mathbf{x}) \supset f'(\mathbf{x})$. Let us denote by $\lambda_f(\mathbf{x})$ the Lipschitz constant in the definition of f being Lipschitz-continuous in $\mathbf{x}$, $\lambda_f(\mathbf{x})$ is obtained in a similar way to the evaluation of $f'(\mathbf{x})$, by taking absolute values at each step. Thus $\lambda_f(\mathbf{x}) \geq |f'(\mathbf{x})|$.

The distinction between $\lambda_f(\mathbf{x})$ and $|f'(\mathbf{x})|$, that is, between the interval evaluation of $\lambda_f(\mathbf{x})$ over $\mathbf{x}$ using inductively the arithmetic expression for f, and the maximal absolute value in the range of the real function f' over $\mathbf{x}$, becomes clear in the following example. If a function contains (usually in a hidden form) a subexpression of the form $f(x) = x - x$, then the interval evaluation $\mathbf{f}(\mathbf{x})$, when $\mathbf{x} = [\underline{x}, \overline{x}]$, is $[\underline{x} - \overline{x}, \overline{x} - \underline{x}]$ that contains 0 but not only, and that is twice as large as $\mathbf{x}$: $\mathrm{wid}(\mathbf{f}(\mathbf{x})) = (\overline{x} - \underline{x}) - (\underline{x} - \overline{x}) = 2(\overline{x} - \underline{x}) = 2\mathrm{wid}(\mathbf{x})$. The value of $\lambda_f(\mathbf{x})$ is obtained as follows:

- the derivative of each occurrence of x is 1 and so is the corresponding Lipschitz constant;
- the Lipschitz constant of a sum or difference of two terms is the sum of the Lipschitz constants of these terms (there is a sign error in the formula for the subtraction in [8, Table 2.1], but not in the proof of it: the Lipschitz constants must be added, never substracted).

Thus the Lipschitz constant for $f(x) = x - x$ is 2. It corresponds to the fact that the width of $\mathbf{f}(\mathbf{x})$ is twice the width of $\mathbf{x}$.

This example is a specific case of a general statement: Theorem 2.1.1 in [8] applied to $\{f(x)\}$ and to $\mathbf{f}(\mathbf{x})$ yields

$$\mathrm{wid}(\mathbf{f}(\mathbf{x})) \leq \lambda_f(\mathbf{x})\mathrm{wid}(\mathbf{x}). \tag{4}$$

Equation (4) is analogous to Eq. (1), as long as we keep in mind the distinction between $|f'(\mathbf{x})|$ and $\lambda_f(\mathbf{x})$.

As it can be difficult to define what is the value of interest in an interval, it is difficult to define a notion of relative error that corresponds to all contexts: should $\frac{\mathrm{rad}(\mathbf{x})}{|\mathrm{mid}(\mathbf{x})|}$ be used, or $\frac{\mathrm{rad}(\mathbf{x})}{|\mathbf{x}|}$, that yields the smallest possible value, or $\frac{\mathrm{rad}(\mathbf{x})}{\mathrm{mig}(\mathbf{x})}$ where $\mathrm{mig}(\mathbf{x}) = \min\{|x| : x \in \mathbf{x}\}$, that yields the largest possible value? As there is no universal notion of relative error in interval computations, we will not proceed any further in our attempt to mimic and adapt the definition of condition number for interval computations. In the experiments below, only the width of the output will be observed.

We will thus stick to Eq. (4) and this bound $\lambda_f(\mathbf{x})$ on the amplification factor for the error on the input. As $|f'(x)|$ is less than $\lambda_f(\mathbf{x})$, only $|f'(x)|$, or the usual condition number of the problem will vary in our experiments, and the effect of this condition number on the width of the output will be observed.

