On the Stability of Fast Polynomial Arithmetic

Sven Köhler
University of Paderborn
skoehler@upb.de

Martin Ziegler
University of Paderborn
ziegler@upb.de

Abstract

Operations on univariate dense polynomials—multiplication, division with remainder, multipoint evaluation—constitute central primitives entering as build-up blocks into many higher applications and algorithms. Fast Fourier Transform permits to accelerate them from naive quadratic to running time $\mathcal{O}(n \cdot \mathrm{polylog}\, n)$, that is softly linear in the degree n of the input. This is routinely employed in complexity theoretic considerations and, over integers and finite fields, in practical number theoretic calculations.

The present work explores the benefit of fast polynomial arithmetic over the field of real numbers where the precision of approximation becomes crucial. To this end, we study the computability of the above operations in the sense of Recursive Analysis as an effective refinement of continuity. This theoretical worst-case stability analysis is then complemented by an empirical evaluation: We use GMP and the iRRAM to find the precision required for the intermediate calculations in order to achieve a desired output accuracy.

1 Introduction

Consider two univariate polynomials $p,q \in F[X]$ over some fixed field F, represented as a dense list of coefficients: $p = \sum_{k=0}^{n} a_k \cdot X^k$, $q = \sum_{k=0}^{m} b_k \cdot X^k$. The problem of calculating a similar representation of their product $p \cdot q$ is naively solved by evaluating the Cauchy Product:

$$p \cdot q = \sum_{k=0}^{n+m} c_k \cdot X^k, \qquad c_k = \sum_{i=0}^{k} a_i \cdot b_{k-i} \ . \tag{1.1}$$

This takes $\mathcal{O}(nm)$ operations (additions and multiplications in F) which means quadratic running time for $n \approx m$. Similarly concerning polynomial division with remainder (given $p,q \in F[X]$, calculate $p \operatorname{div} q$ and $p \operatorname{rem} q$), the high school method may take up to $\mathcal{O}\left(n^2\right)$ arithmetic operations over F. And in order to evaluate a given $p \in F[X]$ of degree n at given $x_1,\ldots,x_n \in F$ simultaneously, n-fold repetition of Horner's Method takes $\mathcal{O}\left(n^2\right)$ steps.

Several theoretical methods are well-known which improve over these naive, quadratic time algorithms [GG03, BCS97]. We briefly review them and their applications in Sections 1.1 and 1.2. The purpose of the present work is to explore the practical benefit of these algorithms over the real. To this end, Section 1.3 reminds the reader of some basic notions in real number computation and their relation to numerical stability. Section 2 then turns fast multipoint evaluation from an algorithm into two computational problems, namely polynomial multiplication and remainder calculation, and investigates their computability in the sense of qualitative stability. For the cases which turn out as computable, Section 3 then gives a quantitative estimation on their stability: both theoretically and, using the iRRAM, empirically. They reveal that the need for intermediate high precision completely removes the theoretical benefits of fast multipoint evaluation.

1.1 Fast Polynomial Arithmetic

The key to the acceleration of many computational problems over $F[X]$ are efficient algorithms for the multiplication of two given polynomials of equal degree, say n. In the sequel let $M(n) = M_F(n)$

denote the number of arithmetic steps in F it takes to do so. The naive method has $M(n) \leq \mathcal{O}(n^2)$ which can be improved to $M(n) \leq \mathcal{O}(n^{1.585})$ and further to $M_F(n) \leq \mathcal{O}(n \cdot \log n)$ for $F = \mathbb{C}$ or $F = \mathbb{R}$; also $M_F(n) \leq \mathcal{O}(n \cdot \log n \cdot \operatorname{loglog} n)$ for $F = \mathbb{Z}_2$.

1.1.1 Karatsuba Multiplication

Suppose $n = m$ and split each polynomial into two parts: $p = p_1 \cdot X^k + p_0$, $q = q_1 \cdot X^k + q_0$. Naively,

$$p \cdot q = (p_1 \cdot q_1) \cdot X^{2k} + (p_1 \cdot q_0 + p_0 \cdot q_1) \cdot X^k + (p_0 \cdot q_0)$$

can be calculated recursively using 4 multiplications of polynomials of degree $k \approx n/2$ (because "$\times X^k$" means a mere shift of the coefficient vector); or with only 3 multiplications according to

$$p \cdot q = p_0 \cdot q_0 + p_1 \cdot q_1 \cdot X^{2k} + \big((p_0 + p_1) \cdot (q_0 + q_1) - p_0 \cdot q_0 - p_1 \cdot q_1\big) \cdot X^k \quad .$$

The latter leads to a recursive algorithm for multiplying two polynomials of degree n (w.l.o.g. a power of 2) using $M(n) \leq 3 \cdot M(n/2) + \mathcal{O}(n) \leq \mathcal{O}(n^{\log_2 3})$ arithmetic operations over F.

