Ball arithmetic[*]

JORIS VAN DER HOEVEN

LIX, CNRS
École polytechnique
91128 Palaiseau Cedex
France

Email: `vdhoeven@lix.polytechnique.fr`
Web: `http://lix.polytechnique.fr/~vdhoeven`

First publication: November 13, 2009

Last modified: April 2, 2011

Latest corrections: October 6, 2018

The MATHEMAGIX project aims at the development of a "computer analysis" system, in which numerical computations can be done in a mathematically sound manner. A major challenge for such systems is to conceive algorithms which are both efficient, reliable and available at any working precision. In this paper, we survey several classical and newly developed methods for designing such algorithms. We mainly concentrate on the automatic and efficient computation of high quality error bounds. This is done using ball arithmetic, which is also known under the name midpoint-radius interval arithmetic.

KEYWORDS: Ball arithmetic, interval arithmetic, reliable computing, computable analysis

A.M.S. SUBJECT CLASSIFICATION: 65G20, 03F60, 65F99, 37-04

1. INTRODUCTION

1.1. Motivation

Computer algebra systems are widely used today in order to perform mathematically correct computations with objects of algebraic or combinatorial nature. It is tempting to develop a similar system for analysis.

Typical problems of this nature are the numerical resolution of a system of equations or the integration of a dynamical system with known initial conditions. In a "computer analysis" system, it should be possible to solve such problems in an exact way. In particular, it should be possible to compute arbitrarily close approximations for all numbers occurring in the input and the output. Computable analysis [Wei00, BB85, Abe80, Tur36] provides a suitable logical framework for computing with this kind of "computable numbers".

Assume that we require an approximation $\tilde{y} \in \mathbb{Q}$ of a number $y = f(x) \in \mathbb{R}$ with $|\tilde{y} - y| \leqslant \varepsilon$, where $\varepsilon \in \mathbb{Q}^{>} = \{a \in \mathbb{Q} : a > 0\}$ is provided by the user. A typical strategy in order to compute such an approximation is to evaluate $y_p \approx f(x)$ numerically at a fixed bit precision p, while computing a rigorous upper bound δ_p for the error $|y_p - f(x)|$. As soon as $\delta_p \leqslant \varepsilon$, then we are done. Otherwise, we keep repeating the same procedure at higher and higher precisions. Interval analysis constitutes a classical theory for the computation

[*]. This work has been supported by the ANR-09-JCJC-0098-01 MAGIX project, as well as a Digiteo 2009-36HD grant and Région Ile-de-France.

of such rigorous upper bounds [Moo66, AH83, Neu90, JKDW01, Kul08, MKC09, Rum10]. The website [Kreb] provides a wealth of information about the interval analysis community.

The preceding discussion imposes some minimal requirements on the system we are striving for: it should contain at least three layers for multiple precision arithmetic, interval analysis and computable analysis. It is natural to add another layer, or at least an interface, for classical numerical analysis. Taken on its own, each of these layers is well understood. However, it is also well known that multiple precision operations are at least one order of magnitude slower than hardware operations on single or double precision numbers. More generally, the naive combination of known techniques leads to highly inefficient algorithms, as we will see in section 5 on the hand of a problem as simple as matrix multiplication.

In order to design a reasonably efficient system for computer analysis, we therefore need to rethink many of the algorithms from the individual layers in a more integrated way. Moreover, in a reasonably complete system, this rethinking should be performed in a very systematic way for the most basic mathematical objects: besides operations on numbers, we need efficient, numerically stable and reliable algorithms for standard operations on matrices, polynomials, series and analytic functions.

In this paper, we will focus on the single layer of interval analysis (with the short exception of section 5). The first part of the paper provides a survey of well known techniques, but from our perspective of usefulness for the design of a computer analysis system. In particular, it turns out that ball enclosures are usually more convenient than interval enclosures in our context. This is partly due to the fact that multiple precision enclosures are usually very narrow. In the second part of the paper, we provide a more detailed study of basic operations on non scalar objects, such as matrices and polynomials. To the best of our knowledge, several of the algorithms in the second part are new.

Although ball arithmetic is often referred to as "midpoint-radius interval analysis", we prefer our naming for three main reasons. First, since we will be using balls all around, it is nice to have a short name. Secondly, complex balls are not intervals. More generally, there is a considerable latitude concerning the choice of a center type and a radius type. Finally, balls are standard in mathematics, where they correspond to classical δ-ε calculus.

1.2. Existing software

There are several specialized systems and libraries which contain work into the direction of system for computer analysis. A fairly complete list of software for interval computations is maintained at [Krea]. For instance, Taylor models have been used with success for the validated long term integration of dynamical systems [Ber98, MB96, MB04]. A fairly complete Matlab interval arithmetic library for linear algebra and polynomial computations is Intlab [Rum99b, Rum99a]. Another historical interval library, which continues to be developed is [ea67].

There exist several libraries for multiple precision arithmetic with correct or at least well specified rounding [HLRZ00, Hai95], as well as a library for multiple precision interval arithmetic [Rev01], and libraries for computable real numbers [M0, Lam07, BCC+06]. There are also libraries for very specific problems. For instance, the MPSolve library [BF00] allows for the certified computation of the roots of multiple precision polynomials. Similarly, PHCpack [Ver99] and Bertini [BHSW06] are systems for polynomial system solving *via* numerical homotopy methods.

Unfortunately, the mere agglomeration of existing software is insufficient for building an efficient general purpose system for computer analysis. Besides licensing issues and technical interfacing problems, one of the main reasons is that the reliable computation community has a traditional focus on limited precision computations. Even though this situation is evolving, we have already mentioned the fact that the mere replacement of machine arithmetic by arbitrary precision arithmetic induces a huge performance penalty.

Software in which this efficiency problem is addressed (libraries for computable real numbers, MPSolve, etc.) usually focusses on numbers or a very specific problem.

```
Mmx] use "analyziz"
Mmx] bit_precision := 128;
Mmx] z == series (ball 0.0, ball 1.0);
Mmx] B == exp (exp z - 1)

     1.000000000000000000000000000000000000 + 1.00000000000000000000000000000000\
     0000 z + 1.000000000000000000000000000000000000 z^2 + 0.83333333333333333333333333333\
     3333333 z^3 + 0.625000000000000000000000000000000000 z^4 + 0.433333333333333333333333333\
     33333333333 z^5 + 0.281944444444444444444444444444444444 z^6 + 0.174007936507936507936\
     507936507936508 z^7 + 0.102678571428571428571428571428571429 z^8 + 0.0582754629629629\
     6296296296296296296296296296296296296296296296296296296296296296296296296296296962962963 z^9 + O(z^10)

Mmx] B[10000] * 10000!

     1.59217229255742103113048135619321932e27664
                                                                              7.954 sec
```

Figure 1. A sample Mathemagix session for the reliable computation of the Bell number B_{1000}.

These considerations have motivated the development of Mathemagix [vdHLM+02], a new system for computer analysis, which is distributed under a free licence and does not depending on proprietary software such as Matlab. On the one hand, Mathemagix provides a wide variety of numerical types, such as machine doubles, multiple precision numbers, complex numbers, tangent numbers, intervals, balls, computable numbers, but also integers, rationals, algebraic numbers, modular integers, etc. On the other hand, we implemented standard mathematical containers, such as matrices, polynomials and power series. We systematically attempt to make these implementations as efficient as possible for the various kinds of numerical types. For instance, the algorithm for matrix multiplication will be very different for, say, rational matrices and complex ball matrices.

The Mathemagix system has been designed in four layers, with multiple precision arithmetic and classical numeric methods at the bottom, and the certification of numerical results and computable analysis at the top. Although we briefly discuss this general architecture in section 5, this paper is mainly devoted to the automatic computation of error bounds *via* ball arithmetic. We will outline both the mathematical and implementation techniques which are used in order to make this certification layer as efficient as possible. With the exception of some ideas in section 7.5, all algorithms in this paper have been implemented in our system.

Although we will focus on ball arithmetic, it should be noticed that high efficiency of the bottom layers is crucial in order to achieve high performance. Actually, one important design principle is to delay certification methods to the very end, so that the bulk of the execution time is spent on the actual numerical computations and not on their certification. Now there are many books on numerical analysis at limited precision [PTVF07, SB02]. The systematic development of multiple precision algorithms which are both efficient and numerically stable is still a major challenge. Since switching from machine precision to multiple precision involves a huge overhead, another challenge is to rely as much as possible on algorithms which first compute an approximation at low precision and then adjust the result at high precision. Inside Mathemagix, we are helped by the fact that most current packages are C++ template libraries, which can be instantiated for numbers at single, double, quadruple and multiple precision (see [BHL00, BHL01] for a double-double and quadruple-double precision library).

In figure 1, we have illustrated the new algorithms available in Mathemagix on a simple example from combinatorics: the reliable approximation of Bell numbers B_n in limited precision (128 bits in our case). Although one might use an asymptotic formula for this particularly well known example, we have preferred to compute B_n from the exponential generating series $B(z) = \sum_{n \in \mathbb{N}} B_n \, z^n / n! = \mathrm{e}^{\mathrm{e}^z - 1}$. Indeed, this very general technique can also be applied in many other cases and has applications to the random generation of combinatorial objects [FZVC94]. In figure 1, each floating point number in the output is certified to be correct modulo an error of the order of its last digit.

1.3. Overview

Ball arithmetic, also known as midpoint-radius interval arithmetic, provides a systematic tool for the automatic computation of error bounds. In section 3, we provide a short introduction to this topic and describe its relation to computable analysis. In particular, we recall how to compute with approximate real and complex numbers. Although ball arithmetic is really a variant of interval arithmetic, we will give several arguments in section 4 why we actually prefer balls over intervals for most of our applications. Sections 2, 3 and 4 mainly provide a survey of well known results in the reliable computing community. Only our perspective might be slightly less classical. Some of the considerations in section 3.5 also seem to be less well known.

In sections 6 and 7, we turn our attention to more complicated mathematical objects, such as matrices, polynomials, power series and analytic functions. Although some of the techniques are still well known (such as Hansen's method for the inversion of matrices), many of the algorithms presented in these sections seem to be new or expand on earlier work by the same author [vdH07b]. Usually, there is a trade-off between efficiency and the quality of the obtained error bounds. One may often start with a very efficient algorithm which only computes rough bounds, or bounds which are only good in favourable well-conditioned cases. If the obtained bounds are not good enough, then we may switch to a more expensive and higher quality algorithm.

The main algorithmic challenge in the area of ball arithmetic is to reduce the overhead of the bound computations as much as possible with respect to the principal numeric computation. In favourable cases, this overhead indeed becomes negligible. For instance, for high working precisions p, the centers of real and complex balls are stored with the full precision, but we only need a small precision for the radii. Consequently, the cost of elementary operations ($+$, $-$, $\cdot$, exp, etc.) on balls is dominated by the cost of the corresponding operations on their centers. Similarly, crude error bounds for products of large matrices can be obtained quickly with respect to the actual product computation, by using norm bounds for the rows and columns of the multiplicands; see (22).

Another algorithmic challenge is to use fast algorithms for the actual numerical computations on the centers of the balls. In particular, it is important to use existing high performance libraries, such as Blas, Linpack, etc., whenever this is possible [Rum99a]. Similarly, we should systematically rely on asymptotically efficient algorithms for basic arithmetic, such as fast integer and polynomial multiplication [KO63, CT65, SS71]. There are several techniques to achieve this goal:

1. The representations of objects should be chosen with care. For instance, should we rather work with ball matrices or matricial balls (see sections 6.4 and 7.1)?

2. If the result of our computation satisfies an equation, then we may first solve the equation numerically and only perform the error analysis at the end. In the case of matrix inversion, this method is known as Hansen's method; see section 6.2. It should also be noticed that Newton type iterations which are used in this kind of methods are also useful for doubling the accuracy in multiple precision contexts.

3. When considering a computation as a tree or as a dag, then the error tends to increase with the depth of the tree. If possible, algorithms should be designed so as to keep this depth small. Examples will be given in sections 6.4 and 7.5. Notice that this kind of algorithms are usually also more suitable for parallelization.

When combining the above approaches to the series of problems considered in this paper, we are usually able to achieve a constant overhead for sharp error bound computations. In favourable cases, the overhead becomes negligible. For particularly ill conditioned problems, we need a logarithmic overhead. It remains an open question whether we have been lucky or whether this is a general pattern.

In this paper, we will be easygoing on what is meant by "sharp error bound". Regarding algorithms as functions $f\colon \mathbb{R}^k \rightarrow \mathbb{R}^l$; $x \mapsto y = f(x)$, an error $\delta_x \in \mathbb{R}^k$ in the input automatically gives rise to an error $\delta_y \approx J_f(x)\,\delta_x$ in the output. When performing our computations with bit precision p, we have to consider that the input error δ_x is as least of the order of $2^{-p}\,|x| \in (\mathbb{R}^{\geqslant})^k$ (where $|x|_i = |x_i|$ for all i). Now given an error bound $|\delta_x| \leqslant \varepsilon_x$ for the input, an error bound $|\delta_y| \leqslant \varepsilon_y$ is considered to be sharp if $\varepsilon_y \approx \max_{|\delta_x| \leqslant \varepsilon_x} |J_f(x)\,\delta_x|$. More generally, condition numbers provide a similar device for measuring the quality of error bounds. A detailed investigation of the quality of the algorithms presented in this paper remains to be carried out. Notice also that the computation of optimal error bounds is usually NP-hard [KLRK97].

2. Notations and classical facts

2.1. Floating point numbers and the IEEE 754 norm

We will denote by

$$\mathbb{D} = \mathbb{Z}\,2^{\mathbb{Z}} = \{m\,2^e : m, e \in \mathbb{Z}\}$$

the set of *dyadic numbers*. Given fixed bit precisions $p \in \mathbb{N}$ and $q \in \mathbb{N}$ for the unsigned mantissa and signed exponent, we also denote by $\mathbb{D}_{p,q}$ the corresponding set of floating point numbers. Numbers in $\mathbb{D}_{p,q}$ can be stored in $p+q+1$ bit space (the extra bit being used for the sign) and correspond to "ordinary numbers" in the IEEE 754 norm [ANS08, Mul06]. For double precision numbers, we have $p = 51$ and $q = 12$. For multiple precision numbers, we usually take q to be the size of a machine word, say $q = 64$, and denote $\mathbb{D}_p = \mathbb{D}_{p,64}$.

The IEEE 754 norm also defines several special numbers. The most important ones are the infinities $\pm\infty$ and "not a number" NaN, which corresponds to the result of an invalid operation, such as $\sqrt{-2}$. We will denote

$$\mathbb{D}_{p,q}^* \;=\; \mathbb{D}_{p,q} \cup \{-\infty, +\infty, \mathrm{NaN}\}.$$

Less important details concern the specification of signed zeros ± 0 and several possible types of NaNs. For more details, we refer to [ANS08].

The other main feature of the IEEE 754 is that it specifies how to round results of operations which cannot be performed exactly. There are four basic rounding modes $\uparrow$ (upwards), $\downarrow$ (downwards), $\updownarrow$ (nearest) and 0 (towards zero). Assume that we have an operation $f\colon U \rightarrow \mathbb{R}$ with $U \subseteq \mathbb{R}^k$. Given $x \in U \cap \mathbb{D}_{p,q}^k$ and $y = f(x)$, we define $f^{\uparrow}(x)$ to be the smallest number $\tilde{y}$ in $\mathbb{D}_{p,q}^*$, such that $\tilde{y} \geqslant y$. More generally, we will use the notation $(f(x) + y)^{\uparrow} = f^{\uparrow}(x) +^{\uparrow} y$ to indicate that all operations are performed by rounding upwards. The other rounding modes are specified in a similar way. One major advantage of IEEE 754 arithmetic is that it completely specifies how basic operations are done, making numerical programs behave exactly in the same way on different architectures. Most current microprocessors implement the IEEE 754 norm for single and double precision numbers. A conforming multiple precision implementation also exists [HLRZ00].

2.2. Classical complexity results

Using Schönhage-Strassen multiplication [SS71], it is classical that the product of two p-bit integers can be computed in time $\mathsf{I}(n)=O(n\log n\log\log n)$. A recent algorithm by Fürer [Für07] further improves this bound to $\mathsf{I}(n)=O(n\log n\, 2^{O(\log^* n)})$, where $\log^*$ denotes the iterator of the logarithm (we have $\log^* n=\log^*\log n+1$). Other algorithms, such as the division of two $\leqslant n$-bit integers, have a similar complexity $O(\mathsf{I}(n))$. Given k prime numbers $p_1<\cdots<p_k$ and a number $0\leqslant q<p_1\cdots p_k$, the computation of $q \bmod p_i$ for $i=1,\ldots,k$ can be done in time $O(\mathsf{I}(n)\log n)$ using binary splitting [GG02, Theorem 10.25], where $n=\log(p_1\cdots p_k)$. It is also possible to reconstruct q from the remainders $q \bmod p_i$ in time $O(\mathsf{I}(n)\log n)$.