3 Summation

The first problem considered in this paper is the summation of n real numbers $x_1, \ldots, x_n$: if $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the problem is to compute

$$s(x) = \sum_{i=1}^{n} x_i.$$

In our experiments, the $\mathbf{x}_i$ are chosen as tiny intervals around a given real value: $\mathbf{x}_i = [\mathrm{RD}(x_i), \mathrm{RU}(x_i)]$. The sum $\mathbf{s}$ is computed using interval addition and from left to right. In Octave this is done as

```
s = infsup (0.0);
for  i = 1:n, s = s + x(i); end;
```

Let us apply the first of the two possible formulas in Eq. (3) to determine the condition number of this problem. (This is exercice 4.1 in [5, Chapter 4, p.91].) The Jacobian of s at any x is

$$Js(x) = \left(\frac{\partial s}{\partial x_1}(x),\ \frac{\partial s}{\partial x_2}(x) \ldots \frac{\partial s}{\partial x_n}(x) \right) = (1,\, 1, \ldots 1).$$

Thus $|Js(x)|\,|x| = \sum_{i=1}^{n} |x_i|$ and thus

$$c_s(x) = \frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|}.$$

From this expression, it is clear that the summation problem is ill-conditioned when $\sum_{i=1}^{n} |x_i|$ is much larger than $|\sum_{i=1}^{n} x_i|$: inputs x that correspond to ill-conditioned problems are problems where heavy cancellations occur.

Our tests use the following vector x, of odd dimension n, parametrized by c:

- $x_1, \ldots x_{\lceil \frac{n}{2} \rceil - 1}$ are positive,
- $x_{\lceil \frac{n}{2} \rceil + 1}, \ldots x_n$ are negative, equal to $-x_1, \ldots -x_{\lceil \frac{n}{2} \rceil - 1}$ so that cancellations occur,
- $x_{\lceil \frac{n}{2} \rceil} = 1$ thus the sum of the x_i is 1,
- the x_i vary greatly in magnitude, so that cancellations occur for every order of magnitude, however the sum of their absolute value is large and thus the condition number is large; we use the successive powers of 10 in a round-robin way: we set $x_1 = 10^1, x_2 = 10^2 \ldots x_c = 10^c$ and then again $x_{c+1} = 10^1, x_{c+2} = 10^2 \ldots$.

The formulas for x and $\mathbf{x}$ are

- from x_1 to $x_{\lceil n/2 \rceil - 1}$,

$$x_i = 10^{(i-1 \mod c)+1},\ \mathbf{x}_i = [\mathrm{RD}(10^{(i-1 \mod c)+1}), \mathrm{RU}(10^{(i-1 \mod c)+1})],$$

- from $x_{\lceil n/2 \rceil} + 1$ to x_n,

$$\begin{aligned} x_i &= -10^{(i-\lceil n/2 \rceil - 1 \mod c)+1}, \\ \mathbf{x}_i &= [\mathrm{RD}(-10^{(i-\lceil n/2 \rceil - 1 \mod c)+1}), \mathrm{RU}(-10^{(i-\lceil n/2 \rceil - 1 \mod c)+1})], \end{aligned}$$

- $x_{\lceil n/2 \rceil} = 1$, $\mathbf{x}_{\lceil n/2 \rceil} = 1$.

If $n > 2c$, the sum $\sum_{i=1}^{n} |x_i| = \frac{n}{c} \sum_{i=1}^{c} 10^i \simeq \frac{n}{c} 10^c$ and the condition number $c_s(x) \simeq \frac{n}{c} 10^c$: the radix-10 logarithm of the condition number, which is the number of decimal digits, is close to c.

In the experiments presented below, the dimension n of the vector x was fixed to 1011 and the parameter c varied between 1 and 500. Fig. 1 shows on the x-axis the value of the parameter c and on the y-axis $log_{10}\mathrm{wid}(s)$.

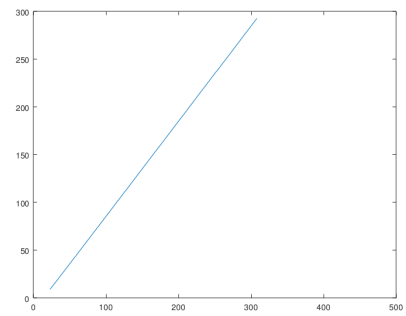

Fig. 1 Result of the interval sum of the vector $x(c)$ of dimension 1011: on the x-axis, the value of the parameter c, which corresponds to the number of decimal digits of the condition number; on the y-axis, the radix-10 logarithm of the width of the sum $\mathbf{s}$.