1.1.2 Fast Fourier Transform

A further approach applies to the coefficient vectors $\bar{a}$ and $\bar{b}$ the N-dimensional discrete fourier transform DFT_N for some $N \geq n + m + 1$ to the coefficient vectors $\bar{a}$ and $\bar{b}$, then multiplies their results $\tilde{a} := \mathrm{DFT}_N(\bar{a})$ and $\tilde{b} := \mathrm{DFT}_N(\bar{b})$ *componentwise* (i.e. in linear time), and finally transforms this product $\tilde{c}$ back: Since DFT_N is an algebra isomorphism, this indeed yields the desired $\bar{c} = \mathrm{DFT}_N^{-1}(\tilde{c})$. The benefit of this method over the naive one of course hinges—apart from the requirement that F admits a discrete fourier transform—on DFT_N (and DFT_N^{-1}) taking $\mathrm{o}(N^2)$ steps to evaluate. This turns out to be the case for $F = \mathbb{C}$ the field of complex numbers with $M(n) \leq \mathcal{O}(N \cdot \log N)$ as well as for surprisingly many further rings. For instance, the famous Schönhage-Strassen algorithm adjoins to $F = \mathbb{Z}_2$ certain 'symbolic' roots of unity in order to perform multiplication of n-bit integers in time $M(n) \leq \mathcal{O}(n \cdot \log n \cdot \operatorname{loglog} n)$, see e.g. [GG03, BCS97].

1.1.3 Fast Multiplication of Several Polynomials

Given $p_1, \ldots, p_k \in F[X]$, in order to calculate their product $p_1 \cdots p_k$, it is usually a bad idea to do this iteratively as $(p_1 \cdots p_{k-1}) \cdot p_k$ due to the imbalance of the degree of the arguments to most binary multiplication operations issued. Instead suppose that the inputs all have equal degree d; then combining them in form of a balanced binary tree leads to running time $M(dk \log k)$. In particular, the k-th power of a given $p \in F[X]$ of degree $\leq d$ can be calculated in $M(dk \log k)$ arithmetic operations.

1.1.4 Fast Division with Remainder

As already mentioned, the school method for polynomial division may take a number of steps quadratic in the degree of the inputs. A faster approach applies (a symbolic variant of) Newton's method in order to compute, given $f \in F[X]$ with $f(0) \neq 0$, the truncated formal power series $f^{-1} \operatorname{rem} X^{n+1}$ in as few as $\mathcal{O}(M(n))$ steps; cf. e.g. [BCS97, THEOREM 2.22]. By applying this algorithm to the polynomials obtained from reversing the (order of the) coefficients of given $a, b \in F[X]$, one obtains the coefficients q_i of $q := a \operatorname{div} b$ from

$$\Big(\sum_{i=0}^{n} a_i \cdot X^{n-i}\Big) \cdot \Big(\sum_{j=0}^{m} b_i \cdot X^{m-j}\Big)^{-1} \equiv \sum_{i=0}^{n-m} q_i \cdot X^{n-m-i} \bmod X^{n-m+1} \tag{1.2}$$

using one further multiplication; see the proof of [BCS97, COROLLARY 2.26] or [BLS03, SECTION 5.2]. This method is due to SIEVEKING and KUNG.

1.1.5 Fast Multipoint Evaluation

In order to evaluate a given $p \in F[X]$ at a given $x \in F$, Horner's rule with its linear running time is basically optimal [BCS97, THEOREM 6.5]. However when calculating the entire n-tuple $p(x_1), \ldots, p(x_n)$ for given $x_1, \ldots, x_n \in F$ and given p of degree $< n$, one can do better than an n-fold repeated invocation of Horner. The crucial observation is that,

$$\text{since } p \;=\; (p \operatorname{rem} q) + (p \operatorname{div} q) \cdot q, \quad \text{it holds:} \quad q(x) = 0 \quad \Rightarrow \quad p(x) \;=\; \big(p \operatorname{rem} q\big)(x) \;. \tag{1.3}$$

Hence, for $q := \prod_{i=1}^{n/2}(X - x_i)$, one division with remainder allows to effectively split the original problem into two of half the size: evaluating $p \operatorname{rem} q$ of degree $< n/2$ at $x_1, \ldots, x_{n/2}$; and doing so similarly for $p \operatorname{rem} \prod_{i=n/2+1}^{n}(X - x_i)$ on the other $n/2$ points $x_{n/2+1}, \ldots, x_n$. This divide&conquer approach leads to a recurrence for the running time $T(n) \leq 2 \cdot T(n/2) + \mathcal{O}(M(n)) \leq \mathcal{O}(M(n) \cdot \log n)$.