Let $\mathbb{K}$ be an effective ring, in the sense that we have algorithms for performing the ring operations in $\mathbb{K}$. If $\mathbb{K}$ admits 2^p-th roots of unity for $2^p\geqslant n$, then it is classical [CT65] that the product of two polynomials $P,Q\in\mathbb{K}[z]$ with $\deg P Q<n$ can be computed using $O(n\log n)$ ring operations in $\mathbb{K}$, using the fast Fourier transform. For general rings, this product can be computed using $\mathsf{M}(n)=O(n\log n\log\log n)$ operations [CK91], using a variant of Schönhage-Strassen multiplication.

A formal power series $f=f_0+f_1 z+f_2 z^2+\cdots\in\mathbb{K}[[z]]$ is said to be *computable*, if there exists an algorithm which takes $n\in\mathbb{N}$ on input and which computes $f_n\in\mathbb{K}$. We denote by $\mathbb{K}[[z]]^{\mathrm{com}}$ the set of computable power series. Many simple operations on formal power series, such as multiplication, division, exponentiation, etc., as well as the resolution of algebraic differential equations can be done modulo $O(z^n)$ using a similar time complexity $O(\mathsf{M}(n))$ as polynomial multiplication [BK78, BCO+06, vdH10]. Alternative *relaxed* multiplication algorithms of time complexity $\mathsf{R}(n)=O(\mathsf{M}(n))$ are given in [vdH02, vdH07a]. In this case, the coefficients $(f g)_n$ are computed gradually and $(f g)_n$ is output as soon as $f_0,\ldots,f_n$ and $g_0,\ldots,g_n$ are known. This strategy is often most efficient for the resolution of implicit equations.

Let $\mathbb{K}^{m\times n}$ denote the set of $m\times n$ matrices with entries in $\mathbb{K}$. Given two $n\times n$ matrices $M,N\in\mathbb{K}^{n\times n}$, the naive algorithm for computing $M N$ requires $O(n^\omega)$ ring operations with $\omega=3$. The exponent ω has been reduced by Strassen [Str69] to $\omega=\log 7/\log 2$. Using more sophisticated techniques, ω can be further reduced to $\omega<2.376$ [Pan84, CW87]. However, current machine implementations of matrix multiplication are far from reaching this asymptotic complexity.

2.3. Notations

Intervals and balls. Given $x<y$ in a totally ordered set R, we will denote by $[x,y]$ the closed interval

$$[x,y] \;=\; \{z\in R: x\leqslant z\leqslant y\}. \tag{1}$$

Assume now that R is a totally ordered field and consider a normed vector space V over R. Given $c\in V$ and $r\in R^{\geqslant}=\{r\in\mathbb{R}: r\geqslant 0\}$, we will denote by

$$\mathcal{B}(c,r) \;=\; \{x\in V: \|x-c\|\leqslant r\} \tag{2}$$

the closed ball with center c and radius r. Notice that we will never work with open balls in what follows. We denote the set of all balls of the form (2) by $\mathcal{B}(V,R)$. Given $X\in\mathcal{B}(V,R)$, we will denote by X_c and X_r the center resp. radius of x.

Linear algebra. We will use the standard Euclidean norm

$$\|x\| = \sqrt{\frac{|x_1|^2 + \cdots + |x_n|^2}{n}}$$

as the default norm on $\mathbb{R}^n$ and $\mathbb{C}^n$. Occasionally, we will also consider the max-norm

$$\|x\|_{\infty} = \max\{|x_1|, \ldots, |x_n|\}.$$

For matrices $M \in \mathbb{R}^{m \times n}$ or $M \in \mathbb{C}^{m \times n}$, the default operator norm is defined by

$$\|M\| = \max_{\|x\|=1} \|M x\|. \tag{3}$$

Occasionally, we will also consider the max-norm

$$\|M\|_{\infty} = \max_{i,j} |M_{i,j}|,$$

which satisfies

$$\|M\|_{\infty} \leqslant \|M\| \leqslant n \, \|M\|_{\infty}.$$

Polynomials and series. Given a polynomial $P \in \mathbb{R}[x]$ or $P \in \mathbb{C}[x]$, we define its max-norm by

$$\|P\|_{\infty} = \max |P_i|.$$

For power series $f \in \mathbb{R}[[z]]$ or $f \in \mathbb{C}[[z]]$ which converge on the compact disk $\mathcal{B}(0, r)$, we define the norm

$$\|f\|_r = \max_{|z| \leqslant r} |f(z)|. \tag{4}$$

After rescaling $f(z) \mapsto f(r\,z)$, we will usually work with the norm $\|\cdot\| = \|\cdot\|_1$, which should not be confused with the L_1 norm on the coefficients of f.

Generalized radii. In some cases, it is useful to generalize the concept of a ball and allow for radii in partially ordered rings. For instance, given two vectors $X_c \in \mathbb{R}^n$ and $X_r \in (\mathbb{R}^{\geqslant})^n$, we define $X = \mathcal{B}(X_c, X_r)$ by

$$X = \{x \in \mathbb{R}^n | \forall 1 \leqslant i \leqslant n, |x_i - (X_c)_i| \leqslant (X_r)_i\}.$$

We denote by $\mathcal{B}(\mathbb{R}^n, \mathbb{R}^n)$ the set of all such "balls". The sets $\mathcal{B}(\mathbb{R}^{m \times n}, \mathbb{R}^{m \times n})$, $\mathcal{B}(\mathbb{R}[x], \mathbb{R}[x])$, $\mathcal{B}(\mathbb{R}[[x]], \mathbb{R}[[x]])$, etc. are defined in a similar componentwise way. Given $x \in \mathbb{R}^n$, it will also be convenient to denote by $|x|$ the vector with $|x|_i = |x_i|$. For matrices $M \in \mathbb{R}^{m \times n}$, polynomials $P \in \mathbb{R}[x]$ and power series $f \in \mathbb{R}[[x]]$, we define $|M|$, $|P|$ and $|f|$ similarly.

3. BALLS AND COMPUTABLE REAL NUMBERS

3.1. Ball arithmetic

Let $\mathbb{V}$ be a normed vector space over $\mathbb{R}$ and recall that $\mathcal{B}(\mathbb{V}, \mathbb{R})$ stands for the set of all balls with centers in $\mathbb{V}$ and radii in $\mathbb{R}$. Given an operation φ: $\mathbb{V}^k \to \mathbb{V}$, the operation is said to lift to an operation

$$f^{\circ}: \mathcal{B}(\mathbb{V}, \mathbb{R})^k \to \mathcal{B}(\mathbb{V}, \mathbb{R}),$$

if we have

$$\{f(x_1, \ldots, x_k) : x_1 \in X_1, \ldots, x_k \in X_k\} \;\subseteq\; f^{\circ}(X_1, \ldots, X_k),$$

for all $X_1, \ldots, X_k \in \mathcal{B}(\mathbb{V}, \mathbb{R})$. For instance, both the addition $+: \mathbb{V}^2 \rightarrow \mathbb{V}$ and subtraction $-: \mathbb{V}^2 \rightarrow \mathbb{V}$ admits lifts

$$\begin{aligned} \mathcal{B}(x, r) +^{\circ} \mathcal{B}(y, s) &= \mathcal{B}(x + y, r + s) \qquad (5)\\ \mathcal{B}(x, r) -^{\circ} \mathcal{B}(y, s) &= \mathcal{B}(x - y, r + s). \qquad (6) \end{aligned}$$

Similarly, if $\mathbb{V}$ is a normed algebra, then the multiplication lifts to

$$\mathcal{B}(x, r) \cdot^{\circ} \mathcal{B}(y, s) \;=\; \mathcal{B}(x\, y, (|x| + r)\, s + r\, |y|) \qquad (7)$$

The lifts $+^{\circ}$, $-^{\circ}$, $\cdot^{\circ}$, etc. are also said to be the ball arithmetic counterparts of addition, subtractions, multiplication, etc.

Although ball arithmetic is really a variant of interval arithmetic, the use of ball enclosures has several advantages, which will be discussed in section 4. Both ball arithmetic and interval arithmetic provide a systematic way for the computation of error bounds when the input of a numerical operation is known only approximately. The bounds are usually not sharp. For instance, consider the mathematical function $f: x \in \mathbb{R} \mapsto x - x$ which evaluates to zero everywhere, even if x is only approximately known. Taking $x = \mathcal{B}(2, 0.001)$, we have $x -^{\circ} x = \mathcal{B}(0, 0.002) \neq \mathcal{B}(0, 0)$. This phenomenon is known as *overestimation*. In general, ball algorithms have to be designed carefully so as to limit overestimation.

In the above definitions, $\mathbb{R}$ can be replaced by a subfield $R \subseteq \mathbb{R}$, $\mathbb{V}$ by an R-vector space $V \subseteq \mathbb{V}$, and the domain of f by an open subset of $\mathbb{V}^k$. If V and R are *effective* in the sense that we have algorithms for the additions, subtractions and multiplications in V and R, then basic ball arithmetic in $\mathcal{B}(V, R)$ is again effective. If we are working with finite precision floating point numbers in $\mathbb{D}_{p,q}$ rather than a genuine effective subfield R, we will now show how to adapt the formulas (5), (6) and (7) in order to take into account rounding errors; it may also be necessary to allow for an infinite radius in this case.

3.2. Finite precision ball arithmetic

Let us detail what needs to be changed when using IEEE conform finite precision arithmetic, say $V = R = \mathbb{D}_{p,q}$. We will denote

$$\begin{aligned} \mathbb{B} &= \mathcal{B}(\mathbb{D}, \mathbb{D})\\ \mathbb{B}_{p,q} &= \mathcal{B}(\mathbb{D}_{p,q}, \mathbb{D}_{p,q})\\ \mathbb{B}[\mathrm{i}]_{p,q} &= \mathcal{B}(\mathbb{D}_{p,q}[\mathrm{i}], \mathbb{D}_{p,q}). \end{aligned}$$

When working with multiple precision numbers, it usually suffices to use low precision numbers for the radius type. Recalling that $\mathbb{D}_p = \mathbb{D}_{p,64}$, we will therefore denote

$$\begin{aligned} \mathbb{B}_p &= \mathcal{B}(\mathbb{D}_p, \mathbb{D}_{64})\\ \mathbb{B}[\mathrm{i}]_p &= \mathcal{B}(\mathbb{D}_p[\mathrm{i}], \mathbb{D}_{64}). \end{aligned}$$

We will write $\epsilon = \epsilon_p = 2^{-p}$ for the machine accuracy and $\eta = \eta_{p,q} = 2^{2-2^q}$ for the smallest normal positive number in $\mathbb{D}_{p,q}$ (we might also take $\eta = 2^{2-2^q-p}$ to be the smallest strictly positive subnormal number, but this choice deteriorates performance on architectures where computations with subnormal numbers are penalized).

Given an operation $f: \mathbb{R}^k \rightarrow \mathbb{R}$ as in section 3.1, together with balls $X_i = \mathcal{B}(x_i, r_i)$, it is natural to compute the center y of

$$\mathcal{B}(y, s) \;=\; f^{\circ}(X_1, \ldots, X_s)$$

by rounding to the nearest:

$$y \;=\; f^{\updownarrow}(x_1, \ldots, x_k). \tag{8}$$

One interesting point is that the committed error

$$\delta \;=\; |y - f(x_1, \ldots, x_k)|$$

does not really depend on the operation f itself: we have the universal upper bound

$$\begin{aligned} \delta \;&\leqslant\; \Delta(y) \\ \Delta(y) \;&:=\; (|y| +^{\uparrow} \eta) \cdot^{\uparrow} \epsilon. \end{aligned} \tag{9}$$

It would be useful if this *adjustment function* Δ were present in the hardware.

For the computation of the radius s, it now suffices to use the sum of $\Delta(y)$ and the theoretical bound formulas for the infinite precision case. For instance,

$$\mathcal{B}(x, r) +^{\circ} \mathcal{B}(y, s) \;=\; \mathcal{B}^{\Delta}(x +^{\updownarrow} y, r +^{\uparrow} s) \tag{10}$$

$$\mathcal{B}(x, r) -^{\circ} \mathcal{B}(y, s) \;=\; \mathcal{B}^{\Delta}(x -^{\updownarrow} y, r +^{\uparrow} s) \tag{11}$$

$$\mathcal{B}(x, r) \cdot^{\circ} \mathcal{B}(y, s) \;=\; \mathcal{B}^{\Delta}(x \cdot^{\updownarrow} y, [(|x| + r)\, s + r\, |y|]^{\uparrow}), \tag{12}$$

where $\mathcal{B}^{\Delta}$ stands for the “adjusted constructor”

$$\mathcal{B}^{\Delta}(y, s) \;=\; \mathcal{B}(y, s +^{\uparrow} \Delta(y)).$$

The approach readily generalizes to other “normed vector spaces” $\mathbb{V}$ over $\mathbb{D}_{p,q}$, as soon as one has a suitable rounded arithmetic in $\mathbb{V}$ and a suitable adjustment function Δ attached to it.

Notice that $\Delta(y) = \infty$, if $y = \infty$ or y is the largest representable real number in $\mathbb{D}_{p,q}$. Consequently, the finite precision ball computations naturally take place in domains of the form $\mathbb{B}_{p,q}^{*} = \mathcal{B}(\mathbb{D}_{p,q}^{*}, \mathbb{D}_{p,q}^{*})$ rather than $\mathbb{B}_{p,q}$. Of course, balls with infinite radius carry no useful information about the result. In order to ease the reading, we will assume the absence of overflows in what follows, and concentrate on computations with ordinary numbers in $\mathbb{B}_{p,q}$. We will only consider infinities if they are used in an essential way during the computation.

Similarly, if we want ball arithmetic to be a natural extension of the IEEE 754 norm, then we need an equivalent of NaN. One approach consists of introducing a NaB (not a ball) object, which could be represented by $\mathcal{B}(\mathrm{NaN}, \mathrm{NaN})$. A ball function f° returns NaB if f returns NaN for one selection of members of the input balls. For instance, $\mathrm{sqrt}^{\circ}(\mathcal{B}(1, 3)) = \mathrm{NaB}$. An alternative approach consists of the attachment of an additional flag to each ball object, which signals a possible invalid outcome. Following this convention, $\mathrm{sqrt}^{\circ}(\mathcal{B}(1, 3))$ yields $\mathcal{B}(1, 1)^{\mathrm{NaN}}$.

3.3. Implementation details

Using the formulas from the previous section, it is relatively straightforward to implement ball arithmetic as a C++ template library, as we have done in the Mathemagix system. However, in the case of multiple precision arithmetic this is far from optimal. Let us discuss several possible optimizations:

1. Multiple precision libraries such as Mpfr [HLRZ00] suffer from a huge overhead when it comes to moderate (e.g. quadruple) precision computations. Since the radii are always stored in low precision, it is recommended to inline all computations on the radii. In the case of multiplication, this divides the number of function calls by four.

2. When computing with complex numbers $z \in \mathbb{D}_p[\mathrm{i}]$, one may again save several function calls. Moreover, it is possible to regard z as an element of $\mathbb{Z}[\mathrm{i}]\, 2^{\mathbb{Z}}$ rather than $(\mathbb{Z}\, 2^{\mathbb{Z}})[\mathrm{i}]$, i.e. use a single exponent for both the real and imaginary parts of z. This optimization reduces the time spent on exponent computations and mantissa normalizations.

3. Consider a ball $\mathcal{B}(c, r) \in \mathbb{B}_p$ and recall that $c \in \mathbb{D}_p$, $r \in \mathbb{D}_{64}$. If $|c| < 2^{p-64}\, r$, then the $\lfloor \log_2 r + p - \log_2 |c| - 64 \rfloor$ least significant binary digits of c are of little interest. Hence, we may replace c by its closest approximation in $\mathbb{D}_{p'}$, with $p' = \lceil \log_2 |c| + 64 - \log_2 r \rceil$, and reduce the working precision to p'. Modulo slightly more work, it is also possible to share the exponents of the center and the radius.

4. If we don't need large exponents for our multiple precision numbers, then it is possible to use machine doubles $\mathbb{D}_{51,12}$ as our radius type and further reduce the overhead of bound computations.

When combining the above optimizations, it can be hoped that multiple precision ball arithmetic can be implemented almost as efficiently as standard multiple precision arithmetic. However, this requires a significantly higher implementation effort.

Remark 1. It should be noticed that each of the optimizations described above implies some loss of accuracy. For instance, if z is close to the real axis in the second optimization, then the accuracy of the imaginary part will deteriorate significantly with respect to the usual representation. This kind of examples where quality is traded against efficiency are encountered frequently in the area of reliable computing and we will see more of them in sections 6 and 7. Whether we are willing to accept a lower quality for higher efficiency usually depends on our specific subarea or problem. For instance, our second optimization is usually acceptable in a complex number library, since errors for complex numbers are usually bounded in norm. Furthermore, if the computed result is too inaccurate, then it is always possible to repeat the computation using a more expensive algorithm.

