

The Influence of the Computing Precision on Interval Computations

Nathalie Revol

INRIA, Pascaline team - LIP, ENS de Lyon - France

`Nathalie.Revol@ens-lyon.fr`

Keywords: Computing Precision, Floating-point Arithmetic, Interval Arithmetic, Interval Computations, Trade-off Speed / Accuracy

Introduction

For numerical computations, the use of the minimal useful computing precision enables one to reduce both the computing time and the memory usage. Indeed, most processors offer single precision and double precision (floating-point numbers stored on 32 bits, resp. on 64 bits) and each operation on 32-bit operands is twice faster (or even faster) than the corresponding operation on 64-bit operands. The energy consumption, memory bandwidth as well as storage requirements follow the same trend. However, users do not want to sacrifice the numerical accuracy of the final results. The question is thus to choose the adequate computing precision for each part of the algorithm, in order to improve the performances without significantly degrading the numerical quality [4, 1]. Such algorithms are said to use "mixed-precision".

Interval arithmetic is a tool to obtain numerical guarantees, by providing enclosures of the computed results [7, 6, 12]. It can also benefit from a sound choice of the computing precision: we will survey the use of mixed-precision in interval arithmetic for matrix multiplication, certification of the solution of a linear system, and Newton method.

Intervals can be represented by their endpoints: $\mathbf{x} = [\underline{x}, \bar{x}] = \{x : \underline{x} \leq x \leq \bar{x}\}$ and the endpoints of the interval $\mathbf{x}$ are $\underline{x}$ and $\bar{x}$, or by the *mid-rad* representation, using the midpoint and the radius of the interval: $\mathbf{x} = \langle x_m, x_r \rangle$ with $x_r \geq 0$ is the interval $[x_m - x_r, x_m + x_r]$.

Matrix multiplication

In [11], Rump proposes a fast algorithm for interval matrix multiplication. It uses the mid-rad representation of the two matrices to be multiplied: $\mathbf{A} = \langle A_m, A_r \rangle$ and $\mathbf{B} = \langle B_m, B_r \rangle$, and it trades off accuracy for speed by computing a matrix $\mathbf{C} = \langle C_m, C_r \rangle$ such that $\mathbf{C} \supset \mathbf{A} \times \mathbf{B}$, but the overestimation can be up to 50%, that is, for each coefficient of $\mathbf{C}$, its radius can be 1.5 times the radius of the corresponding coefficient of $\mathbf{A} \times \mathbf{B}$.

It uses 3 floating-point matrix multiplications, which are highly optimized routines in many libraries, usually called `gemm`, and it is much faster than the use of the classical formulae.

Then several authors [8, 10] have proposed other algorithms that either use more floating-point matrix multiplications, that is, more calls to `gemm`, in order to reduce the overestimation, or less calls to `gemm` in order to reduce the computing time.

Some of these variants have been studied by Ozaki et al. [9] in order to reduce the computing precision during the computation of the radius matrix. Indeed, in the case of narrow intervals, the radius conveys only the error or the inaccuracy on the midpoint, and there is no use in keeping many digits for the radius. Thus Ozaki et al. suggest to perform all computations related to the radii in small precision. Their main contribution is to account for the roundoff errors incurred by this reduction of precision, in order to preserve the inclusion property.

Certified solutions of linear systems

Another direction of research is the certification of the solution of a (either floating-point, or extremely tiny) linear system. The floating-point linear system is $Ax = b$, with A and b are of dimensions respectively $n \times n$ and n and have floating-point coefficients. We assume that the linear system is solved using a LU factorization of A and that $\hat{x}$ is an approximate solution. Let us denote by x^* the exact solution, that is, $Ax^* = b$ in $\mathbb{R}^n$. An iterative refinement approach is used.

In [8], the proposed algorithm uses a reduced computing preci-

sion for every time-consuming parts, namely the LU factorization and then the linear system solving using LU (which amounts to solve two linear triangular systems), that require respectively $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ floating-point operations. This algorithm uses the largest computing precision only for the numerically sensitive parts, namely the computation of the residual $r = b - A\hat{x}$ and the correction of $\hat{x}$ as $\hat{x} + \text{mid}(\mathbf{e})$, where $\mathbf{e}$ is the current enclosure of the error: these parts require only $\mathcal{O}(n^2)$ floating-point operations. The parts computed in higher precision thus take a negligible fraction of the whole computing time, which is dominated by the LU factorization, unless many iterations are required in order to reach the prescribed accuracy.

Newton method for univariate equations

If we now consider Newton method to solve a nonlinear univariate equation, we go even further: each iteration is a refinement of the current zero, and here we know that (except in the case of multiple roots, or otherwise ill-conditioned zeros) the convergence is quadratic. This means that at each step, the number of correct bits roughly doubles. It is thus easy to choose the computing precision: use a small one for the first iterations, until it limits the number of correct bits that can be reached, then increase the computing precision [5].

Conclusion

The following Julia libraries: `IntervalArithmetic.jl` [2], along with `IntervalLinearAlgebra.jl` [3], make it relatively easy to experiment the algorithms mentioned above.

References

- [1] A. Abdelfattah et al. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The Int. J. of High Performance Computing Applications*, 35(4):344–369, 2021.

- [2] L. Benet, O. Hénot, B. Richard, and D.P. Sanders. IntervalArithmetic.jl, 2025.
- [3] L. Ferranti, M. Forets, and D.P. Sanders. IntervalLinearAlgebra.jl: linear algebra done rigorously, 2021.
- [4] N.J. Higham and T. Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.
- [5] P. Langlois and N. Revol. Validating polynomial numerical computations with complementary automatic methods. *Mathematics and Computers in Simulation*, minor revision, 2001.
- [6] R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.
- [7] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.
- [8] H.D. Nguyen. *Efficient algorithms for verified scientific computing: numerical linear algebra using interval arithmetic*. PhD thesis, École Normale Supérieure de Lyon, France, 2011.
- [9] K. Ozaki, T. Ogita, F. Bünger, and S. Oishi. Accelerating interval matrix multiplication by mixed precision arithmetic. *Nonlinear Theory and Its Applications, IEICE*, 6(3):364–376, 2015.
- [10] N. Revol and P. Théveny. Parallel Implementation of Interval Matrix Multiplication. *Reliable Computing*, 19(1):91–106, 2013.
- [11] S.M. Rump. Fast interval matrix multiplication. *Numerical Algorithms*, 61(1):1–34, 2012.
- [12] W. Tucker. *Validated Numerics – A Short Introduction to Rigorous Computations*. Princeton University Press, 2011.