1.1.6 Alternatives

The above algorithms, and the rest of the present work, deal with polynomial coefficients w.r.t. the standard base $1, X, X^2, \ldots$. Although Bernstein Bases tend to behave better in terms of stability, fast arithmetic thereon, and in particular multipoint evaluation, is known today only rudimentarily [Far00, BG04]; similarly for the straight line program representation of (particularly sparse) polynomials.

1.2 Applications

1.2.1 Complexity Theory

One aspect of the algorithms from Section 1.1, they establish upper bounds on the asymptotic complexity of the problems they solve; and thus tighten the gap to the lower bounds particularly in the algebraic setting [BCS97].

The algorithms are also regularly referred to, and employed as black boxes, when upper bounding the complexity of other problems: recall e.g. how fast multipoint evaluation relied on fast division with remainder and in turn on fast multiplication. The same applies to greatest common divisor of polynomials [GG03, SECTION 11]. Further examples in this spirit consider structured matrices [Pan01, Bür00] and linearly recurrent sequences [Fid85, BGS04].

1.2.2 Practical Benefits

The algorithms realizing fast arithmetic are of course more complex than their naive counterparts. The large overheads of the first might therefore suggest a benefit of purely asymptotic nature. But in fact the input sizes n where it succeeds over the naive approach are generally modest in the range of 100 to 1000; see e.g. [Moe76] or [GG03, SECTION 9.7] or [LMS07]. These *breakeven-points* (as opposed to the absolute running times) are largely independent of technological progress—subject to a fair comparison: the competing implementations employ an equal degree of optimization like, e.g., both making use of modern processor extensions like cache or SSE [FLPR99, FJ05]. Further improvements store and use these breakeven-points for a hybrid approach: for a given input size n, automatically switch to that algorithm which suits best for it. This is of particular advantage for recursive approaches like FFT or for polynomial multiplication with its large choice from hard-coded over naive and Karatsuba to FFT.

Based on such techniques, fast polynomial and integer arithmetic has over the last 10 years made its entry into practice, conveniently available (often freely) as highly optimized libraries. Fast arithmetic on long integers has proven particularly beneficial in computational number theory (NTL, LiDIA); e.g. searching for large or twin primes. In fact all practical applications mentioned so-far apply to a discrete setting of bits, prime fields, or finite fields (i.e. polynomials over prime fields)!

1.2.3 Applications over the Reals

Over the reals (rational or floating point numbers, say) numerical issues and bit costs have to be taken into account which may spoil the advantages predicted by unit cost model [Pan01, SECTION 1.8].

The DFT itself is of course a unitary transformation and FFT rather well-behaved [PST02]. This is routinely exploited in digital signal processing for real-time spectral analyzers [Wel97]. It has also helped accelerating multipoint evaluating the Riemann-Siegel Z-function [OS88] which, although closely connected to the distribution of primes in number theory, is inherently *non*-discrete but in fact analytic. Fast real polynomial arithmetic however uses, in addition to FFT, multiplication of several polynomials and division with remainder whose stability is not clear at all.

As a matter of fact we had queried several experts in Symbolic and Algebraic Computation on their estimation on this issue and received contradictory replies; which led us to explore it ourselves. The (otherwise meritable) survey [Bre99] for instance avoids $\mathcal{O}(n \cdot \operatorname{polylog} n)$ algorithms. For Chebyshev points of multi-evaluation, a purportedly stable implementation is described in [Ger88]; but in view of the many-body application we are interested in the roughly equally-spaced case.

A similar question, namely concerning the stability of fast *matrix* multiplication, has recently (and surprisingly) been answered positively [DDHK07].

1.3 Effective Computation on Real Numbers

When should a (possibly partial) function $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ be considered effectively computable? Any sensible answer must take into account that actual digital computers can handle only rational numbers as an approximation to reals. The question thus boils down to approximate calculatation of $x \mapsto f(x)$.