3.4. Computable numbers

Given $x \in \mathbb{R}$, $\tilde{x} \in \mathbb{D}$ and $\varepsilon \in \mathbb{D}^{>} = \{\varepsilon \in \mathbb{D} : \varepsilon > 0\}$, we say that $\tilde{x}$ is an *ε-approximation* of x if $|\tilde{x} - x| \leqslant \varepsilon$. A real number $x \in \mathbb{R}$ is said to be *computable* [Tur36, Grz57, Wei00] if there exists an *approximation algorithm* which takes an absolute tolerance $\varepsilon \in \mathbb{D}^{>}$ on input and which returns an ε-approximation of x. We denote by $\mathbb{R}^{\mathrm{com}}$ the set of computable real numbers. We may regard $\mathbb{R}^{\mathrm{com}}$ as a data type, whose instances are represented by approximation algorithms (this is also known as the *Markov representation* [Wei00, Section 9.6]).

In practice, it is more convenient to work with so called ball approximation algorithms: a real number is computable if and only if it admits a *ball approximation algorithm*, which takes a working precision $p \in \mathbb{N}$ on input and returns a *ball approximation* $\mathcal{B}(c_p, r_p) \in \mathbb{B}$ with $x \in \mathcal{B}(c_p, r_p)$ and $\lim_{p \to \infty} r_p = 0$. Indeed, assume that we have a ball approximation algorithm. In order to obtain an ε-approximation, it suffices to compute ball approximations at precisions $p = 32, 64, 128, \ldots$ which double at every step, until $r_p \leqslant \varepsilon$. Conversely, given an approximation algorithm $\check{x}: \mathbb{D}^{>} \to \mathbb{D}$ of $x \in \mathbb{R}^{\text{com}}$, we obtain a ball approximation algorithm $p \mapsto \mathcal{B}(c_p, r_p)$ by taking $r_p = 2^{-p}$ and $c_p = \check{x}(r_p)$.

Given $x, y \in \mathbb{R}^{\text{com}}$ with ball approximation algorithms $\check{x}, \check{y}: \mathbb{N} \mapsto \mathbb{B}$, we may compute ball approximation algorithms for $x + y, x - y, x y$ simply by taking

$$\begin{aligned} (\widecheck{x + y})(p) &= \check{x}(p) +^{\circ} \check{y}(p) \\ (\widecheck{x - y})(p) &= \check{x}(p) -^{\circ} \check{y}(p) \\ (\widecheck{x y})(p) &= \check{x}(p) \cdot^{\circ} \check{y}(p). \end{aligned}$$

More generally, assuming a good library for ball arithmetic, it is usually easy to write a wrapper library with the corresponding operations on computable numbers.

From the efficiency point of view, it is also convenient to work with ball approximations. Usually, the radius r_p satisfies

$$\log_2 r_p = -p + o(1),$$

or at least

$$\log_2 r_p = -c p + o(1),$$

for some $c \in \mathbb{Q}^{>}$. In that case, doubling the working precision p until a sufficiently good approximation is found is quite efficient. An even better strategy is to double the "expected running time" at every step [vdH06, KR06]. Yet another approach will be described in section 3.6 below.

The concept of computable real numbers readily generalizes to more general normed vector spaces. Let $\mathbb{V}$ be a normed vector space and let $\mathbb{V}^{\text{dig}}$ be an effective subset of *digital points* in $\mathbb{V}$, i.e. the elements of $\mathbb{V}^{\text{dig}}$ admit a computer representation. For instance, if $\mathbb{V} = \mathbb{R}$, then we take $\mathbb{V}^{\text{dig}} = \mathbb{D}$. Similarly, if $\mathbb{V} = \mathbb{R}^{m \times n}$ is the set of real $m \times n$ matrices, with one of the matrix norms from section 2.3, then it is natural to take $\mathbb{V}^{\text{dig}} = (\mathbb{R}^{\text{dig}})^{m \times n} = \mathbb{D}^{m \times n}$. A point $x \in \mathbb{V}$ is said to be *computable*, if it admits an *approximation algorithm* which takes $\varepsilon \in \mathbb{D}^{>}$ on input, and returns an ε-*approximation* $\tilde{x} \in \mathbb{V}^{\text{dig}}$ of x (satisfying $\|\tilde{x} - x\| \leqslant \varepsilon$, as above).

3.5. Asymmetric computability

A real number x is said to be *left computable* if there exists an algorithm for computing an increasing sequence $\check{x}: \mathbb{N} \mapsto \mathbb{D}; n \mapsto \check{x}_n$ with $\lim_{n \to \infty} \check{x}_n = x$ (and $\check{x}$ is called a left approximation algorithm). Similarly, x is said to be *right computable* if $-x$ is left computable. A real number is computable if and only if it is both left and right computable. Left computable but non computable numbers occur frequently in practice and correspond to "computable lower bounds" (see also [Wei00, vdH07b]).

We will denote by $\mathbb{R}^{\mathrm{lcom}}$ and $\mathbb{R}^{\mathrm{rcom}}$ the data types of left and right computable real numbers. It is convenient to specify and implement algorithms in computable analysis in terms of these data types, whenever appropriate [vdH07b]. For instance, we have computable functions

$$\begin{array}{rcl} +: \mathbb{R}^{\mathrm{lcom}} \times \mathbb{R}^{\mathrm{lcom}} & \rightarrow & \mathbb{R}^{\mathrm{lcom}} \\ -: \mathbb{R}^{\mathrm{lcom}} \times \mathbb{R}^{\mathrm{rcom}} & \rightarrow & \mathbb{R}^{\mathrm{lcom}} \\ & \vdots & \end{array}$$

More generally, given a subset $S \subseteq \mathbb{R}$, we say that $x \in S$ is left computable in S if there exists a left approximation algorithm $\check{x}: \mathbb{N} \rightarrow S$ for x. We will denote by S^{lcom} and S^{rcom} the data types of left and computable numbers in S, and define $S^{\mathrm{com}} = S^{\mathrm{lcom}} \cap S^{\mathrm{rcom}}$.

Identifying the type of boolean numbers $\mathbb{T}$ with $\{0, 1\}$, we have $\mathbb{T}^{\mathrm{lcom}} = \mathbb{T}^{\mathrm{rcom}} = \mathbb{T}$ as sets, but *not* as data types. For instance, it is well known that equality is non computable for computable real numbers [Tur36]. Nevertheless, equality *is* "ultimately computable" in the sense that there exists a computable function

$$=: \mathbb{R}^{\mathrm{com}} \times \mathbb{R}^{\mathrm{com}} \rightarrow \mathbb{T}^{\mathrm{rcom}}.$$

Indeed, given $x, y \in \mathbb{R}^{\mathrm{com}}$ with ball approximation algorithms $\check{x}$ and $\check{y}$, we may take

$$(\widetilde{x = y})_n = \begin{cases} 1 & \text{if } (\check{x}_0 \cap \check{y}_0 \cap \cdots \cap \check{x}_n \cap \check{y}_n) \neq \varnothing \\ 0 & \text{otherwise} \end{cases}$$

Similarly, the ordering relation $\leqslant$ is ultimately computable.

This asymmetric point of view on equality testing also suggest a semantics for the relations $=$, $\leqslant$, etc. on balls. For instance, given balls $x, y \in \mathbb{B}$, it is natural to take

$$\begin{array}{rcl} x = y & \rightsquigarrow & x \cap y \neq \varnothing \\ x \neq y & \rightsquigarrow & x \cap y = \varnothing \\ x \leqslant y & \rightsquigarrow & \exists a \in x, b \in y, a \leqslant b \\ & \vdots & \end{array}$$

These definitions are interesting if balls are really used as successive approximations of a real number. An alternative application of ball arithmetic is for modeling non-deterministic computations: the ball models a set of possible values and we are interested in the set of possible outcomes of an algorithm. In that case, the natural return type of a relation on balls becomes a "boolean ball". In the area of interval analysis, this second interpretation is more common [ANS09].

Remark 2. We notice that the notions of computability and asymmetric computability do not say anything about the speed of convergence. In particular, it is usually impossible to give useful complexity bounds for algorithms which are based on these mere concepts. In the case of asymmetric computability, there even do not exist any recursive complexity bounds, in general.

3.6. Lipschitz ball arithmetic

Given a computable function $f: \mathbb{R}^{\mathrm{com}} \rightarrow \mathbb{R}^{\mathrm{com}}$, $x \in \mathbb{R}^{\mathrm{com}}$ and $\varepsilon \in \mathbb{D}^{>}$, let us return to the problem of efficiently computing an approximation $\tilde{y} \in \mathbb{D}$ of $y = f(x)$ with $|\tilde{y} - y| < \varepsilon$. In section 3.4, we suggested to compute ball approximations of y at precisions which double at every step, until a sufficiently precise approximation is found. This computation involves an implementation $f^{\circ}: \mathbb{B} \rightarrow \mathbb{B}$ of f on the level of balls, which satisfies

$$\{f(X): x \in X\} \subseteq f^{\circ}(X), \tag{13}$$

for every ball $X\in\mathbb{B}$. In practice, f is often differentiable, with $f'(x)\neq 0$. In that case, given a ball approximation X of x, the computed ball approximation $Y=f^{\circ}(X)$ of y typically has a radius

$$Y_r \;\sim\; |f'(x)|\,X_r, \tag{14}$$

for $X_r\rightarrow 0$. This should make it possible to directly predict a sufficient precision at which $Y_r\leqslant\varepsilon$. The problem is that (14) needs to be replaced by a more reliable relation. This can be done on the level of ball arithmetic itself, by replacing the usual condition (13) by

$$\begin{aligned}\mathcal{B}(f(X_c),M X_r) &\subseteq f^{\circ}(X) \\ M &= \sup_{x\in X}|f'(x)|.\end{aligned} \tag{15}$$

Similarly, the multiplication of balls is carried out using

$$\mathcal{B}(x,r)\cdot^{\circ}\mathcal{B}(y,s) \;=\; \mathcal{B}(x\,y,(|x|+r)\,s+(|y|+s)\,r). \tag{16}$$

instead of (7). A variant of this kind of “Lipschitz ball arithmetic” has been implemented in [M0]. Although a constant factor is gained for high precision computations at regular points x, the efficiency deteriorates near singularities (i.e. the computation of $\sqrt{0}$).

4. Balls versus intervals

In the area of reliable computation, interval arithmetic has for long been privileged with respect to ball arithmetic. Indeed, balls are often regarded as a more or less exotic variant of intervals, based on an alternative midpoint-radius representation. Historically, interval arithmetic is also preferred in computer science because it is easy to implement if floating point operations are performed with correct rounding. Since most modern microprocessors implement the IEEE 754 norm, this point of view is well supported by hardware.

Not less historically, the situation in mathematics is inverse: whereas intervals are the standard in computer science, balls are the standard in mathematics, since they correspond to the traditional ε-δ-calculus. Even in the area of interval analysis, one usually resorts (at least implicitly) to balls for more complex computations, such as the inversion of a matrix [HS67, Moo66]. Indeed, balls are more convenient when computing error bounds using perturbation techniques. Also, we have a great deal of flexibility concerning the choice of a norm. For instance, a vectorial ball is not necessarily a Cartesian product of one dimensional balls.

In this section, we will give a more detailed account on the respective advantages and disadvantages of interval and ball arithmetic.

4.1. Standardization

One advantage of interval arithmetic is that the IEEE 754 norm suggests a natural and standard implementation. Indeed, let f be a real function which is increasing on some interval I. Then the natural interval lift $f^{\mapsto}$ of f is given by

$$f^{\mapsto}([l,h]) \;=\; [f^{\downarrow}(l),f^{\uparrow}(h)].$$

This implementation has the property that $f^{\vdash\!\dashv}([l,\, h])$ is the smallest interval with end-points in $\mathbb{D}_{p,q} \cup \{\pm\infty\}$, which satisfies

$$\{f(x) \colon x \in [l,h]\} \;\subseteq\; f^{\vdash\!\dashv}([l,h]).$$

For not necessarily increasing functions f this property can still be used as a requirement for the "standard" implementation of $f^{\vdash\!\dashv}$. For instance, this leads to the following implementation of the cosine function on intervals:

$$\cos^{\vdash\!\dashv}([l,h]) \;=\; \begin{cases} [\cos^{\downarrow} l, \cos^{\uparrow} h] & \text{if } \lfloor l/2\pi \rfloor = \lfloor h/2\pi \rfloor \in 2\mathbb{Z} - 1 \\ [\cos^{\downarrow} h, \cos^{\uparrow} l] & \text{if } \lfloor l/2\pi \rfloor = \lfloor h/2\pi \rfloor \in 2\mathbb{Z} \\ [\min(\cos^{\downarrow} l, \cos^{\downarrow} h), 1] & \text{if } \lfloor l/2\pi \rfloor = \lfloor h/2\pi \rfloor - 1 \in 2\mathbb{Z} - 1 \\ [-1, \max(\cos^{\uparrow} l, \cos^{\uparrow} h)] & \text{if } \lfloor l/2\pi \rfloor = \lfloor h/2\pi \rfloor - 1 \in 2\mathbb{Z} \\ [-1, 1] & \text{if } \lfloor l/2\pi \rfloor < \lfloor h/2\pi \rfloor - 1 \end{cases}$$

Such a standard implementation of interval arithmetic has the convenient property that programs will execute in the same way on any platform which conforms to the IEEE 754 standard.

By analogy with the above approach for standardized interval arithmetic, we may standardize the ball implementation f° of f by taking

$$f^{\circ}(\mathcal{B}(c,r)) \;=\; \mathcal{B}(f^{\updownarrow}(c), s),$$

where the radius s is smallest in $\mathbb{D}_{p,q} \cup \{+\infty\}$ with

$$\{f(x) \colon x \in \mathcal{B}(c,r)\} \;\subseteq\; \mathcal{B}(f^{\updownarrow}(c), s).$$

Unfortunately, the computation of such an optimal s is not always straightforward. In particular, the formulas (10), (11) and (12) do not necessarily realize this tightest bound. In practice, it might therefore be better to achieve standardization by fixing once and for all the formulas by which ball operations are performed. Of course, more experience with ball arithmetic is required before this can happen.

4.2. Practical efficiency

The respective efficiencies of interval and ball arithmetic depend on the precision at which we are computing. For high precisions and most applications, ball arithmetic has the advantage that we can still perform computations on the radius at single precision. By contrast, interval arithmetic requires full precision for operations on both end-points. This makes ball arithmetic twice as efficient at high precisions.

When working at machine precision, the efficiencies of both approaches essentially depend on the hardware. *A priori*, interval arithmetic is better supported by current computers, since most of them respect the IEEE 754 norm [ANS08], whereas the function Δ from (9) usually has to be implemented by hand. However, changing the rounding mode is often highly expensive (over hundred cycles). Therefore, additional gymnastics may be required in order to always work with respect to a fixed rounding mode. For instance, if ↑ is our current rounding mode, then we may take [vG02, Lam08]

$$x +^{\downarrow} y \;=\; -((-x) +^{\uparrow} (-y)),$$

since the operation $x \mapsto -x$ is always exact (i.e. does not depend on the rounding mode). As a consequence, interval arithmetic becomes slightly more expensive. By contrast, when releasing the condition that centers of balls are computed using rounding to the nearest, we may replace (8) by

$$y = f^{\uparrow}(x_1, \ldots, x_k) \tag{17}$$

and (9) by

$$\Delta(y) := (|y| +^{\uparrow} \eta) \cdot^{\uparrow} (2\,\epsilon).$$

Hence, ball arithmetic already allows us to work with respect to a fixed rounding mode. Of course, using (17) instead of (8) does require to rethink the way ball arithmetic should be standardized.

An alternative technique for avoiding changes in rounding mode exists when performing operations on compound types, such as vectors or matrices. For instance, when adding two vectors, we may first add all lower bounds while rounding downwards and next add the upper bounds while rounding upwards. Unfortunately, this strategy becomes more problematic in the case of multiplication, because different rounding modes may be needed depending on the signs of the multiplicands. As a consequence, matrix operations tend to require many conditional parts of code when using interval arithmetic, with increased probability of breaking the processor pipeline. On the contrary, ball arithmetic highly benefits from parallel architecture and it is easy to implement ball arithmetic for matrices on top of existing libraries: see [Rum99a] and section 6 below.

4.3. Quality

Besides the efficiency of ball and interval arithmetic for basic operations, it is also important to investigate the quality of the resulting bounds. Indeed, there are usually differences between the sets which are representable by balls and by intervals. For instance, when using the extended IEEE 754 arithmetic with infinities, then it is possible to represent $[1, \infty]$ as an interval, but not as a ball.

These differences get more important when dealing with complex numbers or compound types, such as matrices. For instance, when using interval arithmetic for reliable computations with complex numbers, it is natural to enclose complex numbers by rectangles $X + Y\,\mathrm{i}$, where X and Y are intervals. For instance, the complex number $z = 1 + \mathrm{i}$ may be enclosed by

$$z \in [1 - \varepsilon, 1 + \varepsilon] + [1 - \varepsilon, 1 + \varepsilon]\,\mathrm{i},$$

for some small number ε. When using ball arithmetic, we would rather enclose z by

$$z \in \mathcal{B}(1 + \mathrm{i}, \varepsilon).$$

Now consider the computation of $u = z^2$. The computed rectangular and ball enclosures are given by

$$\begin{aligned} u &\in [-2\,\varepsilon, 2\,\varepsilon] + [2 - 2\,\varepsilon, 2 + 2\,\varepsilon]\,\mathrm{i} + o(\varepsilon) \\ u &\in \mathcal{B}(2, \sqrt{2}\,\varepsilon) + o(\varepsilon). \end{aligned}$$

Consequently, ball arithmetic yields a much better bound, which is due to the fact that multiplication by $1 + \mathrm{i}$ turns the rectangular enclosure by 45 degrees, leading to an overestimation by a factor $\sqrt{2}$ when re-enclosing the result by a horizontal rectangle (see figure 2).