One can observe a perfect straight-line with slope 1: the width of the sum is multiplied by 10 when c increases by 1, as predicted by the theory. The difference between c and $\log_{10}\mathrm{wid}(\mathbf{s})$ is 16, which is the number of decimal digits of the double-precision floating-point numbers used in the computations. The curve stops at $c = 308$ as the width of s becomes infinite after that point. This corresponds to the limit of the range of floating-point numbers: 10^{308} can be represented by a bounded interval with floating-point endpoints, however 10^{309} overflows and thus the right endpoint of $\mathbf{x}_{309}$ is infinite. The sum thus becomes equal to $\mathbb{R}$, its width becomes infinite and the plotting command does not plot it.

To improve the numerical quality of a sum, a first heuristic consists in modifying the algorithm, and in this case in modifying the order in which the operands are summed, see [5, § 4.2, pp. 81–83]. We did not observe any improvement: this heuristic improves the result and not the condition number of the problem. A condition number corresponds to the worst case of propagation of errors and interval arithmetic also computes results which correspond to the worst case. Interval computations may thus be more closely correlated with the condition number than the summation with a well-chosen order. Another classical technique, that improves worst-case error analysis, is the so-called *compensated summation*, see [7, §6.3, pp.208-218]. It relies on the `TwoSum` routine that transforms two floating-point numbers x and y into a pair of floating-point numbers s and e such that $s+e = x+y$ exactly and $s = \mathrm{RN}(x+y)$ is the floating-point sum of x and y. Our Pichat-Neumaier-like version for the summation of intervals is given below in Octave syntax:

```
sH=0.0; sL=infsup(0.0);
for i=1:n,
  m=mid(x(i));
  [sH,tmp]=TwoSum(sH,m);
  sL=sL+tmp+(x(i)-m);
end;
```

The results are tighter than for the summation without compensation, as can be seen on Fig. 2: the width of the compensated sum, in red, is less than the width of the original sum, in blue: 2.5 decimal digits are gained through this technique.

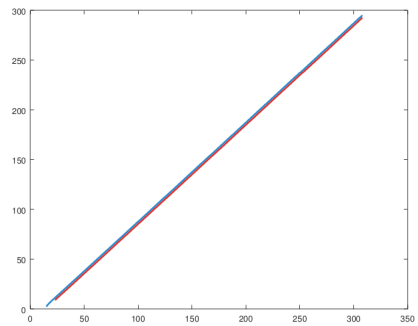

Fig. 2 Result of the interval sum of the vector $x(c)$ of dimension 1011: on the x-axis, the value of the parameter c, which corresponds to the number of decimal digits of the condition number, and on the y-axis, the radix-10 logarithm of the width of the sum: in blue, the sum as previously, in red, the compensated sum.

4 Solving Linear Systems

The second problem is the solution of a linear system. Every result presented in this Section is taken from Nguyen's PhD thesis [10]. Let A be a $n \times n$ real matrix and b a real n-vector, the problem is to solve $Ay = b$. It is well-known (see [5, Chapter 7] for an introduction and references) that the condition number of this problem with respect to perturbations of A is $\|A^{-1}\| \, \|A\|$.

For the numerical computations, the solution is obtained via the MATLAB command `x=A\b`. For interval computations, the employed algorithm is based on the classical iterative refinement technique, which is given below in a MATLAB-like syntax. For details about the following algorithm, see Wilkinson [13] for the original algorithm and Higham [5, Chapter 12] for its analysis and further references.

Algorithm: linear system solving using iterative refinement

Input: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$

$y = A \backslash b$ % in practice: factorization LU of A
% and solving of two triangular linear systems

```
while (not converged)
```

$r = b - Ay$
$e = A \backslash r$
$y = y + e$

Output: y

Regarding the interval computations: we consider A and b to be floating-point matrix and vector respectively, and $\mathbf{A}$ and $\mathbf{b}$ to be equal to A and b, with interval type to contaminate further computations. The algorithm to solve this system

with interval coefficients is given below. The first step, $y = A\backslash b$, is computed using floating-point arithmetic: the LU-factorization of A is done with floating-point arithmetic and if L and U are the factors of A, they are kept for subsequent computations. Interval arithmetic is used in the iterative refinement loop only.