1.3.1 Numerical Analysis

In Numerical Analysis, effective computation on real numbers is closely tied to notion(s) of *stability*:

- *forward*: maximal error of the output $f(x) - f(x^*)$ for any small perturbation x^* of the input x;
- *backward*: minimal perturbation x^* of the input x such that the output y' coincides with $f(x^*)$;
- *mixed* combines the two.

In addition, errors $y - y^* \in \mathbb{R}^m$ and perturbations $x - x^* \in \mathbb{R}^n$ may be quantified

- either with respect to their Euclidean $\| \cdot \|_2$ or their maximum norm $\| \cdot \|_\infty$;
- either absolutely or relatively as in $\|x - x^*\| / \|x\|$.

This notion of stability leads to the concept of condition numbers as introduced by ALAN M. TURING [Tur48]. These may apply either to the computational problem f under consideration; or to an algorithm A purportedly solving it. In the first case in-/stability is inherent to f and a bound to what A in the second case can attain. For instance, [XW07] determine various condition numbers of structured matrices, among them Vandermonde ones. Observe that multiplication of $V = (x_i^{j-1})_{i,j}$ to column vector $(p_0, \ldots, p_{n-1})$ amounts to the evaluation of $p = \sum_{i=0}^{n-1} p_i \cdot X^i$ simultaneously at $x_1, \ldots, x_n$.

1.3.2 Computable Analysis

Initiated by Turing [Tur36, Tur37] and others [Grz57, Lac58], Computable Analysis offers the following

Definition 1. *A function $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ is* computable *if some Turing machine can, given two sequences (q_i) and (ε_i) of (binary encodings of numerator and denominator of) rational vectors with $\|x - q_i\| \leq \varepsilon_i \to 0$, output similar sequences (p_j) and (δ_j) with $\|f(x) - p_j\| \leq \delta_j \to 0$; w.l.o.g. $\varepsilon_i = 2^{-i} = \delta_i$.*

In other words: given arbitrary close absolute approximations to x, output similar ones to $f(x)$; a reasonable, well-adopted notion indeed [Abe80, PER89, Wei00]. Conversely, a function *lacking* computability in the sense of Definition 1 can*not* be evaluated approximately, neither theoretically nor practically. For instance the so-called Main Theorem of Computable Analysis asserts that any *dis*continuous function f is *un*computable in this sense. The intuition underlying its proof is convincing indeed: In order to approximate f at a 'jump' x up to error $\delta > 0$ smaller than this jump, one needs to know x *exactly* (rather than up to any finite ε). Put differently: at a jump, arbitrarily small perturbations ε of the argument x' may affect the value $f(x')$ by some δ *un*bounded in $\varepsilon \to 0$. This is instability!

Regarding that continuity is thus necessary for a function f to be computable (i.e. stably evaluable), it is natural to consider f's *modulus* of continuity as a quantitative measure for the degree of stability. Indeed, such a modulus describes which input precision ε is required in order to obtain a given output precision δ; cf [Wei00, EXERCISE 7.1.7] and is a lower bound on the running time of any machine computing f [Wei00, LEMMA 7.1.3]—quite similar to the condition number of f lower bounding that of any algorithm solving f; recall the last paragraph of Section 1.3.1.

1.3.3 Comparison

Each, numerical and computable analysis has its own, independent notion of effective computation on real numbers based on an intuitive concept of "stability".

Example 2. *Recall that the Halting problem $H \subseteq \mathbb{N}$ consists of (integer encodings of) all Turing machines which terminate (on the empty input) and provably cannot be decided by any Turing machine. Now let $0 < u \ll 1$ denote the machine precision and consider a 'characteristic' function f_A of the set*

$$A := \{(2n+1)/2^{-k} : n \in H, k \in \mathbb{N}\}, \qquad f_A : x \mapsto 1 \text{ for } x \in A, \quad f_A : x \mapsto 1 - u/2 \text{ for } x \notin A \ .$$

In numerical analysis, since f's values hardly vary, it would be considered forward stable, both relatively and absolutely[1]*; and also backward stable because A is dense.*

On the other hand a Turing machine cannot approximate f (even when restricting to rational arguments) to error smaller *than $u/2$; hence in computable analysis f is considered unstable.*

Example 3. *Iterations of the Logistic Map $f : [0,1] \ni x \mapsto 4x \cdot (1-x) \in [0,1]$ are generally considered chaotic and numerically instable; whereas in computable analysis they are uniformly computable* [Wei00, THEOREM 6.2.1] *(and routinely dealt with by the iRRAM, see below).*

It seems that the main difference between numerical and computable analysis amounts to the distinction between fixed versus arbitrary/adaptive precision. We find that both[2] notions of stability equally have their justification and relevance, depending on the specific application and purpose.