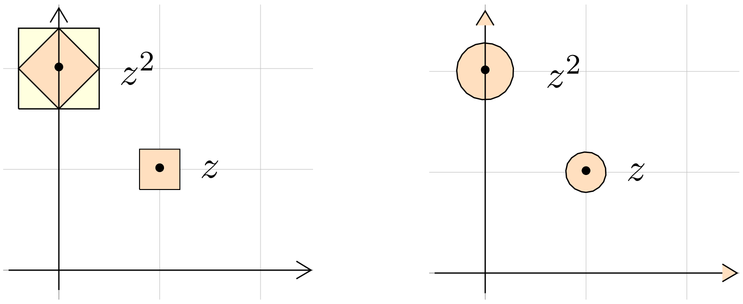

Figure 2. Illustration of the computation of z^2 using interval and ball arithmetic, for $z = 1 + \mathrm{i}$.

The above computation of $\mathcal{B}(1 + \mathrm{i}, \varepsilon)^2$ is one of the simplest instances of the *wrapping effect* [Moo66]. This example highlights another advantage of ball arithmetic: we have a certain amount of flexibility regarding the choice of the radius type. By choosing a simple radius type, we do not only reduce the wrapping effect, but also improve the efficiency: when computing with complex balls in $\mathbb{B}[\mathrm{i}]$, we only need to bound one radius instead of two for every basic operation. More precisely, we replace (8) and (9) by

$$\begin{aligned} y &= f^{\updownarrow}(x_1, \ldots, x_k) \\ &= (\operatorname{Re} f)^{\updownarrow}(x_1, \ldots, x_k) + (\operatorname{Im} f)^{\updownarrow}(x_1, \ldots, x_k)\,\mathrm{i} \\ \Delta(y) &:= (\operatorname{abs}^{\uparrow}(y) +^{\uparrow} \eta) \cdot^{\uparrow} (\sqrt{2}\,\epsilon). \end{aligned} \tag{18}$$

Of course, in some very specific situations, interval representations might still be preferred. For instance, consider the rectangle $Z = [a, b] + [-\varepsilon, \varepsilon]\,\mathrm{i}$. Then any ball enclosure of Z is necessarily very rough. However, when developing a general purpose library for reliable complex numbers, this situation is very exotic (see also remark 1): as soon as we will be multiplying truly complex numbers with non trivial real and imaginary parts, the potential benefit of interval enclosures will be lost as a consequence of the wrapping effect.

The choice of a real radius type completely eliminates the wrapping effect in the case of complex balls, due to the fact that $|u\,z| = |u|\,|z|$ for any $u, z \in \mathbb{C}$. In analogy with the complex case, we may prefer to compute the square of the matrix

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{19}$$

in $\mathcal{B}(\mathbb{D}^{2\times 2}, \mathbb{D})$ rather than $\mathcal{B}(\mathbb{D}, \mathbb{D})^{2\times 2}$, while using the operator norm (3) for matrices. Unfortunately, for general 2×2 matrices, some overestimation may occur, due to the fact that we only have $\|M\,N\| \leqslant \|M\|\,\|N\|$ for $M, N \in \mathbb{R}^{2\times 2}$. Similarly, is not uncommon that the entries of the matrices are of different orders of magnitude, in which case a single bound on the norm leads to a deterioration of the accuracy. Finally, generalized norms may be harder to compute, even though a rough bound often suffices (e.g. replacing $\operatorname{abs}^{\uparrow}(y)$ by $|\operatorname{Re} y| +^{\uparrow} |\operatorname{Im} y|$ in (18)).

In the area of reliable integration of dynamical systems, various techniques have been developed [Moo66, Loh88, GS88, Neu93, MB96] in order to reduce the wrapping effect induced by matrix computations. Many of these techniques rely on the use of suitable enclosures: linear transformations of higher dimensional rectangles, non linear transformations under Taylor models, ellipsoids, zonotopes, etc. Another, more algorithmic, technique for reducing the wrapping effect will be discussed in sections 6.4 and 7.5 below.

4.4. Benefits and cost of correct rounding

Implementations of interval arithmetic often rely on floating point arithmetic with correct rounding. One may question how good correct rounding actually is in order to achieve reliability. One major benefit is that it provides a simple and elegant way to specify what a mathematical function precisely does at limited precision. In particular, it allows numerical programs to execute exactly in the same way on many different hardware architectures.

On the other hand, correct rounding does have a certain cost. Although the cost is limited for field operations and elementary functions [Mul06], the cost increases for more complex special functions, especially if one seeks for numerical methods with a constant operation count. For arbitrary computable functions on $\mathbb{R}^{\text{com}}$, correct rounding even becomes impossible. Another disadvantage is that correct rounding is lost as soon we perform more than one operation: in general, $g^{\uparrow} \circ f^{\uparrow}$ and $(g \circ f)^{\uparrow}$ do not coincide.

In the case of ball arithmetic, we only require an upper bound for the error, not necessarily the best possible representable one. In principle, this is just as reliable and usually more economic. In a similar way, when computing with intervals, we may require faithful rounding instead of correct rounding. That is, we require guaranteed lower and upper bounds for the interval, but not necessarily the best possible ones. Of course, faithful rounding is not unique, so we will lose the benefit of standardization (see section 4.1).

Now in an ideal safe world, the development of numerical codes goes hand in hand with the systematic development of routines which compute the corresponding error bounds. In such a world, correct rounding becomes superfluous, since correctness is no longer ensured at the micro-level of hardware available functions, but rather at the top-level, *via* mathematical proof.

4.5. Development of new reliable algorithms

When conceiving a new reliable algorithm, it is often necessary to carry out a lot of tedious bound computations by hand. In the process of software development, not only execution time, but also development time can therefore be a major concern. Depending on the algorithm or application area, as well as personal taste, error bounds may be easier to obtain when using either interval arithmetic or ball arithmetic.

If we are essentially performing computations with real functions on regions where they are monotonic, then interval enclosures are often preferred, since they are both easy to obtain, efficient and tight. Another typical area where interval arithmetic is more convenient is the resolution of a system of equations using dichotomy. Indeed, it is easier to cut an n-dimensional block $[a_1, b_1] \times \cdots \times [a_n, b_n]$ into 2^n smaller blocks than to perform a similar operation on balls.

For many other applications, ball representations are more convenient. Indeed, error bounds are usually obtained by perturbation methods. For any mathematical proof where error bounds are explicitly computed in this way, it is generally easy to derive a certified algorithm based on ball arithmetic. We will see several illustrations of this principle in the sections below.

Whenever possible, it is also useful to develop algorithms in such a way that the underlying arithmetic (e.g. interval or ball representation) can be chosen by the user. For instance, in the C++ libraries of the MATHEMAGIX system, many reliable algorithms are templated by a type which can be either be taken to be an interval type or a ball type. Usually, the important feature of the template type is the *ability* to compute centers, radii, upper or lower bounds, etc., rather than a specific way of doing so.

5. The numerical hierarchy

Computable analysis provides a good high-level framework for the automatic and certified resolution of analytic problems. The user states the problem in a formal language and specifies a required absolute or relative precision. The program should return a numerical result which is certified to meet the requirement on the precision. A simple example is to compute an ε-approximation for π, for a given $\varepsilon \in \mathbb{D}^{>}$.

The Mathemagix system [vdHLM+02] aims the implementation of efficient algorithms for the certified resolution of numerical problems. Our ultimate goal is that these algorithms become as efficient as more classical numerical methods, which are usually non certified and only operate at limited precision. A naive approach is to systematically work with computable real numbers. Although this approach is convenient for theoretical purposes in the area of computable analysis, the computation with functions instead of ordinary floating point numbers is highly inefficient.

In order to address this efficiency problem, the Mathemagix libraries for basic arithmetic on analytic objects (real numbers, matrices, polynomials, etc.) are subdivided into four layers of the so called *numerical hierarchy* (see figure 3). We will illustrate this decomposition on the problem of multiplying two $n \times n$ computable real matrices. The numerical hierarchy turns out to be a convenient framework for more complex problems as well, such as the analytic continuation of the solution to a dynamical system. As a matter of fact, the framework incites the developer to restate the original problem at the different levels, which is generally a good starting point for designing an efficient solution.

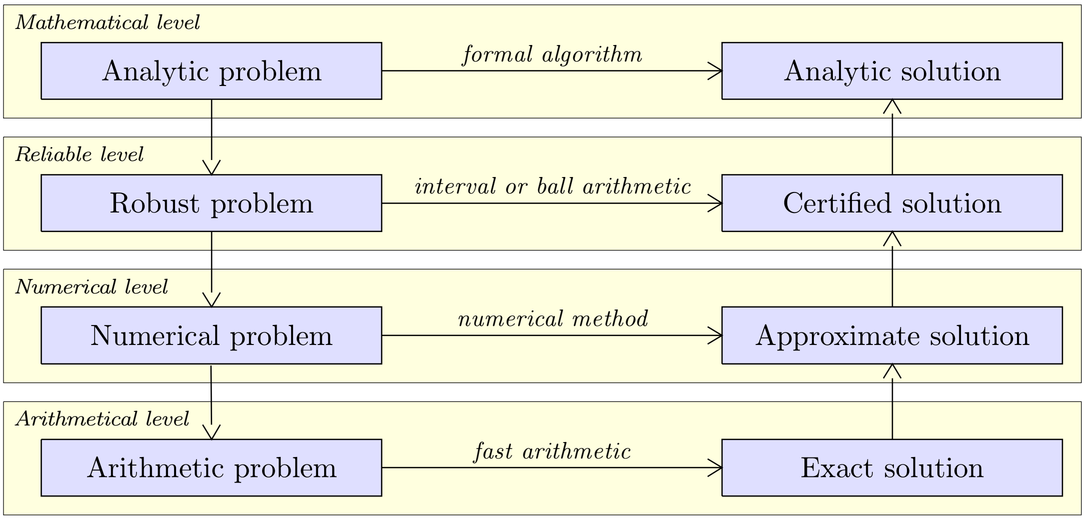

Figure 3. The numerical hierarchy.

Mathematical level. On the mathematical top level, we are given two computable real $n \times n$ matrices $A, B \in (\mathbb{R}^{\text{com}})^{n \times n}$ and an absolute error $\varepsilon \in \mathbb{D}^{>}$. The aim is to compute ε-approximations for all entries of the product $C = A B$.

The simplest approach to this problem is to use a generic formal matrix multiplication algorithm, using the fact that $\mathbb{R}^{\text{com}}$ is an effective ring. However, as stressed above, instances of $\mathbb{R}^{\text{com}}$ are really functions, so that ring operations in $\mathbb{R}^{\text{com}}$ are quite expensive. Instead, when working at precision p, we may first compute ball approximations for all the entries of A and B, after which we form two approximation matrices $\tilde{A}, \tilde{B} \in \mathbb{B}^{n \times n}$. The multiplication problem then reduces to the problem of multiplying two matrices in $\mathbb{B}^{n \times n}$. This approach has the advantage that $O(n^3)$ operations on "functions" in $\mathbb{R}^{\text{com}}$ are replaced by a single multiplication in $\mathbb{B}^{n \times n}$.

Reliable level. The aim of this layer is to implement efficient algorithms on balls. Whereas the actual numerical computation is delegated to the numerical level below, the reliable level should be able to perform the corresponding error analysis automatically.

When operating on non-scalar objects, such as matrices of balls, it is often efficient to rewrite the objects first. For instance, when working in fixed point arithmetic (i.e. all entries of A and B admit similar orders of magnitude), a matrix in $\mathbb{B}^{n \times n}$ may also be considered as a *matricial ball* in $\mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D})$ for the matrix norm $\|\cdot\|_\infty$. We multiply two such balls using the formula

$$\mathcal{B}(x, r) \cdot^\circ \mathcal{B}(y, s) \;=\; \mathcal{B}(x\, y, [n\, ((\|x\|_\infty + r)\, s + r\, \|y\|_\infty)]^\uparrow),$$

which is a corrected version of (7), taking into account that the matrix norm $\|\cdot\|_\infty$ only satisfies $\|x\, y\|_\infty \leqslant n\, \|x\|_\infty\, \|y\|_\infty$. Whereas the naive multiplication in $\mathbb{B}^{n \times n}$ involves $4\, n^3$ multiplications in $\mathbb{D}$, the new method reduces this number to $n^3 + 2\, n^2$: one "expensive" multiplication in $\mathbb{D}^{n \times n}$ and two scalar multiplications of matrices by numbers. This type of tricks will be discussed in more detail in section 6.4 below.

Essentially, the new method is based on the isomorphism

$$\mathcal{B}(\mathbb{D}, \mathbb{D})^{n \times n} \;\cong\; \mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D}).$$

A similar isomorphism exists on the mathematical level:

$$(\mathbb{R}^{\text{com}})^{n \times n} \;\cong\; (\mathbb{R}^{n \times n})^{\text{com}}$$

As a variant, we directly may use the latter isomorphism at the top level, after which ball approximations of elements of $(\mathbb{R}^{n \times n})^{\text{com}}$ are already in $\mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D})$.

Numerical level. Being able to perform an automatic error analysis, the actual numerical computations are done at the numerical level. In our example, we should implement an efficient algorithm to multiply two matrices in $\mathbb{D}^{n \times n}$. In single or double precision, we may usually rely on highly efficient numerical libraries (Blas, Lapack, etc.). In higher precisions, new implementations are often necessary: even though there are efficient libraries for multiple precision floating point numbers, these libraries usually give rise to a huge overhead. For instance, when using Mpfr [HLRZ00] at double precision, the overhead with respect to machine doubles is usually comprised between 10 and 100.

When working with matrices with multiple precision floating point entries, this overhead can be greatly reduced by putting the entries under a common exponent using the isomorphism

$$\mathbb{D}^{n \times n} = (\mathbb{Z}\, 2^{\mathbb{Z}})^{n \times n} \;\cong\; \mathbb{Z}^{n \times n}\, 2^{\mathbb{Z}}.$$

This reduces the matrix multiplication problem for floating point numbers to a purely arithmetic problem. Of course, this method becomes numerically unstable when the exponents differ wildly; in that case, row preconditioning of the first multiplicand and column preconditioning of the second multiplicand usually helps.

We notice that a matrix multiplication algorithm has been proposed in [OOO11], which is essentially based on a similar principle, but with difference that floating point numbers are cut into smaller floating point numbers instead of integers. This approach can be more efficient when an efficient BLAS library is available and if the precision is not too high.

Arithmetic level. After the above succession of reductions, we generally end up with an arithmetic problem such as the multiplication of two $n \times n$ matrices in $\mathbb{Z}^{n \times n}$. The efficient resolution of this problem for all possible n and integer bit lengths p is again non-trivial.

Indeed, libraries such as Gmp [Gra91] do implement Schönhage-Strassen's algorithm [SS71] algorithm for integer multiplication. However, the corresponding naive algorithm for the multiplication of matrices has a time complexity $O(\mathsf{I}(p)\, n^3)$, which is far from optimal for large values of n.

Indeed, for large n, it is better to use multi-modular methods. For instance, choosing sufficiently many small primes $q_1, \ldots, q_k < 2^{32}$ (or 2^{64}) with $q_1 \cdots q_k > 2\, n\, 4^p$, the multiplication of the two integer matrices can be reduced to k multiplications of matrices in $(\mathbb{Z}/q_i\,\mathbb{Z})^{n\times n}$. Recall that a $2\,p$-bit number can be reduced modulo all the q_i and reconstructed from these reductions in time $O(\mathsf{I}(p)\log p)$. The improved matrix multiplication algorithm therefore admits a time complexity $O(p\, n^3 + n^2\, \mathsf{I}(p)\log p)$ and has been implemented in Linbox [DGG+02b, DGG+02a], Mathemagix and several other systems. FFT-based methods achieve similar practical time complexities $O(p\, n^3 + n^2\, \mathsf{I}(p))$ when n and p are of the same order of magnitude.

6. Reliable linear algebra

In this section, we will start the study of ball arithmetic for non numeric types, such as matrices. We will examine the complexity of common operations, such as matrix multiplication, linear system solving, and the computation of eigenvectors. Ideally, the certified variants of these operations are only slightly more expensive than the non certified versions. As we will see, this objective can sometimes be met indeed. In general however, there is a trade-off between the efficiency of the certification and its quality, i.e. the sharpness of the obtained bound. As we will see, the overhead of bound computations also tends to diminish for increasing bit precisions p.