Algorithm: linear system solving using iterative refinement, interval version

Input: $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$
$y = A\backslash b$
`while (not converged)`
  $\mathbf{r} = [b - Ay]$ % $b - Ay$ is computed using interval arithmetic
  $\mathbf{e} = A\backslash \mathbf{r}$ % $\mathbf{e}$ is computed using interval arithmetic
  $\mathbf{y} = \mathbf{y} + \mathbf{e}$
Output: $\mathbf{y}$

A difficulty is to solve $\mathbf{e} = A\backslash\mathbf{r}$. The LU factorization of A is used to prepare the system. One solves $U^{-1}.L^{-1}\mathbf{r} = U^{-1}.L^{-1}.A\mathbf{e}$. The underlying principle is that $U^{-1}.L^{-1}.A$ is close to the identity matrix, thus it is diagonally dominant, so that the Gauss-Seidel iterative method is contractant. However, for this contractant method to be applicable, one needs an initial enclosure of $\mathbf{e}$. Rump [12] was the first to offer a function, called `verifylss` in the IntLab library [12], implementing a method by Neumaier [9]: he gives a heuristic to determine an initial enclosure for $\mathbf{e}$. Nguyen proposes a different heuristic in [10] and the corresponding function is called `certifylss`. A third function, called `certifylss_relaxed`, implements some tricks to improve the execution time but it has no effect on the accuracy of the solution. The results of these functions are very similar, as can be seen on Fig. 3.

The matrix A is generated using MATLAB command `randsvd(n, cond)`, where `n` is the dimension and `cond` is the expected condition number for this matrix. The vector b is chosen as $A(1,1,\ldots 1)^t$. On the figure, the x-axis gives the value of `cond`, varying between 2^5 and 2^{50}, in radix-2 logarithmic scale. The y-axis indicates the number of correct bits of the solution, it corresponds to the maximal width of the components of the solution: $-\log_2 \max \mathrm{wid}(\mathbf{x}_i)$. The pink curve corresponds to MATLAB solution $A\backslash b$: MATLAB always returns an answer, however its accuracy decreases as the condition number increases. No iterative refinement is applied, otherwise the accuracy would be comparable to the next three curves. The three other curves correspond to Rump's `verifylss` and to Nguyen's `certifylss` and `certifylss_relaxed`. All three are able to compute accurately (thanks to iterative refinement) the solution for small condition numbers, up to 2^{37} in this experiment. Then the three methods return less and less accurate solutions as the condition number increases but remains moderate, up to 2^{45}, and then, for large condition numbers, they all fail to return an answer because their heuristics to determine an initial enclosure of the error fail.

Nguyen in [10] proposed several modifications to increase the accuracy of the result. Schematically, his algorithm is as follows:

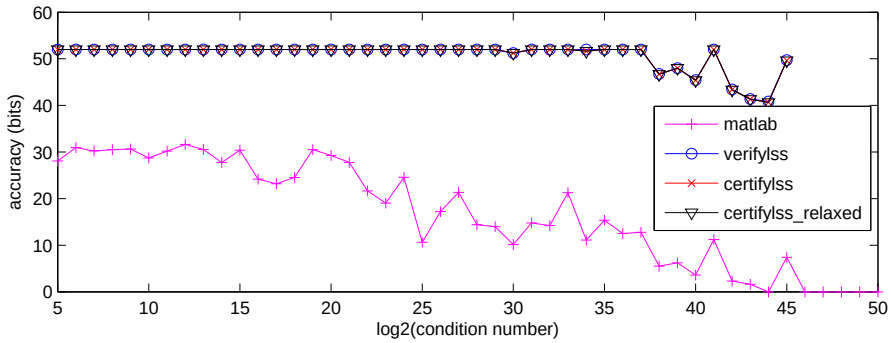

Fig. 3 Solution of a linear system: on the x-axis, the condition number with a radix-2 lgarithmic scale, on the y-axis, the accuracy in bits of the solution.