1.3.4 GMP and iRRAM

Turing machines are awkward to write practical programs on [SGV94]. Actual numerical software is usually (maybe just for reasons of convention?) written in imperative high level languages like `FORTRAN`, `C++`, or `Java` with semantics closely resembling the *real Random Access Machine* (real-RAM), also known as algebraic [Tuc80] or BSS model [BCSS98]: Real numbers are treated as entities which can be input, stored, added, subtracted, multiplied, divided (say), compared, and output in *unit*

[1] Generally speaking, relative approximability of some f is equivalent to absolute approximability of f *plus* decidability of the zero problem for f; see [Yap04, THEOREM 5].

[2] In this context it is only fair to mention also the core library [KLPY99] supporting a nice blend of both aspects. However, there, our implementation of benchmark Algorithms B-D) from Section 3.1 ran into technical difficulties: it seems that when calculating a *tuple* $f_1, \ldots, f_m$ of functions, core evaluates each f_i separately, thus jeopardizing the benefit of fast algorithms sharing large common parts of the respective computations.

time each. This abstraction reflects the way *fixed*-precision floating point numbers are handled in actual computing hardware; it does however become unrealistic when taking into account issues of inaccuracy in numerical computation: *arbitrary* precision arithmetic infers running time depending on the *value* of the numbers involved, i.e. it leads to a different notion of complexity and even of computability [Wei00, EXAMPLE 9.7.2].

The work [BH98] has reconciled the BSS model with Turing computation in the sense of Definition 1: by modifying the semantics of *in*equality tests from decidable to semi-decidable and banning tests for equality (as taught in any introductory course on numerical programming anyway). It has then been put into practice [Mül00] as the library iRRAM with C++ interface overloading the usual mathematical operators. Technically, `main()` is executed first at some initial precision ε using interval arithmetic; if the resulting approximation δ turns out as insufficient (or, in the case of some intermediate test for inequality, if the argument y has not been obtained accurately enough to conclude, say, $y > 0$), the entire computation is restarted automatically at some higher precision ε. This auto-adaptive approach avoids some (although not all) sources of over-estimation often rendering interval calculations impractical. Moreover, iRRAM employs and benefits from the fast FFT-based and highly optimized arbitrary precision arithmetic provided one of the open libraries MPFR or GMP. The *GNU Multiple Precision Arithmetic Library* supports various operations on integers of unbounded size (and, based on that, on arbitrary precision floating point numbers). For maximum performance, it automatically and adaptively hybridizes among naive, Karatsuba, and FFT. (Arithmetic on polynomials is not supported, though.)

As a consequence, and in combination with some further details and optimizations mentioned in [Mül00], the iRRAM constitutes a convenient and surprisingly efficient [NW05] environment for rapid numerical prototyping—just hack in your favorite algorithm and let the system take care of the numerics—and for empirical stability analysis in the sense of Section 1.3.2: feed in some typical input instances and record which intermediate precision ε the iRRAM chooses in order to obtain the prescribed output precision δ.

2 Computability

For fixed n, polynomial multipoint evaluation $\mathbb{R}^{2n} \ni (p_0,\ldots,p_{n-1},x_1,\ldots,x_n) \mapsto \left(\sum_{i=0}^{n-1} p_i \cdot x_j^i\right)_{j\le n} \in \mathbb{R}^n$ is computable (see also Lemma 4a) below): apply Horner's method n times and observe that this algebraic algorithm translates into a computable one—however with running times typically quadratic in n (and of course depending on x_j, $j = 1,\ldots,n$); see Section 3.1.

Then we have fast multipoint evaluation; which, however, pertains to the algebraic (and unit cost model) of real computation; and which moreover is an algorithm rather than a computational problem. On the other hand recall from Section 1.1.5 that fast multipoint evaluation consists of multiplication of (several) polynomials and polynomial remainder calculation. And these *are* computational problems. Their computability (qualitative stability) is the topic of the present section.