6.1. Matrix multiplication

Let us first consider the multiplication of two $n \times n$ double precision matrices

$$M, N \in \mathbb{B}_{51,12}^{n\times n}.$$

Naive strategy. The simplest multiplication strategy is to compute $M\,N$ using the naive symbolic formula

$$(M N)_{i,k} \;=\; \sum_{j=1}^{n} M_{i,j}\, N_{j,k}. \tag{20}$$

Although this strategy is efficient for very small n, it has the disadvantage that we cannot profit from high performance Blas libraries which might be available on the computer.

Revisited naive strategy. Reconsidering M and N as balls with matricial radii

$$M, N \in \mathcal{B}(\mathbb{D}_{51,12}^{n\times n}, \mathbb{D}_{51,12}^{n\times n}),$$

we may compute $M\,N$ using

$$M N \;=\; \mathcal{B}([M_c N_c]^{\updownarrow}, [|M_c|\, N_r + M_r\, (|N_c| + N_r) + n\,\epsilon\, |M_c|\, |N_c|]^{\uparrow}), \tag{21}$$

where $|M|$ is given by $|M|_{i,j} = |M_{i,j}|$. A similar approach was first proposed in [Rum99a]. Notice that the additional term $n\,\epsilon\, |M_c| \cdot^{\uparrow} |N_c|$ replaces $\Delta(M\,N)$. This extra product is really required: the computation of $(M_c\, N_c)_{i,j}$ may involve cancellations, which prevent a bound for the rounding errors to be read off from the end-result. The formula (21) does assume that the underlying Blas library computes $M_c\, N_c$ using the naive formula (20) and correct rounding, with the possibility to compute the sums in any suitable order. Less naive schemes, such as Strassen multiplication [Str69], may give rise to additional rounding errors.

Fast strategy. The above naive strategies admit the disadvantage that they require four non certified $n \times n$ matrix products in $\mathbb{D}_{51,12}^{n \times n}$. If M and N are well-conditioned, then the following formula may be used instead:

$$MN = \mathcal{B}(M_c N_c, R) \tag{22}$$

$$R_{i,k} = [\|(M_{i,\cdot})_c\| \, \|(N_{\cdot,k})_r\| + \|(M_{i,\cdot})_r\| \, \|(N_{\cdot,k})_c\| + n \, \epsilon \, \|(M_{i,\cdot})_c\| \, \|(N_{i,\cdot})_c\|]^{\uparrow}, \tag{23}$$

where $M_{i,\cdot}$ and $N_{\cdot,k}$ stand for the i-th row of M and the k-th column of N. Since the $O(n)$ norms can be computed using only $O(n^2)$ operations, the cost of the bound computation is asymptotically negligible with respect to the $O(n^3)$ cost of the multiplication $M_c N_c$.

Hybrid strategy. For large $n \times n$ matrices, chances increase that M or N gets badly conditioned, in which case the quality of the error bound (23) decreases. Nevertheless, we may use a compromise between the naive and the fast strategies: fix a not too small constant K, such as $K \approx 16$, and rewrite M and N as $\lceil \frac{n}{K} \rceil \times \lceil \frac{n}{K} \rceil$ matrices whose entries are $K \times K$ matrices. Now multiply M and N using the revisited naive strategy, but use the fast strategy on each of the $K \times K$ block coefficients. Being able to choose K, the user has an explicit control over the trade-off between the efficiency of matrix multiplication and the quality of the computed bounds.

As an additional, but important observation, we notice that the user often has the means to perform an "*a posteri* quality check". Starting with a fast but low quality bound computation, we may then check whether the computed bound is suitable. If not, then we recompute a better bound using a more expensive algorithm.

High precision multiplication. Assume now that we are using multiple precision arithmetic, say $M, N \in \mathbb{B}_p^{n \times n}$. Computing MN using (21) requires one expensive multiplication in $\mathbb{D}_p^{n \times n}$ and three cheap multiplications in $\mathbb{D}_{64}^{n \times n}$. For large p, the bound computation therefore induces no noticeable overhead.

6.2. Matrix inversion

Assume that we want to invert an $n \times n$ ball matrix $M \in \mathbb{B}_p^{n \times n}$. This a typical situation where the naive application of a symbolic algorithm (such as gaussian elimination or LR-decomposition) may lead to overestimation. An efficient and high quality method for the inversion of M is called Hansen's method [HS67, Moo66]. The main idea is to first compute the inverse of M_c using a standard numerical algorithm. Only at the end, we estimate the error using a perturbative analysis. The same technique can be used for many other problems.

More precisely, we start by computing an approximation $N_c \in \mathbb{D}_p^{n \times n}$ of $(M_c)^{-1}$. Putting $N = \mathcal{B}(N_c, 0)$, we next compute the product MN using ball arithmetic. This should yield a matrix of the form $1 - E$, where E is small. If E is indeed small, say $\|E\| \leqslant 2^{-p/2}$, then

$$\|(1 - E)^{-1} - 1 - E\| \leqslant \frac{\|E\|^2}{1 - \|E\|}. \tag{24}$$

Denoting by Ω_n the $n \times n$ matrix whose entries are all $\mathcal{B}(0, 1)$, we may thus take

$$(1 - E)^{-1} := 1 + E + \frac{\|E\|^2}{1 - \|E\|} \Omega_n.$$

Having inverted $1 - E$, we may finally take $M^{-1} = N \, (1 - E)^{-1}$. Notice that the computation of $\|E\|$ can be quite expensive. It is therefore recommended to replace the check $\|E\| \leqslant 2^{-p/2}$ by a cheaper check, such as $n \, \|E\|_{\infty} \leqslant 2^{-p/2}$.

Unfortunately, the matrix E is not always small, even if M is nicely invertible. For instance, starting with a matrix M of the form

$$M = J_{n,K} \;=\; \begin{pmatrix} 1 & K & & \\ & 1 & \ddots & \\ & & \ddots & K \\ & & & 1 \end{pmatrix},$$

with K large, we have

$$M^{-1} \;=\; \begin{pmatrix} 1 & -K & K^2 & \cdots & (-K)^{n-1} \\ & 1 & -K & \ddots & \vdots \\ & & 1 & \ddots & K^2 \\ & & & \ddots & -K \\ & & & & 1 \end{pmatrix}.$$

Computing $(M_c)^{-1}$ using bit precision p, this typically leads to

$$\|E\| \;\approx\; K^{n-1} 2^{-p}.$$

In such cases, we rather reduce the problem of inverting $1-E$ to the problem of inverting $1-E^2$, using the formula

$$(1-E)^{-1} \;=\; (1+E)\,(1-E^2)^{-1}. \tag{25}$$

More precisely, applying this trick recursively, we compute E^2, E^4, E^8, ... until $\|E^{2^k}\|$ becomes small (say $\|E^{2^k}\| \leqslant 2^{-p/2}$) and use the formula

$$\begin{aligned} (1-E)^{-1} \;&=\; (1+E)\,(1+E^2)\cdots(1+E^{2^{k-1}})\,(1-E^{2^k})^{-1} \\ (1-E^{2^k})^{-1} \;&:=\; 1+E^{2^k}+\frac{\|E^{2^k}\|^2}{1-\|E^{2^k}\|}\,\Omega_n. \end{aligned} \tag{26}$$

We may always stop the algorithm for $2^k > n$, since $(J_{n,K}-1)^n = 0$. We may also stop the algorithm whenever $\|E^{2^k}\| \geqslant 1$ and $\|E^{2^k}\| \geqslant \|E^{2^{k-1}}\|$, since M usually fails to be invertible in that case.

In general, the above algorithm requires $O(\log m)$ ball $n \times n$ matrix multiplications, where m is the size of the largest block of the kind $J_{K,m}$ in the Jordan decomposition of M. The improved quality therefore requires an additional $O(\log n)$ overhead in the worst case. Nevertheless, for a fixed matrix M and $p \to \infty$, the norm $\|E\|$ will eventually become sufficiently small (i.e. $2^{-p/2}$) for (24) to apply. Again, the complexity thus tends to improve for high precisions. An interesting question is whether we can avoid the ball matrix multiplication MN altogether, if p gets really large. Theoretically, this can be achieved by using a symbolic algorithm such as Gaussian elimination or LR-decomposition using ball arithmetic. Indeed, even though the overestimation is important, it does not depend on the precision p. Therefore, we have $(M^{-1})_r = O(2^{-p})$ and the cost of the bound computations becomes negligible for large p.

6.3. Eigenproblems

Let us now consider the problem of computing the eigenvectors of a ball matrix $M \in \mathbb{B}_p^{n \times n}$, assuming for simplicity that the corresponding eigenvalues are non zero and pairwise distinct. We adopt a similar strategy as in the case of matrix inversion. Using a standard numerical method, we first compute a diagonal matrix $\Lambda \in \mathbb{D}_p^{n \times n}$ and an invertible transformation matrix $T \in \mathbb{D}_p^{n \times n}$, such that

$$T^{-1} M_c T \;\approx\; \Lambda. \tag{27}$$

The main challenge is to find reliable error bounds for this computation. Again, we will use the technique of small perturbations. The equation (27) being a bit more subtle than $M_c N_c \approx 1$, this requires more work than in the case of matrix inversion. In fact, we start by giving a numerical method for the iterative improvement of an approximate solution. A variant of the same method will then provide the required bounds. Again, this idea can often be used for other problems. The results of this section are work in common with B. MOURRAIN and PH. TREBUCHET; in [vdHMT], an even more general method is given, which also deals with the case of multiple eigenvalues.

Given $\tilde{M}$ close to M_c, we have to find $\tilde{T}$ close to T and $\tilde{\Lambda}$ close to Λ, such that

$$\tilde{T}^{-1} \tilde{M} \tilde{T} = \tilde{\Lambda}. \tag{28}$$

Putting

$$\begin{aligned} \tilde{T} &= T(1+E) \\ \tilde{\Lambda} &= \Lambda(1+\Delta) \\ N &= T^{-1}\tilde{M}T \\ H &= N - \Lambda, \end{aligned}$$

this yields the equation

$$(1+E)^{-1} N (1+E) = \Lambda(1+\Delta).$$

Expansion with respect to E yields

$$\begin{aligned} \frac{1}{1+E} N(1+E) &= N(1+E) - \frac{E}{1+E} N(1+E) \\ &= N + NE - EN + \frac{E^2}{1+E} N - \frac{E}{1+E} NE \\ &= N + [\Lambda, E] + [H, E] + \frac{E^2}{1+E} N - \frac{E}{1+E} NE \qquad (29) \\ &= N + [\Lambda, E] + O([H, E]) + O(E^2). \end{aligned}$$

Forgetting about the non-linear terms, the equation

$$[E, \Lambda] + \Lambda \Delta = H$$

admits a unique solution

$$E_{i,j} = \begin{cases} \frac{H_{i,j}}{\Lambda_{j,j} - \Lambda_{i,i}} & \text{if } j \neq i \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta_{i,j} = \begin{cases} \frac{H_{i,i}}{\Lambda_{i,i}} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Setting

$$\kappa = \kappa(\Lambda) = \max\left\{ \max\left\{ \frac{1}{|\Lambda_{j,j} - \Lambda_{i,i}|} : 1 \leqslant i < j \leqslant n \right\}, \max\left\{ \frac{1}{\Lambda_{i,i}} : 1 \leqslant i \leqslant n \right\} \right\},$$

it follows that

$$\max\{\|E\|, \|\Delta\|\} \leqslant \kappa \sqrt{n} \, \|H\|. \tag{30}$$

Setting $T' = T\,(1+E)$, $\Lambda' = \Lambda\,(1+\Delta)$ and $H' = (T')^{-1}\,\tilde{M}\,T' - \Lambda'$, the relation (29) also implies

$$H' \;=\; [H,E] + \frac{E^2}{1+E}\,(\Lambda+H) - \frac{E}{1+E}\,(\Lambda+H)\,E.$$

Under the additional condition $\|E\| \leqslant \frac{1}{2}$, it follows that

$$\|H'\| \;\leqslant\; 3\,\|H\|\,\|E\| + 4\,\|E\|^2\,\|\Lambda\|. \tag{31}$$

For sufficiently small H, we claim that iteration of the mapping Φ: $(T,\,\Lambda) \mapsto (T',\,\Lambda')$ converges to a solution of (28).

Let us denote $(T^{(k)},\Lambda^{(k)}) = \Phi^k(T,\Lambda)$, $H^{(k)} = (T^{(k)})^{-1}\,\tilde{M}\,T^{(k)} - \Lambda^{(k)}$ and let $(E^{(k)},\Delta^{(k)})$ be such that $T^{(k+1)} = T^{(k)}\,(1+E^{(k)})$ and $\Lambda^{(k+1)} = \Lambda^{(k)}\,(1+\Delta^{(k)})$. Assume that

$$\|H\| \;\leqslant\; \frac{1}{80\,n\,\kappa^2\,\|\Lambda\|} \tag{32}$$

and let us prove by induction over k that

$$\begin{aligned} \|H^{(k)}\| &\leqslant 2^{-k}\,\|H\| && (33)\\ \|\Lambda^{(k)}\| &\leqslant 2\,\|\Lambda\| && (34)\\ \max\{\|E^{(k)}\|,\|\Delta^{(k)}\|\} &\leqslant 2\sqrt{n}\,\kappa\,\|H^{(k)}\| && (35)\\ &\leqslant \frac{1}{40\sqrt{n}\,\kappa\,\|\Lambda\|\,2^k} \end{aligned}$$

This is clear for $k=0$, so assume that $k>0$. In a similar way as (31), we have

$$\|H^{(k)}\| \;\leqslant\; 3\,\|H^{(k-1)}\|\,\|E^{(k-1)}\| + 4\,\|E^{(k-1)}\|^2\,\|\Lambda^{(k-1)}\|. \tag{36}$$

Using the induction hypotheses and $\kappa\,\|\Lambda\| \geqslant 1$, it follows that

$$\begin{aligned} \|H^{(k)}\| &\leqslant (3+16\sqrt{n}\,\kappa\,\|\Lambda\|)\,\|E^{(k-1)}\|\,\|H^{(k-1)}\|\\ &\leqslant \tfrac{1}{2}\,\|H^{(k-1)}\|, \end{aligned}$$

which proves (33). Now let $\Sigma^{(k)}$ be such that

$$\begin{aligned} \Lambda^{(k)} &= \Lambda\,(1+\Delta^{(0)})\cdots(1+\Delta^{(k-1)})\\ &= \Lambda\,(1+\Sigma^{(k)}). \end{aligned}$$

From (35), it follows that

$$\|\Sigma^{(k)}\| \;\leqslant\; \frac{1}{4\,\kappa\,\|\Lambda\|}.$$

On the one hand, this implies (34). On the other hand, it follows that

$$\kappa(\Lambda^{(k)}) \;\leqslant\; 2\,\kappa,$$

whence (30) generalizes to (35). This completes the induction and the linear convergence of $\|H^{(k)}\|$ to zero. In fact, the combination of (35) and (36) show that we even have quadratic convergence.

Let us now return to the original bound computation problem. We start with the computation of $H = T^{-1}\,M\,T - \Lambda$ using ball arithmetic. If the condition (32) is met (using the most pessimistic rounding), the preceding discussion shows that for every $\tilde{M} \in M$ (in the sense that $\tilde{M}_{i,j} \in M_{i,j}$ for all i,j), the equation (28) admits a solution of the form

$$\begin{aligned} \tilde{T} &= T\,(1+\tilde{\Upsilon}) = T\,(1+E^{(0)})\,(1+E^{(1)})\cdots\\ \tilde{\Lambda} &= \Lambda\,(1+\tilde{\Sigma}) = \Lambda\,(1+\Delta^{(0)})\,(1+\Delta^{(1)})\cdots, \end{aligned}$$

with

$$\max \{\|E^{(k)}\|, \|\Delta^{(k)}\|\} \leqslant 2^{1-k} \sqrt{n}\, \kappa\, \|H\|,$$

for all k. It follows that

$$\max \{\|\tilde{\Upsilon}\|, \|\tilde{\Sigma}\|\} \leqslant \eta := 6 \sqrt{n}\, \kappa\, \|H\|.$$

We conclude that

$$\begin{aligned} \tilde{T} &\in T\,(1 + \eta\, \Omega_n) \\ \tilde{\Lambda} &\in \mathcal{B}(1, \eta)\, \Lambda. \end{aligned}$$

We may thus return $(T\,(1 + \eta\, \Omega_n), \mathcal{B}(1, \eta)\, \Lambda)$ as the solution to the original eigenproblem associated to the ball matrix M.

The reliable bound computation essentially reduces to the computation of three matrix products and one matrix inversion. At low precisions, the numerical computation of the eigenvectors is far more expensive in practice, so the overhead of the bound computation is essentially negligible. At higher precisions p, the iteration $(T, \Lambda) \mapsto \Phi(T, \Lambda)$ actually provides an efficient way to double the precision of a numerical solution to the eigenproblem at precision $p/2$. In particular, even if the condition (32) is not met initially, then it usually can be enforced after a few iterations and modulo a slight increase of the precision. For fixed M and $p \to \infty$, it also follows that the numerical eigenproblem essentially reduces to a few matrix products. The certification of the end-result requires a few more products, which induces a constant overhead. By performing a more refined error analysis, it is probably possible to make the cost certification negligible, although we did not investigate this issue in detail.