Algorithm: linear system solving using iterative refinement, interval version 2

Input: $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$

$y = A\backslash b$

modifications, including a floating-point matrix R and an interval matrix $\mathbf{K} \supset RA$

```
while (not converged)
```

$\mathbf{r} = [R(b - Ay)]$ % computed in doubled precision

$\mathbf{e} = \mathbf{K}\backslash\mathbf{r}$

$\mathbf{y} = \mathbf{y} + \mathbf{e}$ % computed in doubled precision

Output: $\mathbf{y}$

The first version is called `certifylssx` and reaches full precision for the problems it can solve. The second version is called `certifylssxs`: it uses $\mathbf{K}$ which enlarges RA, and thus it degrades the accuracy on $\mathbf{e}$. However, solving $\mathbf{e} = \mathbf{K}\backslash\mathbf{r}$ is faster than in the previous version and $\mathbf{y}$ remains as accurate, as shown on Fig. 4.

For the problem of solving a linear system, the impact of the condition number can be seen on the accuracy of the solution, but also on the fact that the methods fail to solve the linear system for large enough condition numbers. Nguyen also put in evidence the effect of the condition number on the execution time in [10]: as long as the method succeeds in computing an enclosure of the solution with full accuracy, the computing time increases with the condition number.

5 Univariate Nonlinear Equations

The last problem used in these experiments is the determination of the zeros of a nonlinear equation in one variable, using Newton method. Usually, the problem is introduced in the following form: determine z such that a given function F vanishes at z. As we want to vary the condition number of the problem, we need a parameter upon which the condition number depends. The problem considered in this Section

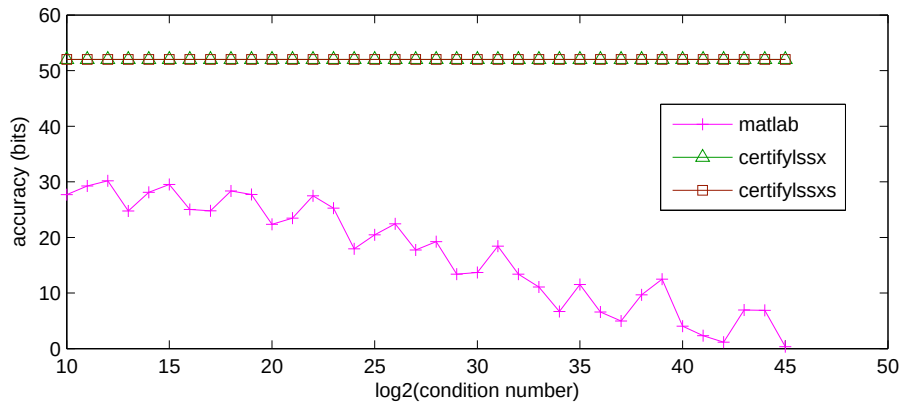

Fig. 4 Solution of a linear system: on the x-axis, the condition number with a radix-2 lgarithmic scale, on the y-axis, the accuracy in bits of the solution.

is thus: for a given d, determine $z = f(d)$ such that $F(f(d),d) = 0$, where

$$\begin{aligned} F &: \mathbb{R}^2 \to \mathbb{R} \\ (z,d) &\mapsto F(z,d). \end{aligned}$$

What is the condition number of this problem? Let us differentiate both sides of the equality $F(f(d),d) = 0$:

$$f'(d).\frac{\partial F}{\partial z}(f(d),d) + \frac{\partial F}{\partial d}(f(d),d) = 0,$$

and thus, if $\frac{\partial F}{\partial z}(f(d),d) \neq 0$, we get

$$f'(d) = -\frac{\frac{\partial F}{\partial d}(f(d),d)}{\frac{\partial F}{\partial z}(f(d),d)}.$$

If we replace $f'(d)$ by this expression in Eq. (2), one gets

$$c_f(d) = \left| \frac{\frac{\partial F}{\partial d}(f(d),d)}{\frac{\partial F}{\partial z}(f(d),d)} \right| . \frac{|d|}{|f(d)|}.$$