2.1 Computability of Polynomial Evaluation and Multiplication

In Computable Analysis (Section 1.3.2), it is essential to exactly specify how objects like real numbers are encoded. This leads to Weihrauch's Type-2 Theory of Effectivity [Wei00, SECTION 4.1] as a convenient tool for expressing and handling such so-called *representations*. In our context, the computability of several problems turns out to hinge crucially on whether polynomials $p \in \mathbb{R}[X]$ are given as sequences of coefficients

i) $(p_0,p_1,\ldots,p_n,\ldots) \in \mathbb{R}^\infty$, zero almost everywhere (i.e. $p_i = 0 \forall i > n$ for *some* $n \in \mathbb{N}$);

ii) of *given* length, i.e. as $(p_0,p_1,\ldots,p_n) \in \mathbb{R}^n$ with some explicit *upper* bound $n \in \mathbb{N}$ on $\deg(p)$;

iii) as $(p_0,p_1,\ldots,p_n) \in \mathbb{R}^n$ such that, in addition, $p_n \neq 0$; i.e. $\deg(p)$ is given exactly.

Regarding that already a *single* real number is given as an infinite sequence of rationals (Definition 1), Encoding i) is quite natural; plus: it extends from polynomials to formal power series as employed in Section 2.2 below. Technically speaking it amounts to specifying at stage #N of the computation a finite number of terms (with degree $< N$, say) up to finite precision (2^{-N}, say). Consequently, w.r.t. Encoding i), an upper bound on $\deg(p)$ is not a continuous function of the "series" $(p_0, p_1, \ldots, p_n, \ldots)$; nor does the exact value $\deg(p)$ in Encoding ii) depend continuously on the tuple $(p_0, p_1, \ldots, p_n)$.

Lemma 4. *Consider polynomial evaluation* $\mathbb{R}[X] \times \mathbb{R} \ni (p,x) \mapsto p(x) \in \mathbb{R}$.

a) With respect to Encoding ii), this is computable.

b) For polynomial coefficients p_i restricted to some bounded subset of $\mathbb{R}$, and for arguments $x \in (-1,+1)$, evaluation is computable with respect to Encoding i);

c) whereas without either restriction, it is not. *More precisely:*

c1) even for polynomial coefficients from $[0,\varepsilon)$ encoded as i), evaluation $p \mapsto p(1)$ at $x = 1$ is discontinuous for each fixed $\varepsilon > 0$;

c2) and so is evaluation $p \mapsto p(\delta)$ for polynomial coefficients from unbounded $\mathbb{R}$ and fixed $\delta > 0$.

d) Multiplication $\mathbb{R}[X] \times \mathbb{R}[X] \ni (p,q) \mapsto p \cdot q \in \mathbb{R}[X]$ *on the other hand is computable with respect to all three encodings.*

2.2 Computability of Polynomial Division

Polynomial division consists of two operations: $p \operatorname{div} q$ determines the quotient, $p \operatorname{rem} q$ the remainder. Of course both can be computed from one another by virtue of Lemma 4d) according to the left part of Equation (1.3). In view of Section 1.1.4, it is worthwhile to recall the following

Definition 5. *For F a field, let $(F[\![X]\!], +, \cdot)$ denote the ring of formal power series in X (i.e. sequences over F w.r.t. convolution "$\star$") and $F[\![X]\!]^* := \{f \in F[\![X]\!] : f(0) \neq 0\}$ the group of units.*

As already mentioned, Encoding i) extends from $\mathbb{R}[X]$ to $\mathbb{R}[\![X]\!]$. Moreover, the proof of Lemma 4d) shows that multiplication in $\mathbb{R}[\![X]\!]$ is computable w.r.t. Encoding i). Concerning division, it becomes essential to know the exact degree of the denominator and an upper bound on the degree of the numerator polynomial:

Theorem 6. *a) The mapping* $\mathbb{R}[\![X]\!]^* \ni f \mapsto f^{-1} \in \mathbb{R}[\![X]\!]^*$ *is computable with respect to Encoding i).*

b) The mapping $\operatorname{rem} : \mathbb{R}[X] \times \mathbb{R}[X] \ni (a,b) \mapsto a \operatorname{rem} b \in \mathbb{R}[X]$ *is computable, provided an upper bound on $\deg(a)$ and the exact degree of b are given (i.e. w.r.t. Encoding ii for a and Encoding iii for b).*

c) For every non-zero $f \in \mathbb{R}$ and $\varepsilon > 0$ and even restricted to at most linear polynomials g, the mapping $g \mapsto f \operatorname{rem} g \in \mathbb{R}$ is discontinuous w.r.t. Encoding ii);

d) and so is $\mathbb{R}[X] \ni f \mapsto f \operatorname{rem}(X-1) \in \mathbb{R}$ w.r.t. Encoding i), even restricted to polynomial coefficients in $[0,\varepsilon)$.