6.4. Matricial balls versus ball matrices

In section 4.3, we have already seen that matricial balls in $\mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D})$ often provide higher quality error bounds than ball matrices in $\mathcal{B}(\mathbb{D}, \mathbb{D})^{n \times n}$ or essentially equivalent variants in $\mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D}^{n \times n})$. However, ball arithmetic in $\mathcal{B}(\mathbb{D}^{n \times n}, \mathbb{D})$ relies on the possibility to quickly compute a sharp upper bound for the operator norm $\|M\|$ of a matrix $M \in \mathbb{D}^{n \times n}$. Unfortunately, we do not know of any really efficient algorithm for doing this.

One expensive approach is to compute a reliable singular value decomposition of M, since $\|M\|$ coincides with the largest singular value. Unfortunately, this usually boils down to the resolution of the eigenproblem associated to $M^*\, M$, with a few possible improvements (for instance, the dependency of the singular values on the coefficients of M is less violent than in the case of a general eigenproblem).

Since we only need the largest singular value, a faster approach is to reduce the computation of $\|M\|$ to the computation of $\|M^*\, M\|$, using the formula

$$\|M\| = \sqrt{\|M^*\, M\|}.$$

Applying this formula k times and using a naive bound at the end, we obtain

$$\|M\| \leqslant \sqrt[2^k]{n\, \|(M^*\, M)^{2^{k-1}}\|_\infty}.$$

This bound has an accuracy of $\approx k - O(\log_2 n)$ bits. Since $M^*\, M$ is symmetric, the $k - 1$ repeated squarings of $M^*\, M$ only correspond to about $\frac{k}{2}$ matrix multiplications. Notice also that it is wise to renormalize matrices before squaring them, so as to avoid overflows and underflows.

The approach can be speeded up further by alternating steps of tridiagonalization and squaring. Indeed, for a symmetric tridiagonal matrix D, the computation of D^2 and its tridiagonalization only take $O(n)$ steps instead of $O(n^3)$ for a full matrix product. After a few $k = O(n^2)$ steps of this kind, one obtains a good approximation μ of $\|D\|$. One may complete the algorithm by applying an algorithm with quadratic convergence for finding the smallest eigenvalues of $D - \mu$. In the lucky case when D has an isolated maximal eigenvalue, a certification of this method will provide sharp upper bounds for $\|D\|$ in reasonable time.

Even after the above improvements, the computation of sharp upper bounds for $\|M\|$ remains quite more expensive than ordinary matrix multiplication. For this reason, it is probably wise to avoid ball arithmetic in $\mathcal{B}(\mathbb{D}^{n\times n}, \mathbb{D})$ except if there are good reasons to expect that the improved quality is really useful for the application in mind.

Moreover, when using ball arithmetic in $\mathcal{B}(\mathbb{D}^{n\times n}, \mathbb{D}^{n\times n})$, it is often possible to improve algorithms in ways to reduce overestimation. When interpreting a complete computation as a dag, this can be achieved by minimizing the depth of the dag, i.e. by using an algorithm which is better suited for parallelization. Let us illustrate this idea for the computation of the k-th power M^k of a matrix M. When using Horner's method (multiply the identity matrix k times by M), we typically observe an overestimation of $O(k)$ bits (as for the example (19)). If we use binary powering, based on the rule

$$M^k = M^{\lfloor k/2 \rfloor} M^{\lceil k/2 \rceil}, \tag{37}$$

then the precision loss drops down to $O(\log k)$ bits. We will encounter a less obvious application of the same idea in section 7.5 below.

7. Reliable power series arithmetic

7.1. Ball series versus serial balls

There are two typical applications of power series $f \in \mathbb{R}[[z]]$ or $f \in \mathbb{C}[[z]]$ with certified error bounds. When f occurs as a generating function in a counting problem or random object generator, then we are interested in the computation of the coefficients f_n for large n, together with reliable error bounds. A natural solution is to systematically work with computable power series with ball coefficients in $\mathbb{B}_p[[z]]^{\text{com}}$. For many applications, we notice that p is fixed, whereas $n \gg p$ may become very large.

The second typical application is when $f \in \mathbb{C}[[z]]$ is the local expansion of an analytic function on a disk $\mathcal{B}(0, \rho)$ and we wish to evaluate f at a point z with $|z| < \rho$. The geometric decrease of $|f_n z^n|$ implies that we will need only $n = O(p)$ coefficients of the series. In order to bound the remaining error using Cauchy's formula, we do not only need bounds for the individual coefficients f_n, but also for the norm $\|f\|_\rho$ defined in (4). Hence, it is more natural to work with serial balls in $\mathcal{B}(\mathbb{D}_p[\mathrm{i}][[z]]^{\text{com}}, \mathbb{D}_{64})$, while using the $\|\cdot\|_\rho$ norm. Modulo a rescaling $f(z) \mapsto f(\rho z)$, it will be convenient to enforce $\rho = 1$. In order to compute sharp upper bounds $\lceil\!\lceil f \rceil\!\rceil$ for $\|f\| = \|f\|_1$, it will also be convenient to have an algorithm which computes bounds $\lceil\!\lceil f_{n;} \rceil\!\rceil$ for the tails

$$f_{n;} = f_n z^n + f_{n+1} z^{n+1} + \cdots.$$

Compared to the computation of the corresponding head

$$f_{;n} = f_0 + \cdots + f_{n-1} z^{n-1},$$

we will show in section 7.4 that the computation of such a tail bound is quite cheap.

Again the question arises how to represent $f_{;n}$ in a reliable way. We may either store a global upper bound for the error, so that $f_{;n} \in \mathcal{B}(\mathbb{D}_p[\mathrm{i}][z], \mathbb{D}_{64})$, or compute individual bounds for the errors, so that $f_{;n} \in \mathbb{B}[\mathrm{i}]_p[z]$. If our aim is to evaluate f at a point z with $|z| \approx 1$, then both representations $P \in \mathbb{B}[\mathrm{i}]_p[z]$ and $\hat{P} = \mathcal{B}(P_c, \|P_r\|_\infty) \in \mathcal{B}(\mathbb{D}_p[\mathrm{i}][z], \mathbb{D}_{64})$ give rise to evaluations $P(z), \hat{P}(z) \in \mathbb{B}[\mathrm{i}]_p$ with equally precise error bounds. Since the manipulation of global error bounds is more efficient, the corresponding representation should therefore be preferred in this case. In the multivariate case, one has the additional benefit that "small" coefficients $f_i z^i$ (e.g. $|f_i| \leqslant \epsilon_p \|f\|$) can simply be replaced by a global error $\mathcal{B}(0, |f_i|)$, thereby increasing the sparsity of f. On the other hand, individual error bounds admit the advantage that rescaling $f(z) \mapsto f(\lambda z)$ is cheap. If we suddenly find out that f is actually convergent on a larger disk and want to evaluate f at a point z with $|z| > 1$, then we will not have to recompute the necessary error bounds for $f_{;n}$ from scratch.

Serial ball representations similar to what has been described above are frequently used in the area of Taylor models [MB96, MB04] for the validated long term integration of dynamical systems. In the case of Taylor models, there is an additional twist: given a dynamical system of dimension d, we not only compute a series expansion with respect to the time t, but also with respect to small perturbations $\varepsilon_1, \ldots, \varepsilon_d$ of the initial conditions. In particular, we systematically work with power series in several variables. Although such computations are more expensive, the extra information may be used in order to increase the sharpness of the computed bounds. A possible alternative is to compute the expansions in $\varepsilon_1, \ldots, \varepsilon_d$ only up to the first order and to use binary splitting for the multiplication of the resulting Jacobian matrices on the integration path. This approach will be detailed in section 7.5.

7.2. Reliable multiplication of series and polynomials

In order to study the reliable multiplication of series f and g, let us start with the case when $f, g \in \mathcal{B}(\mathbb{D}_p[[z]]^{\mathrm{com}}, \mathbb{D}_{64})$, using the sup-norm on the unit disk. We may take

$$h = f g = \mathcal{B}(\widetilde{f_c g_c}, [\|f_c\| \|g_r\| + \|f_r\| (\|g_c\| + \|g_r\|) + \delta]^{\uparrow}),$$

where $h_c = \widetilde{f_c g_c}$ stands for a δ-approximation of $f_c g_c$. Since h_c is really a numeric algorithm for the computation of its coefficients, the difficulty resides in the fact that δ has to be chosen once and for all, in such a way that the bound $\|h_c - f_c g_c\|$ will be respected at the limit. A reasonable choice is $\delta = \epsilon_p \|f_c\| \|g_c\|$. We next distribute this error over the infinity of coefficients: picking some $\alpha < 1$, each coefficient $(h_c)_n$ is taken to be an $[(1 - \alpha) \alpha^n \delta]$-approximation of $f_c g_c$. Of course, these computation may require a larger working precision than p. Nevertheless, f and g are actually convergent on a slightly larger disk $\mathcal{B}(0, \rho)$. Picking $\alpha = 1/\rho$, the required increase of the working precision remains modest.

Let us now turn our attention to the multiplication of two computable series $f, g \in \mathbb{B}_p[[z]]^{\mathrm{com}}$ with ball coefficients. Except for naive power series multiplication, based on the formula $(f g)_n = \sum_{i+j=n} f_i g_j$, most other multiplication algorithms (whether relaxed or not) use polynomial multiplication as a subalgorithm. We are thus left with the problem of finding an efficient and high quality algorithm for multiplying two polynomials $P, Q \in \mathbb{B}_p[z]$ of degrees $< n$. In order to simplify the reading, we will assume that $P_0, P_{n-1}, Q_0, Q_{n-1}$ are all non-zero.

As in the case of matrix multiplication, there are various approaches with different qualities, efficiencies and aptitudes to profit from already available fast polynomial arithmetic in $\mathbb{D}_p[z]$. Again, the naive $O(n^2)$ approach provides almost optimal numerical stability and qualities for the error bounds. However, this approach is both slow from an asymptotic point of view and unable to rely on existant multiplication algorithms in $\mathbb{D}_p[z]$.

If the coefficients of P and Q are all of the same order of magnitude, then we may simply convert P and Q into polynomial balls in $\mathcal{B}(\mathbb{D}_p[z], \mathbb{D}_{64})$ for the norm $\|\cdot\|_\infty$ and use the following crude formula for their multiplication:

$$PQ \;=\; \mathcal{B}(\widetilde{P_c Q_c}, [n\,\|P_c\|_\infty\,\|Q_r\|_\infty + n\,\|P_r\|_\infty\,(\|Q_c\|_\infty + \|Q_r\|_\infty) + \delta]^{\uparrow}), \tag{38}$$

where $\widetilde{P_c Q_c}$ stands for a δ-approximation of $P_c\,Q_c$. In other words, we may use any efficient multiplication algorithm in $\mathbb{D}_p[z]$ for the approximation of $P_c\,Q_c$, provided that we have a means to compute a certified bound δ for the error.

In our application where P and Q correspond to ranges of coefficients in the series f and g, we usually have $P_i \approx P_0\,\rho_f^{-i}$ and $Q_i \approx Q_0\,\rho_g^{-i}$ for the convergence radii ρ_f and ρ_g of P and Q. In order to use (38), it is therefore important to scale $P(z) \mapsto P(z/\rho)$ and $Q(z) \mapsto Q(z/\rho)$ for a suitable ρ. If we are really interested in the evaluation of $f\,g$ at points z in a disk $\mathcal{B}(0, r)$, then we may directly take $\rho = r$. In we are rather interested in the coefficients of $f\,g$, then $\rho = \min(\rho_f, \rho_g)$ is the natural choice. However, since ρ_f and ρ_g are usually unknown, we first have to compute suitable approximations for them, based on the available coefficients of P and Q. A good heuristic approach is to determine indices $i < n/2 \leqslant j$ such that

$$\left|\frac{P_k}{P_i}\right|^{j-i} \;\leqslant\; \left|\frac{P_j}{P_i}\right|^{k-i},$$

for all k, and to take

$$\rho_f \;\approx\; \left|\frac{P_i}{P_i}\right|^{\frac{1}{j-i}}$$

as the approximation for ρ_f. Recall that the numerical Newton polygon of N_P is the convex hull of all points $(i, \log|P_i| - \lambda) \in \mathbb{R}^2$ with $\lambda \geqslant 0$. Consider the edge of N_P whose projection on the x-axis contains $n/2$. Then i and j are precisely the extremities of this projection, so they can be computed in linear time.

For large precisions $n = O(p)$, the scaling algorithm is both very efficient and almost of an optimal quality. For small p and large n, there may be some precision loss which depends on the nature of the smallest singularities of f and g. Nevertheless, for many singularities, such as algebraic singularities, the precision loss is limited to $O(\log n)$ bits. For a more detailed discussion, we refer to [vdH02, Section 6.2].

In other applications, where P and Q are not obtained from formal power series, it is usually insufficient to scale using a single factor ρ. This is already the case when multiplying $P = 1 + z$ and $Q = 1 + \delta\,z$ for small $\delta \ll \epsilon_p$, since the error bound for $(PQ)_2 = \delta$ exceeds ϵ_p. One possible remedy is to “precondition” P and Q according to their numerical Newton polygons and use the fact that N_{PQ} is close to the Minkowski sum $N_P + N_Q$.

More precisely, for each i, let $(N_P)_i > 0$ denote the number such that $(i, \log(N_P)_i)$ lies on one of the edges of N_P. Then $(N_P)_i$ is of the same order of magnitude as P_i, except for indices for which P_i is small “by accident”. Now consider the preconditioned relative error

$$\nu_P \;=\; \max_i \frac{(P_i)_r}{(N_P)_i}$$

and similarly for Q. Then

$$[(PQ)_i]_r \;\leqslant\; n\,(\nu_P + \nu_Q + \nu_P \nu_Q)\,(N_P + N_Q)_i, \tag{39}$$

if PQ is computed using infinite precision ball arithmetic. Assuming a numerically stable multiplication algorithm for the centers, as proposed in [vdH08], and incorporating the corresponding bound for the additional errors into the right hand side of (39), we thus obtain an efficient way to compute an upper bound for $(PQ)_r$.

Notice that the numerical Newton polygon N_P has a close relation to the orders of magnitudes of the roots of P. Even though the error bounds for some “accidentally small” coefficients $(P Q)_i$ may be bad for the above method, the error bounds have a good quality if we require them to remain valid for small perturbations of the roots of $P Q$.

7.3. Reliable series division and exponentiation

The algorithms from the previous section can in particular be used for the relaxed multiplication of two ball power series $f, g \in \mathbb{B}_p[[z]]^{\text{com}}$. In particular, using the implicit equation

$$g = 1 + z f g, \tag{40}$$

this yields a way to compute $g = (1 - z f)^{-1}$. Unfortunately, the direct application of this method leads to massive overestimation. For instance, the computed error bounds for the inverses of

$$\begin{aligned} g &= \frac{1}{1 - 3 z - 2 z^2} \\ h &= \frac{1}{1 - 3 z + 2 z^2} \end{aligned}$$

coincide. Indeed, even when using a naive relaxed multiplication, the coefficients of g and h are computed using the recurrences

$$\begin{aligned} g_n &= 3 g_{n-1} + 2 g_{n-2} \\ h_n &= 3 h_{n-1} - 2 h_{n-2}, \end{aligned}$$

but the error bound ε_n for g_n and h_n is computed using the same recurrence

$$\varepsilon_n = 3 \varepsilon_{n-1} + 2 \varepsilon_{n-2},$$

starting with $\varepsilon_0 \approx \epsilon_p$. For $n \to \infty$, it follows that $g_n \sim \lambda \alpha^{-n}$ and $\varepsilon_n \sim \lambda \epsilon_p \alpha^{-n}$ for some λ, where $\alpha \approx 0.281$ is the smallest root of $1 - 3 \alpha - 2 \alpha^2$. Hence the error bound for g_n is sharp. On the other hand, $h_n \sim \mu \beta^n$ for some μ, where $\beta = \frac{1}{2}$ is the smallest root of $1 - 3 \beta + 2 \beta^2$. Hence, the error bound ε_n is $n \log_2 (\beta / \alpha)$ bits too pessimistic in the case of h. The remedy is similar to what we did in the case of matrix inversion. We first introduce the series

$$\begin{aligned} \varphi &= \frac{1}{1 - z f_c} \in \mathbb{D}_p[[z]]^{\text{com}} \\ \psi &= \frac{[\varphi (z f - 1)]_{1;}}{z} \in \mathbb{B}_p[[z]]^{\text{com}} \end{aligned}$$

and next compute g using

$$g = \frac{\varphi}{1 - z \psi},$$

where $1 - z \psi$ is inverted using the formula (40). This approach has the advantage of being compatible with relaxed power series expansion and it yields high quality error bounds.