For our experiments, the chosen function F is $F(z,d) = \frac{\log d}{z} - 1$. For a given d^*, the corresponding zero is $z^* = \log d^*$. The condition number of this problem in d is determined as follows. Let us compute the partial derivatives of F:

$$\frac{\partial F}{\partial d}(z,d) = \frac{1}{dz} \text{ and } \frac{\partial F}{\partial d}(\log d,d) = \frac{1}{d \log d},$$

$$\frac{\partial F}{\partial z}(z,d) = \frac{-\log d}{z^2} \text{ and } \frac{\partial F}{\partial z}(\log d, d) = \frac{-1}{\log d},$$

thus

$$c_f(d^*) = \left| \frac{\frac{1}{d^* \log d^*}}{\frac{-1}{\log d^*}} \right| \cdot \frac{|d^*|}{|\log d^*|} = \frac{1}{|\log d^*|}.$$

When $d^* \to 1$, $c_f(d^*) \simeq \frac{1}{|d^*-1|} \to \infty$.

In the experiments, the solution is computed using the `fzero` routine of the `interval` package in Octave [4]. The initial interval, in which the zeros of the function are sought, are $[-100, (\log d)/2]$ when $d < 1$ and $[(\log d)/2, 100]$ when $d > 1$. One observes the proportionality, predicted by the theory, between the condition number and the accuracy of the enclosure. The non-monotonic behavior of the accuracy, or the "steps" that can be observed on the curve, corresponds to the different cases $d < 1$ and $d > 1$: the condition numbers are such that the results for $d = 1 - 10^{-i}$ and $d = 1 + 10^{-i}$ are interleaved. These two different cases are presented separately on Fig. 6.

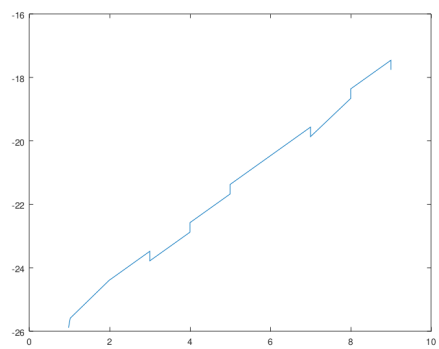

Fig. 5 Solution of the non-linear equation $F(z,d) = (\log d)/z - 1$ for varying d. The x-axis corresponds to the radix-10 logarithm of the condition number of the problem and the y-axis corresponds to the radiz-10 logarithm of the width of the enclosure of the zero.

Increasing the computing precision would certainly improve the accuracy of the sought zeros. However, in this case, one would need to resort to a dedicated library for increased precision, as operations and functions more elaborate than additions and multiplications need to be evaluated with a large precision. This makes it more cumbersome than in the previous experiments. Instead, we resorted to a simple but usually efficient technique, classically used in interval computations, which is a direct consequence of Eq. (4): bisection of the input intervals. Alas! we split the input interval in 50 subintervals of equal length and a (slight) gain in accuracy could be observed only for well-conditioned inputs. This is easily explained: splitting the input interval beforehand created many subintervals containing no zero and rapidly discarded. This was of no use to the algorithm, which is able to do so by construction. The same observation has been made for Branch-and-Bound algorithms for global optimization [11]: splitting the initial domain does not improve the search, as most initial subintervals are discarded very rapidly. Bisecting the search interval should

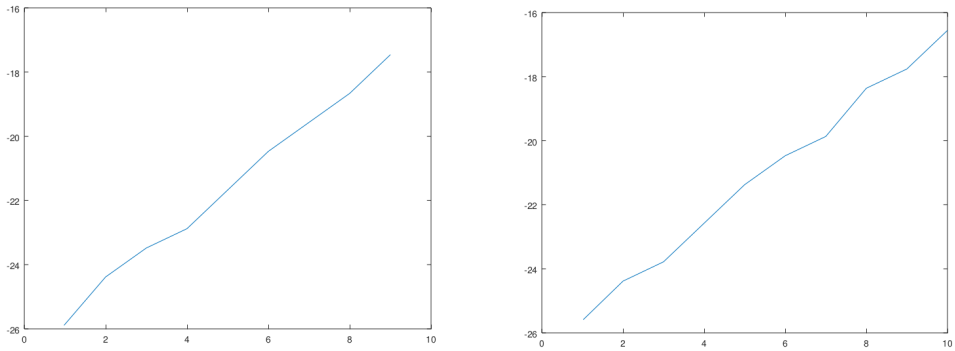

Fig. 6 Solution of the nonlinear equation $F(z,d) = (\log d)/z - 1$ for varying d. On the x-axis: radix-10 logarithm of the condition number of the problem, on the y-axis: radiz-10 logarithm of the width of the enclosure of the zero. On the left: $d < 1$, on the right: $d > 1$.

be done only by the algorithm itself (in case of Branch-and-Bound algorithms) or during the last steps (in case of Newton algorithm).