Another typical operation on power series is exponentiation. Using relaxed multiplication, we may compute the exponential g of an infinitesimal power series $f \in z \mathbb{B}_p[[z]]^{\text{com}}$ using the implicit equation

$$g = 1 + \int f' g, \tag{41}$$

where $\int$ stands for "distinguished integration" in the sense that $(\int h)_0 = 0$ for all h. Again, one might fear that this method leads to massive overestimation. As a matter of fact, it usually does not. Indeed, assume for simplicity that $f_r = 0$, so that $f \in \mathbb{D}_p[[z]]$. Recall that $|f|$ denotes the power series with $|f|_n = |f_n|$. Roughly speaking, the error bound for g_n, when computed using the formula (41) will be the coefficient ε_n of the power series $\varepsilon = \exp(\epsilon_p |f|)$. Since $|f|$ has the same radius of convergence as f, it directly follows that the bit precision loss is sublinear $o(n)$. Actually, the dominant singularity of $|f|$ often has the same nature as the dominant singularity of f. In that case, the computed error bounds usually become very sharp. The observation generalizes to the resolution of linear differential equations, by taking a square matrix of power series for f.

7.4. Automatic tail bounds

In the previous sections, we have seen various reliable algorithms for the computation of the expansion $f_{;n}$ of a power series $f \in \mathbb{C}[[z]]$ up till a given order n. Such expansions are either regarded as ball polynomials in $\mathbb{B}[\mathrm{i}]_p[z]$ or as polynomial balls in $\mathcal{B}(\mathbb{D}_p[\mathrm{i}][z], \mathbb{D}_{64})$. Assuming that f is convergent on the closed unit disk $\mathcal{B}(0,1)$, it remains to be shown how to compute tail bounds $\lceil\!\lceil f_{n;} \rceil\!\rceil$. We will follow [vdH07b]. Given a ball polynomial P, then we notice that reasonably sharp upper and lower bounds $\lceil\!\lceil P \rceil\!\rceil$ and $\lfloor\!\lfloor P \rfloor\!\rfloor$ for P can be obtained efficiently by evaluating P at $O(\deg P)$ primitive roots of unity using fast Fourier transforms [vdH07b, Section 6.2].

We will assume that the series f is either an explicit series, the result of an operation on other power series or the solution of an implicit equation. Polynomials are the most important examples of explicit series. Assuming that f is a polynomial of degree $< d$, we may simply take

$$\lceil\!\lceil f_{n;} \rceil\!\rceil \;=\; \begin{cases} \lceil\!\lceil z^{-n} f_{n;d} \rceil\!\rceil & \text{if } n < d \\ 0 & \text{otherwise} \end{cases}, \tag{42}$$

where

$$f_{n;d} \;=\; f_n z^n + \cdots + f_{d-1} z^{d-1}.$$

For simple operations on power series, we may use the following bounds:

$$\lceil\!\lceil (f+g)_{;n} \rceil\!\rceil \;=\; \lceil\!\lceil f_{;n} \rceil\!\rceil + \lceil\!\lceil g_{;n} \rceil\!\rceil \tag{43}$$

$$\lceil\!\lceil (f-g)_{;n} \rceil\!\rceil \;=\; \lceil\!\lceil f_{;n} \rceil\!\rceil + \lceil\!\lceil g_{;n} \rceil\!\rceil \tag{44}$$

$$\lceil\!\lceil (f g)_{;n} \rceil\!\rceil \;=\; \lceil\!\lceil f_{n;} \rceil\!\rceil \left(\lceil\!\lceil g_{;n} \rceil\!\rceil + \lceil\!\lceil g_{n;} \rceil\!\rceil\right) + \lceil\!\lceil f_{;n} \rceil\!\rceil \, \lceil\!\lceil g_{n;} \rceil\!\rceil + \lceil\!\lceil (f_{;n} \, g_{;n})_{n;} \rceil\!\rceil \tag{45}$$

$$\lceil\!\lceil (\textstyle\int f)_{n;} \rceil\!\rceil \;=\; \frac{1}{n+1} \lceil\!\lceil f_{n;} \rceil\!\rceil, \tag{46}$$

where

$$\lceil\!\lceil (f_{;n} \, g_{;n})_{n;} \rceil\!\rceil \;\leqslant\; \sum_{k=0}^{n-1} |f_k| \left(\sum_{l=n-k}^{n-1} |g_l| \right)$$

can be computed in time $O(n)$. Due to possible overestimation, division has to be treated with care. Given an infinitesimal power series ε and

$$f \;=\; \frac{1}{1-\varepsilon},$$

so that

$$f_{n;} \;=\; \frac{1 + \varepsilon \, f_{;n} - f_{;n}}{1-\varepsilon},$$

we take

$$\lceil\!\lceil f_{n;}\rceil\!\rceil \;=\; \frac{\lceil\!\lceil (\varepsilon\, f_{;n})_{n;}\rceil\!\rceil}{\lfloor\!\lfloor 1-\varepsilon_{;n}+\mathcal{B}(0,\lceil\!\lceil \varepsilon_{n;}\rceil\!\rceil)\rfloor\!\rfloor}, \tag{47}$$

where $\lceil\!\lceil (\varepsilon\, f_{;n})_{n;}\rceil\!\rceil$ is computed using (45).

Let $\Phi(f)$ be an expression which is constructed from f and polynomials, using the ring operations and distinguished integration (we exclude division in order to keep the discussion simple). Assume that each coefficient $\Phi(f)_n$ only depends on the previous coefficients $f_0,\ldots,f_{n-1}$ of f and not on $f_n, f_{n-1},\ldots$. Then the sequence $0,\Phi(0),\Phi^2(0),\ldots$ tends to the unique solution f of the implicit equation

$$f \;=\; \Phi(f), \tag{48}$$

Moreover, $\Phi^k(0)_n = f_n$ for any $k>n$. In (40) and (41), we have already seen two examples of implicit equations of this kind.

The following technique may be used for the computation of tail bounds $\lceil\!\lceil f_{n;}\rceil\!\rceil$. Given $c\in\mathbb{D}^{\geqslant}$ and assuming that $\lceil\!\lceil f_{n;}\rceil\!\rceil\leqslant c$, we may use the rules (42–47) in order to compute a “conditional tail bound” $\lceil\!\lceil \Phi(f)_{n;}|c\rceil\!\rceil$ for $\|\Phi(f)_{n;}\|$. Mathematically speaking, this bound has the property that for any power series g with $g_{;n}=f_{;n}$ and $\|g_{n;}\|\leqslant c$, we have $\|\Phi(g)_{n;}\|\leqslant\lceil\!\lceil \Phi(g)_{n;}|c\rceil\!\rceil$. If $\lceil\!\lceil \Phi(g)_{n;}|c\rceil\!\rceil\leqslant c$, then it follows that $\|[\Phi^{(k)}(f_{;n})]_{n;}\|\leqslant c$ for all k. Given any disk $\mathcal{B}(0,\rho)$ with $\rho<1$, it follows that $\|\Phi^{(k)}(f_{;n})-f\|_\rho\leqslant 2\,c\,\rho^k/(1-\rho)$ for any $k\geqslant n$. Indeed, setting $\delta=\Phi^{(k)}(f_{;n})-f$, we get from $\|\delta\|\leqslant 2\,c$ and $\delta=O(z^k)$ that $\|\delta\|_\rho\leqslant|\delta_k|\,\rho^k+|\delta_{k+1}|\,\rho^{k+1}+\cdots\leqslant 2\,c\,\rho^k/(1-\rho)$. In other words, $\Phi^{(k)}(f_{;n})$ uniformly converges to f on $\mathcal{B}(0,\rho)$. Therefore, $\|f_{n;}\|_\rho\leqslant c$ for any $\rho<1$. Modulo application of the method on a disk of radius slightly larger than one and scaling back, this yields a way to compute a certified tail bound $\lceil\!\lceil f_{n;}\rceil\!\rceil$.

Let us now consider the mapping $\varphi\colon c\mapsto\lceil\!\lceil \Phi(f)_{n;}|c\rceil\!\rceil$ and assume that Φ involves no divisions. When computing $\lceil\!\lceil \Phi(f)_{n;}|c\rceil\!\rceil$ using infinite precision and the rules (42–45), we notice that φ is a real analytic function, whose power series expansion contains only positive coefficients. Finding the smallest c with $\varphi(c)\leqslant c$ thus reduces to finding the smallest fixed point c_{fix} of φ, if such a fixed point exists. We may use the secant method:

$$\begin{aligned} c_0 &:= 0\\ c_1 &:= \varphi(c_0)\\ c_{k+2} &:= c_k+\frac{\varphi(c_k)-c_k}{c_{k+1}-\varphi(c_{k+1})+\varphi(c_k)-c_k}\,(c_{k+1}-c_k)\end{aligned}$$

If $c_{k+1}<c_k$ for some k or if k exceeds a given threshold K, then the method fails and we set $\lceil\!\lceil f_{n;}\rceil\!\rceil=+\infty$. Otherwise, c_k converges quadratically to c_{fix}. As soon as $|c_{k+1}/c_k-1|<\varepsilon$, for some given $\varepsilon>0$, we check whether $\varphi(\tilde c_{\text{fix}})\leqslant\tilde c_{\text{fix}}$ for $\tilde c_{\text{fix}}=2\,c_{k+1}-c_k$, in which case we stop. The resulting $\tilde c_{\text{fix}}$ is an approximation of c_{fix} with relative accuracy $\varepsilon>0$.

Assuming that $\Psi(f)_{;n}$ has been computed for every subexpression of Φ, we notice that the computation of $\lceil\!\lceil \Phi(f)_{n;}|c\rceil\!\rceil$ only requires $O(n\,s)$ floating point operations, where s is the size of Φ as an expression. More generally, the evaluation of $\lceil\!\lceil \Phi(f)_{n;}|c_i\rceil\!\rceil$ for k different hypotheses $c_1,\ldots,c_k$ only requires $O(n\,s\,k)$ operations, since the heads $\Psi(f)_{;n}$ do not need to be recomputed. Our algorithm for the computation of $\tilde c_{\text{fix}}$ therefore requires at most $O(n\,s\,K)$ floating point operations. Taking $K=o(\mathsf{R}(n)/n)$, it follows that the cost of the tail bound computation remains negligible with respect to the series expansion itself.

The approach generalizes to the case when f is a vector or matrix of power series, modulo a more involved method for the fixed point computation [vdH07b, Section 6.3]. If f is indeed convergent on $\mathcal{B}(0,1)$, then it can also be shown that φ indeed admits a fixed point if n is sufficiently large.

7.5. Long term integration of dynamical systems

Let us now consider a dynamical system

$$f \;=\; f(0) + \textstyle\int \Phi(f), \tag{49}$$

where f is a vector of d unknown complex power series, $f(0) \in \mathbb{B}[\mathrm{i}]_p^d$, and Φ an expression built up from the entries of f and polynomials in $\mathbb{D}_p[\mathrm{i}][z]$ using the ring operations. Given $t \in \mathbb{D}[\mathrm{i}] \setminus \{0\}$, denote by $f_{z \times t}$ the scaled power series $f(t\,z)$ and by f_{z+t} the shifted power series $f(t+z)$, assuming that f converges at t. Also denote by $\Phi_{z \times t}$ and Φ_{z+t} the expressions which are obtained when replacing each polynomial P in Φ by $P_{z \times t}$ resp. P_{z+t}. Then $f_{z \times t}$ and f_{z+t} satisfy

$$f_{z \times t} \;=\; f(0) + t \textstyle\int \Phi_{z \times t}(f_{z \times t}) \tag{50}$$

$$f_{z+t} \;=\; f(t) + \textstyle\int \Phi_{z+t}(f_{z+t}) \tag{51}$$

Since (49) is a particular case of an implicit equation of the form (48), we have an algorithm for the computation of tail bounds $\lceil\!\lceil f_{n;} \rceil\!\rceil \in (\mathbb{D}_{64}^{\geqslant} \cup \{\infty\})^d$ for f on $\mathcal{B}(0,\,1)$. Modulo rescaling (50), we may also compute tail bounds $\lceil\!\lceil f_{n;} \rceil\!\rceil_\rho$ on more general disks $\mathcal{B}(0,\rho)$.

Assume that we want to integrate (49) along the real axis, as far as possible, and performing all computations with an approximate precision of p bits. Our first task is to find a suitable initial step size t and evaluate f at t. Since we require a relative precision of approximately p bits, we roughly want to take $t \approx \rho_f / 2$, where ρ_f is the radius of convergence of f, and evaluate the power series f up to order $n \approx p$. We thus start by the numerical computation of $f_{n;}$ and the estimation $\tilde{\rho}_f$ of ρ_f, based on the numerical Newton polygon of $f_{n;}$. Setting $t := \tilde{\rho}_f / 2$, we next compute a tail bound $\lceil\!\lceil f_{n;} \rceil\!\rceil_t$. This bound is considered to be of an acceptable quality if

$$\lceil\!\lceil f_{n;} \rceil\!\rceil_t \;\leqslant\; \epsilon_p \, \lceil\!\lceil f_{;n} \rceil\!\rceil_t.$$

In the contrary case, we keep setting $t := t/2$ and recomputing $\lceil\!\lceil f_{n;} \rceil\!\rceil_t$, until we find a bound of acceptable quality. It can be shown [vdH07b] that this process ultimately terminates, when p is sufficiently large. We thus obtain an appropriate initial step size t which allows us to compute a δ-approximation of $f(t)$ with $\delta = 2\,\epsilon_p \, \lceil\!\lceil f_{;n} \rceil\!\rceil_t +^{\uparrow} (f_{;n}(t))_r$.

In principle, we may now perform the translation $z \mapsto t + z$ and repeat the analytic continuation process using the equation (51). Unfortunately, this approach leads to massive overestimation. Indeed, if the initial condition $f(0)$ is given with relative precision η, then the relative precisions of the computed coefficients f_k are typically a non trivial factor times larger than η, as well as the next "initial condition" $f(t)$ at t. Usually, we therefore lose $O(1)$ bits of precision at every step.

One remedy is the following. Let $\Delta = \Delta_{0,t}$ be the analytic continuation operator which takes an initial condition $c \in \mathbb{C}^d$ on input and returns the evaluation $f(t) \in \mathbb{C}^d$ of the unique solution to (49) with $f(0) = c$. Now instead of computing $f_{;n}$ using a ball initial condition $f(0) \in \mathbb{B}[\mathrm{i}]_p^d$, we rather use its center $f(0)_c$ as our initial condition and compute $\Delta(f(0)_c)$ using ball arithmetic. In order to obtain a reliable error bound for $f(t)$, we also compute the Jacobian J_Δ of Δ at $f(0)_c$ using ball arithmetic, and take

$$f(t) \;=\; \Delta(f(0)_c) + \mathcal{B}(0, [J_\Delta(f(0)_c)\, f(0)_r]^{\uparrow}).$$

The Jacobian can either be computed using the technique of automatic differentiation [BS83] or using series with coefficients in a jet space of order one.

With this approach $\Delta(f(0)_c)$ is computed with an almost optimal p-bit accuracy, and $J_{\Delta}(f(0)_c)$ with an accuracy which is slightly worse than the accuracy of the initial condition. In the lucky case when $J_{\Delta}(f(0)_c)$ is almost diagonal, the accuracy of $f(t)$ will therefore be approximately the same as the accuracy of $f(0)$. However, if $J_{\Delta}(f(0)_c)$ is not diagonal, such as in (19), then the multiplication $J_{\Delta}(f(0)_c)\ f(0)_r$ may lead to overestimation. This case may already occur for simple linear differential equations such as

$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \int \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix} \tag{52}$$

and the risk is again that we lose $O(1)$ bits of precision at every step.

Let us describe a method for limiting the harm of this manifestation of the wrapping effect. Consider the analytic continuations of f at successive points $0 = t_0 < t_1 < \cdots < t_k$ and denote

$$J_i = J_{\Delta_{t_{i-1}, t_i}}(f(t_{i-1})_c), \qquad i = 1, \ldots, k.$$

Instead of computing the error at t_i due to perturbation of the initial condition $f(0)$ using the formula

$$[f(t)_r]^{\mathrm{per}(0)} = J_k\,(J_{k-1}\,(\ldots J_1\, f(0)_r \ldots)),$$

we may rather use the formula

$$[f(t)_r]^{\mathrm{per}(0)} = (J_k\, J_{k-1} \cdots J_1)\, f(0)_r,$$

where matrix chain products are computed using a variant of binary powering (37):

$$J_j \cdots J_i = (J_j \cdots J_{\lfloor (i+j)/2 \rfloor + 1})\, (J_{\lfloor (i+j)/2 \rfloor} \cdots J_i).$$

In order to be complete, we also have to take into account the additional error δ_i, which occurs during the computation of $\Delta_{t_{i-1}, t_i}(f(t_{i-1})_c)$. Setting $\delta_0 = f(0)_r$, we thus take

$$f(t_i)_r = \delta_i + J_i\, \delta_{i-1} + \cdots + (J_k \cdots J_1)\, \delta_0. \tag{53}$$

When using this algorithm, at least in the case of simple systems such as (52), the precision loss at step k will be limited to $O(\log k)$ bits. Notice that we benefit from the fact that the Jacobian matrices remain accurate as long as the initial conditions remain accurate.

Although we have reduced the wrapping effect, the asymptotic complexity of the above algorithm is cubic or at least quadratic in k when evaluating (53) using a binary splitting version of Horner's method. Let us now describe the final method which requires only $O(k \log k)$ matrix multiplications up till step k. For $0 \leqslant i < j$, we define

$$\begin{aligned} \delta_{i,j} &= \delta_{j-1} + J_{j-1}\, \delta_{j-2} + \cdots + (J_{j-1} \cdots J_{i+1})\, \delta_i. \\ J_{i,j} &= J_{j-1} \cdots J_i, \end{aligned}$$

where $J_0 = 1$. At stage $i = 2^{e_1} + \cdots + 2^{e_l}$ with $e_1 > \cdots > e_l$, we assume that

$$\begin{aligned} \delta_{[j]} &:= \delta_{[i;j]} := \delta_{2^{e_1} + \cdots + 2^{e_{j-1}}, 2^{e_1} + \cdots + 2^{e_j}} \\ J_{[j]} &:= J_{[i;j]} := J_{2^{e_1} + \cdots + 2^{e_{j-1}}, 2^{e_1} + \cdots + 2^{e_j}} \end{aligned}$$

are stored in memory for all $1 \leqslant j \leqslant l$. From these values, we may compute

$$f(t_{i-1})_r = \delta_{[l]} + J_{[l]}\, (\delta_{[l-1]} + \cdots J_{[3]}\, (\delta_{[2]} + J_{[2]}\, \delta_{[1]}) \cdots)$$

using only $O(l) = O(\log i)$ matrix multiplications. Now assume that we go to the next stage $i + 1$. If m is maximal such that 2^m divides $i + 1$, then $i + 1 = 2^{e_1} + \cdots + 2^{e_{l-m}} + 2^m$. Consequently, $\delta_{[i+1;j]} = \delta_{[i;j]}$ and $J_{[i+1;j]} = J_{[i;j]}$ for $j < l' := l - m + 1$ and

$$\begin{aligned} \delta_{[i+1;l']} &= \delta_i + J_i\, (\delta_{[i;l]} + \cdots J_{[i;l'+2]}\, (\delta_{[i;l'+1]} + J_{[i;l'+1]}\, \delta_{[i;l']}) \cdots) \\ J_{[i+1;l']} &= J_i\, J_{[i;l]} \cdots J_{[i;l']}. \end{aligned}$$

Hence, the updated lists $\delta_{[i+1;j]}$ and $J_{[i+1;j]}$ can be computed using $O(m)=O(\log i)$ matrix multiplications. Furthermore, we only need to store $O(\log i)$ auxiliary matrices at step i.

7.6. Discussion

There is a vast literature on validated integration of dynamical systems and reduction of the wrapping effect [Moo65, Moo66, Nic85, Loh88, GS88, Neu93, K98, Loh01, Neu02, MB04]. We refer to [Loh01] for a nice review. Let us briefly discuss the different existing approaches.

The wrapping effect was noticed in the early days of interval arithmetic [Moo65] and local coordinate transformations were proposed as a remedy. The idea is to work as much as possible with respect to coordinates in which all errors are parallel to axes. Hence, instead of considering blocks $x\in\mathbb{B}_p^d$, we rather work with parallelepipeds $x=c+T\mathcal{B}(0,r)$, with $c\in\mathbb{D}_p^d$, $T\in\mathbb{D}_p^{d\times d}$ and $r\in\mathbb{D}_p^d$. A natural choice for T after k steps is $T=J_k\cdots J_1$, but more elaborate choices may be preferred [Loh88]. Other geometric shapes for the enclosures have been advanced in the literature, such as ellipsoids [Neu93], which are also invariant under linear transformations, and zonotopes [K98]. However, as long as we integrate (49) using a straightforward iterative method, and even if we achieve a small average loss $\varepsilon\ll 1$ of the bit precision at a single step, the precision loss after k steps will be of the form $k\,\varepsilon$.

The idea of using dichotomic algorithms in order to reduce the wrapping effect was first described in the case of linear differential equations [GS88]. The previous section shows how to adapt that technique to non linear equations. We notice that the method may very well be combined with other geometric shapes for the enclosures: this will help to reduce the precision loss to $(\log k)\,\varepsilon$ instead of $k\,\varepsilon$ in the above discussion.

Notice that there are two common misunderstandings about the dichotomic method. Contrary to what we have shown above (see also [GS88] in the linear case), it is sometimes believed that we need to keep all matrices $J_1,\ldots,J_k$ in memory and that we have to reevaluate products $J_j\cdots J_i$ over and over again. Secondly, one should not confuse *elimination* and *reduction* of the wrapping effect: if M is the 2×2 rotation matrix by a $30°$ angle, then none of its repeated squarings M^{2^l} will be the identity matrix, so every squaring $(M^{2^l})^2$ will involve a wrapping. Even though we have *not* eliminated the wrapping effect, we *did* achieve to reduce the number of wrappings to l instead of 2^l.

Taylor models are another approach for the validated long term integration of dynamical systems [EKW84, EMOK91, MB96, MB04]. The idea is to rely on higher k-th order expansions of the continuation operator Δ. This allows for real algebraic enclosures which are determined by polynomials of degree k. Such enclosures are *a priori* more precise for non linear problems. However, the method requires us to work in order k jet spaces in d variables; the mere storage of such a jet involves $\binom{d+k}{d}$ coefficients. From a theoretical point of view it is also not established that Taylor methods eliminate the wrapping effect by themselves. Nevertheless, Taylor models can be combined with any of the above methods and the non linear enclosures seem to be more adapted in certain cases. For a detailed critical analysis, we refer to [Neu02].

Let us finally investigate how sharp good error bounds actually may be. If $\rho_{f(0)}=\|f(0)_r\|/\|f(0)\|_c$ denotes the relative precision of the initial condition at the start and $\rho_{f(t)}=\|f(t)_r\|/\|f(t)\|_c$ the relative precision of the final result, then it is classical that

$$\begin{aligned}\rho_{f(t)} &\geqslant \kappa(J_\Delta(f(0)))\,\rho_{f(0)},\\ \kappa(J_\Delta(f(0))) &= \|J_\Delta(f(0))\|\,\|J_\Delta(f(0))^{-1}\|,\end{aligned}$$

where $\kappa(J_\Delta(f(0)))$ is the condition number of the matrix $J_\Delta(f(0))$. We propose to define the condition number $\kappa(\Phi,f(0),0,t)$ for the integration problem (49) on $[0,t]$ by

$$\mathrm{K}=\kappa(\Phi,f(0),0,t) = \max_{0\leqslant t_1\leqslant t_2\leqslant t}\kappa(J_{\Delta_{t_1,t_2}}(f(t_1))).$$

Indeed, without using any particular mathematical properties of Φ or f, we somehow have to go through the whole interval $[0,t]$. Of course, it may happen that Φ is indeed special. For instance, if $\Phi = M f$ for a matrix M with a special triangular or diagonal form, then special arguments may be used to improve error bounds and more dedicated condition numbers will have to be introduced.

However, in general, we suspect that a precision loss of $\log \mathrm{K}$ cannot be avoided. If K gets large, the only real way to achieve long term stability is to take $p > 2 \log \mathrm{K}$ (say), whence the interest of efficient multiple precision and high order ball algorithms. It seems to us that the parallelepiped method should manage to keep the precision loss bounded by $\log \mathrm{K}$, for $p > 2 \log \mathrm{K}$ and $\varepsilon \approx \epsilon_p$. The algorithm from section 7.5 (without coordinate transformations) only achieves a $\log k \log \mathrm{K}$ in the worst case, although a $\log \mathrm{K}$ bound is probably obtained in many cases of interest. We plan to carry out a more detailed analysis once we will have finished a first efficient multiple precision implementation.

Bibliography

[Abe80] O. Aberth. *Computable analysis*. McGraw-Hill, New York, 1980.

[AH83] G. Alefeld and J. Herzberger. *Introduction to interval analysis*. Academic Press, New York, 1983.

[ANS08] ANSI/IEEE. IEEE standard for binary floating-point arithmetic. Technical report, ANSI/IEEE, New York, 2008. ANSI-IEEE Standard 754-2008. Revision of IEEE 754-1985, approved on June 12, 2008 by IEEE Standards Board.

[ANS09] ANSI/IEEE. IEEE interval standard working group - p1788. `http://grouper.ieee.org/groups/1788/`, 2009.

[BB85] E. Bishop and D. Bridges. *Foundations of constructive analysis*. Die Grundlehren der mathematische Wissenschaften. Springer, Berlin, 1985.

[BCC+06] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of the 17th International Symposium on the Mathematical Theory of Networks and Systems*, pages 1269âĂŞ–1267, Kyoto, July 2006.

[BCO+06] A. Bostan, F. Chyzak, F. Ollivier, B. Salvy, É. Schost, and A. Sedoglavic. Fast computation of power series solutions of systems of differential equation. preprint, april 2006. submitted, 13 pages.

[Ber98] M. Berz. Cosy infinity version 8 reference manual. Technical Report MSUCL-1088, Michigan State University, East Lansing, 1998.

[BF00] D. A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, 23:127–173, 2000.

[BHL00] D.H. Bailey, Y. Hida, and X.S. Li. QD, a C/C++/Fortran library for double-double and quad-double arithmetic. `http://crd.lbl.gov/~dhbailey/mpdist/`, 2000.

[BHL01] D.H. Bailey, Y. Hida, and X.S. Li. Algorithms for quad-double precision floating point arithmetic. In *15th IEEE Symposium on Computer Arithmetic*, pages 155–162, June 2001.

[BHSW06] D. Bates, J. Hauenstein, A. Sommese, and C. Wampler. Bertini: Software for numerical algebraic geometry. `http://www.nd.edu/~sommese/bertini/`, 2006.

[BK78] R.P. Brent and H.T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25:581–595, 1978.

[BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[CK91] D.G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28:693–701, 1991.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Computat.*, 19:297–301, 1965.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc. of the* 19^{th} *Annual Symposium on Theory of Computing*, pages 1–6, New York City, may 25–27 1987.

[DGG+02a] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Linbox: A generic library for exact linear algebra. In *First Internat. Congress Math. Software ICMS*, pages 40–50, Beijing, China, 2002.

[DGG+02b] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Project linbox: Exact computational linear algebra. `http://www.linalg.org/`, 2002.

[ea67] U.W. Kulisch et al. Scientific computing with validation, arithmetic requirements, hardware solution and language support. `http://www.math.uni-wuppertal.de/~xsc/`, 1967.

[EKW84] J.-P. Eckmann, H. Koch, and P. Wittwer. A computer-assisted proof of universality in area-preserving maps. *Memoirs of the AMS*, 47(289), 1984.

[EMOK91] J.-P. Eckmann, A. Malaspinas, and S. Oliffson-Kamphorst. A software tool for analysis in function spaces. In K. Meyer and D. Schmidt, editors, *Computer aided proofs in analysis*, pages 147–166. Springer, New York, 1991.

[Für07] M. Fürer. Faster integer multiplication. In *Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing (STOC 2007)*, pages 57–66, San Diego, California, 2007.

[FZVC94] Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[GG02] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2-nd edition, 2002.

[Gra91] T. Granlund et al. GMP, the GNU multiple precision arithmetic library. `http://www.swox.com/gmp`, 1991.

[Grz57] A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–71, 1957.

[GS88] T.N. Gambill and R.D. Skeel. Logarithmic reduction of the wrapping effect with application to ordinary differential equations. *SIAM J. Numer. Anal.*, 25(1):153–162, 1988.

[Hai95] B. Haible. CLN,a Class Library for Numbers. `http://www.ginac.de/CLN/cln.html`, 1995.

[HLRZ00] G. Hanrot, V. Lefèvre, K. Ryde, and P. Zimmermann. MPFR, a C library for multiple-precision floating-point computations with exact rounding. `http://www.mpfr.org`, 2000.

[HS67] E. Hansen and R. Smith. Interval arithmetic in matrix computations, part ii. *Siam J. Numer. Anal.*, 4:1–9, 1967.

[JKDW01] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis*. Springer, London, 2001.

[K98] W. Kühn. Rigourously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61:47–67, 1998.

[KLRK97] V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations*. Springer, 1997.

[KO63] A. Karatsuba and J. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.

[KR06] V. Kreinovich and S. Rump. Towards optimal use of multi-precision arithmetic: a remark. Technical Report UTEP-CS-06-01, UTEP-CS, 2006.

[Krea] V. Kreinovich. Interval and related verified software. `http://www.cs.utep.edu/interval-comp/intsoft.html`. Software for interval computations.

[Kreb] V. Kreinovich. Interval computations. `http://www.cs.utep.edu/interval-comp/`. Useful information and references on interval computations.

[Kul08] U.W. Kulisch. *Computer Arithmetic and Validity. Theory, Implementation, and Applications*. Number 33 in Studies in Mathematics. de Gruyter, 2008.

[Lam07] B. Lambov. Reallib: An efficient implementation of exact real arithmetic. *Mathematical Structures in Computer Science*, 17(1):81–98, 2007.

[Lam08] B. Lambov. Interval arithmetic using sse-2. In Peter Hertling, Christoph Hoffmann, Wolfram Luther, and Nathalie Revol, editors, *Reliable Implementation of Real Number Algorithms: Theory and Practice*, volume 5045 of *Lecture Notes in Computer Science*, pages 102–113. Springer Berlin / Heidelberg, 2008.

[Loh88] R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendugen*. PhD thesis, Universität Karlsruhe, 1988.

[Loh01] R. Lohner. On the ubiquity of the wrapping effect in the computation of error bounds. In U. Kulisch, R. Lohner, and A. Facius, editors, *Perspectives on enclosure methods*, pages 201–217, Wien, New York, 2001. Springer.

[M0] N. Müller. iRRAM, exact arithmetic in C++. `http://www.informatik.uni-trier.de/iRRAM/`, 2000.

[MB96] K. Makino and M. Berz. Remainder differential algebras and their applications. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational differentiation: techniques, applications and tools*, pages 63–74, SIAM, Philadelphia, 1996.

[MB04] K. Makino and M. Berz. Suppression of the wrapping effect by Taylor model-based validated integrators. Technical Report MSU Report MSUHEP 40910, Michigan State University, 2004.

[MKC09] R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis*. SIAM Press, 2009.

[Moo65] R.E. Moore. Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions to ordinary differential equation. In L.B. Rall, editor, *Error in Digital Computation*, volume 2, pages 103–140. John Wiley, 1965.

[Moo66] R.E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, N.J., 1966.

[Mul06] Jean-Michel Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser Boston, Inc., 2006.

[Neu90] A. Neumaier. *Interval methods for systems of equations*. Cambridge university press, Cambridge, 1990.

[Neu93] A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confedence regions. *Computing Supplementum*, 9:175–190, 1993.

[Neu02] A. Neumaier. Taylor forms - use and limits. *Reliable Computing*, 9:43–79, 2002.

[Nic85] K. Nickel. How to fight the wrapping effect. In Springer-Verlag, editor, *Proc. of the Intern. Symp. on interval mathematics*, pages 121–132, 1985.

[OOO11] K. Ozaki, T. Ogita, and S. Oishi. Tight and efficient enclosure of matrix multiplication by using optimized blas. *Numerical Linear Algebra with Applications*, 18:237–248, 2011.

[Pan84] V. Pan. *How to multiply matrices faster*, volume 179 of *Lect. Notes in Math.* Springer, 1984.

[PTVF07] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical recipes, the art of scientific computing*. Cambridge University Press, 3rd edition, 2007.

[Rev01] N. Revol. MPFI, a multiple precision interval arithmetic library. `http://perso.ens-lyon.fr/nathalie.revol/software.html`, 2001.

[Rum99a] S.M. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.

[Rum99b] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tu-harburg.de/rump/`.

[Rum10] S.M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Texts in Applied Mathematics; 12. Springer, New York, third edition, 2002.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:352–356, 1969.

[Tur36] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Maths. Soc.*, 2(42):230–265, 1936.

[vdH02] J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.

[vdH06] J. van der Hoeven. Computations with effective real numbers. *TCS*, 351:52–60, 2006.

[vdH07a] J. van der Hoeven. New algorithms for relaxed multiplication. *JSC*, 42(8):792–802, 2007.

[vdH07b] J. van der Hoeven. On effective analytic continuation. *MCS*, 1(1):111–175, 2007.

[vdH08] J. van der Hoeven. Making fast multiplication of polynomials numerically stable. Technical Report 2008-02, Université Paris-Sud, Orsay, France, 2008.

[vdH09] J. van der Hoeven. Ball arithmetic. Technical report, HAL, 2009. `http://hal.archives-ouvertes.fr/hal-00432152/fr/`.

[vdH10] J. van der Hoeven. Newton's method and FFT trading. *JSC*, 45(8):857–878, 2010.

[vdHLM+02] J. van der Hoeven, G. Lecerf, B. Mourain, et al. Mathemagix, 2002. `http://www.mathemagix.org`.

[vdHMT] J. van der Hoeven, B. Mourrain, and Ph. Trébuchet. Efficient and reliable resolution of eigenproblems. In preparation.

[Ver99] J. Verschelde. PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software*, 25(2):251–276, 1999. Algorithm 795.

[vG02] J.W. von Gudenberg. Interval arithmetic on multimedia architectures. *Reliable Computing*, 8(4), 2002.

[Wei00] K. Weihrauch. *Computable analysis*. Springer-Verlag, Berlin/Heidelberg, 2000.