6 Conclusion and Future Work

The relation between the amplification factors of the errors for numerical and interval computations has been studied: both use the derivative of the computed function, however in the interval context, an interval evaluation – thus, with overestimation – of the derivative has to be used. The influence of the usual condition number on some interval computations has been observed. When the condition number is small, the computed result is accurate, in the sense that its width is small. This width increases as the condition number increases, unless more efforts are put in the computations to preserve the accuracy, at the expense of the computing time. When the condition number gets large, either the computed result is the whole set of real numbers, which conveys no useful information, or the method fails, which is the case for the linear or nonlinear system solving. As the fundamental theorem of interval arithmetic is sometimes called the *Thou shalt not lie* commandment, this means that interval computations remain silent about the result, instead of ”lying” and returning a incorrect result, that is, a result that does not contain the exact result.

Some possible solutions to obviate the impact of the condition number come to mind, some have been experimented. First, an increase of the computing precision usually yields an increase of the accuracy. How the computing precision should be increased has been dealt with in [6] for the general case: they recommend the choice of a precision that corresponds to doubling the execution time. For specific problems, thorough studies can lead to a more hand-tailored choice, where the precision is increased only for the most sensitive computations, as it has been observed

for the iterative refinement method, for solving linear systems in Section 4 and with a more detailed study in [10, 1, 2]. Another classical approach in interval algorithms is to bisect the input interval, so as to reduce the width of the output interval. In our experiments, bisection is useless if it is performed too early, except maybe on well-conditioned problems. Bisection should occur only when the algorithm has difficulties refining the output, but not too early during the computations. A more promising approach is the design of ad hoc algorithms, such as the iterative refinement of Section 4. It must however concentrate on the interval algorithm and not be a mere adaptation of existing techniques, such as the reordering of the operands for the summation.

Acknowledgements This work has been partially supported by the ANR project MetaLibm ANR-13-INSE-0007-04.
The author thanks H. D. Nguyen for his kind permission to reproduce the material of Section 4.

References

1. E. Carson and N.J. Higham. A New Analysis of Iterative Refinement and its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems. Technical Report MIMS Eprint 2017.12, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2017.
2. E. Carson and N.J. Higham. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. Technical Report MIMS Eprint 2017.24, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2017.
3. 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(2):325–351, 2006.
4. O. Heimlich. Interval arithmetic in GNU Octave. In *SWIM 2016: Summer Workshop on Interval Methods*, 2016.
5. N. Higham. *Accuracy and Stability of Numerical Algorithms (2nd edition)*. SIAM Press, 2002.
6. V. Kreinovich, and S. Rump. Towards optimal use of multi-precision arithmetic: a remark. *Reliable Computing*, 12(5):365–369, 2006.
7. J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Springer, 2009.
8. A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.
9. A. Neumaier. A simple derivation of the Hansen-Bliek-Rohn-Ning-Kearfott enclosure for linear interval equations. *Reliable Computing*, 5((+ Erratum: Reliable Computing 6, p227, 2000)):131–136, 1999.
10. H. D. Nguyen. *Efficient algorithms for verified scientific computing : Numerical linear algebra using interval arithmetic*. PhD thesis, École Normale Supérieure de Lyon - ENS LYON, 2011. `https://tel.archives-ouvertes.fr/tel-00680352/en`
11. N. Revol, and Y. Denneulin, and J.-F. Méhaut, and B. Planquelle. A methodology of parallelization for continuous verified global optimization. In LNCS 2328, pp. 803-801, 2002.
12. S. Rump. *Developments in Reliable Computing, T. Csendes ed.*, chapter INTLAB - Interval Laboratory, pages 77–104. Kluwer, 1999.
13. J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1963.