Corollary 7. *Under the same conditions that render polynomial evaluation computable (Lemma 4a), fast multipoint evaluation is computable, too.*

Proof. Given $x_1, \ldots, x_m \in \mathbb{R}$, one can compute (the coefficients of) $q := \prod_{i=1}^{m} (X - x_i)$ of $\deg(q) = m$. From that, and given $p \in \mathbb{R}[X]$ w.r.t. Encoding ii) as in Lemma 4a), $p \operatorname{rem} q$ is computable according to Theorem 6b). This establishes the recursion step of fast multipoint evaluation (Section 1.1.5). □

3 Quantitative Stability

Section 2 explored whether and when polynomial evaluation, multiplication, and division is stable; but not *how* stable. For a quantitive analysis we first some bounds on the propagation of absolute error:

Fact 8. *a) addition* $|(x+y)-(x'+y')| \leq |x-x'|+|y-y'|$

b) multiplication $|(x\cdot y)-(x'\cdot y')| \leq |x|\cdot|y-y'|+|x-x'|\cdot|y|+|x-x'|\cdot|y-y'|$.

c) scalar product $\left|\sum_i x_i\cdot y_i - \sum_i x'_i\cdot y'_i\right| \leq \sum_i |x_i|\cdot|y_i-y'_i|+|x_i-x'_i|\cdot|y_i|+|x_i-x'_i|\cdot|y_i-y'_i|$.

d) With respect to 1-norm, convolution $(\vec{x}\star\vec{y})_k = \sum_{i\leq k} x_i\cdot y_{k-i}$ *satisfies:* $\|\vec{x}\star\vec{y}\|_1 \leq \|\vec{x}\|_1\cdot\|\vec{y}\|_1$.

e) polynomial evaluation of $p=\sum_{i<n} p_i\cdot X^i$ *and* $q=\sum_{i<n} q_i\cdot X^i$*, where* $\vec{p}=(p_0,\ldots,p_{n-1})$*:*

$$|p(x)-q(x)| \leq \|\vec{p}-\vec{q}\|_1 \quad \textit{for } |x|\leq 1, \qquad |p(x)-q(x)| \leq \|\vec{p}-\vec{q}\|_\infty\cdot|x|^n/(|x|-1) \quad \textit{for } |x|>1$$

f) Power series (and in particular polynomial) multiplication has

$$\left\|\overrightarrow{p\cdot q} - \overrightarrow{p'\cdot q'}\right\|_1 \;\leq\; \|\vec{p}\|_1\cdot\|\vec{q}-\vec{q}'\|_1+\|\vec{p}-\vec{p}'\|_1\cdot\|\vec{q}\|_1+\|\vec{p}-\vec{p}'\|_1\cdot\|\vec{q}-\vec{q}'\|_1 \;\;.$$

Relative errors thus add, as in scalar multiplication. In particular when calculating the product of several polynomials as in Section 1.1.3, errors double in each step; but so do the degrees m, that is, relative accuracy remains bounded linearly in m. The absolute error on the other hand depends on the magnitude of the coefficients—which is usually exponential in the degree m of the polynomials $q:=\prod_{i=1}^m (X-x_i)$ arising typically in fast multipoint evaluation: $(X-a)^m=\sum_{k=0}^m \binom{m}{k}\cdot(-a)^k\cdot X^{m-k}$ for instance has, in case $|a|\leq 1$ and for $k\approx|a|\cdot m/2$, coefficient $\binom{m}{k}(-a)^k$ of order $(|a|\cdot m/k)^k \geq 2^{|a|\cdot m/2}$.

Let us now move on to polynomial division, the second constituent of fast multipoint evaluation. According to the proof of Theorem 6a+b), this amounts to (a polynomial multiplication and) the solution of a lower triangular Toeplitz system of linear equations with unit diagonal and w.r.t. fixed right hand side $(1,0,\ldots,0)^\dagger$. Since we could not find meaningful bounds on the condition numbers for such cases, here comes an

3.1 Empirical Evaluation

We have implemented a spectrum of four algorithms for real polynomial multipoint evaluation of p on $x_1,\ldots,x_{\deg(p)}$, ranging from naive $\mathcal{O}(n^2)$ to fast $\mathcal{O}(n\cdot\log^2 n)$ arithmetic steps:

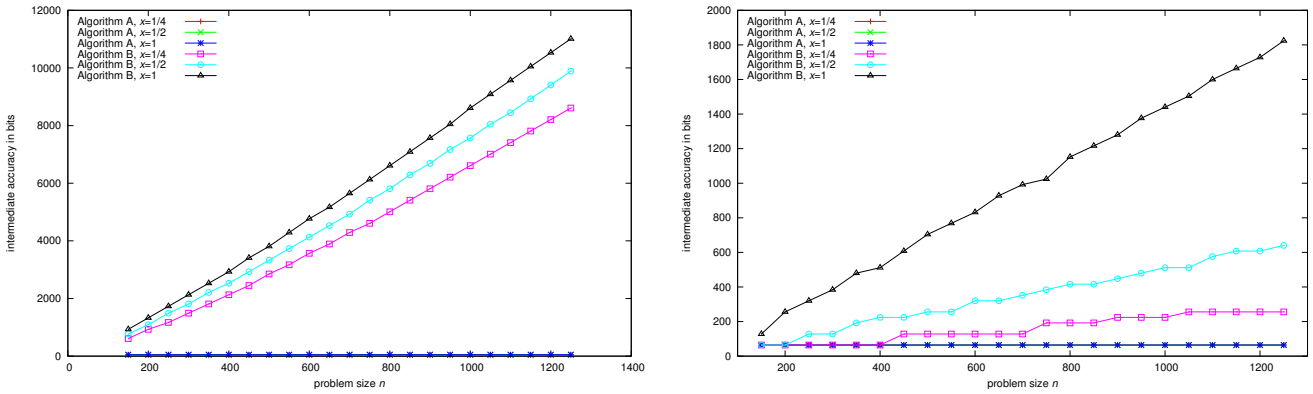
- Repeated application of Horner's method (A)
- The divide and conquer approach from Section 1.1.5
  - using the high school methods for polynomial arithmetic (B)
  - using Sieveking-Kung / Newton's Method for polynomial division
    - and based on Karatsuba multiplication of polynomials (C)
    - and based on FFT multiplication of polynomials. (D)

We considered random p and $(x_1,\ldots,x_n)$—coefficients and arguments drawn uniformly from $[0,1]$—as well as artificial benchmarks $p=\sum_{i=0}^{n-1} X^i$ and $(x,x,\ldots,x)$ where $n=128,\ldots,1024$ and $x=1/4,1/2,1$.

Our measurements (see the Figure below) reveal that Horner's method (A) suffices with basically constant accuracy independent of n, leading as expected to a quadratic running for n-fold invocation in naive multipoint evalution. On the other hand, already the simplest variant of divide and conquer (B) incurs during intermediate calculations a precision growing (at least) linearly in n in order to attain constant output precision; hence the running time be quadratic in n.

3.1.1 Certified Absolute vs. Optimistic Relative Accuracy

The iRRAM, in combination with GMP, finds a precision for intermediate calculations automatically and adaptively sufficient in order to guarantee a prescribed *absolute* output accuracy (certified numerics). This is hard to attain for the large intermediate values arising in method (B): recall the paragraph following Fact 8. Also, interval arithmetic is well-known to usually lead to overly pessimistic precision bounds [KLRK97]: in practice, worst-case inputs for different parts of an algorithm are often mutually exclusive, hence inaccuracies tend to cancel out. That 'explains' for the success of the *un*certified (for distinction in the sequel coined optimistic) approach often taken in numerical computations; which furthermore focuses on *relative* error. In order to quantitatively compare the effect of one attitude versus the other, we have implemented the above algorithms also using GMP directly and *non*-adaptively, i.e. with a fixed mantissa length—and then adjusted the latter manually such that the approximate result obtained just meets a prescribed error relative to the exact one (which was obtained differently). In this way, the minimum accuracy necessary for the above algorithms during intermediate calculations to reach a desired output precision was determined: a quantity is independent of the iRRAM. In fact, comparing it with the intermediate precision chosen by the iRRAM reveals that the conservative error overestimation inherent to the latter remains, at least for fast polynomial arithmetic, bounded by a constant factor.



QUANTITATIVE STABILITY OF FAST POLYNOMIAL MULTIPOINT EVALUATION
LEFT: CERTIFIED, ABSOLUTE ERROR 2^{-30}, RIGHT: OPTIMISTIC, RELATIVE ERROR 0.2

4 Conclusion

We thus have shown that, in spite of its name, 'fast' polynomial arithmetic is as slow as the naive algorithms: for practically relevant instances over real numbers, due to the increased intermediate precision *required* for reasonable output accuracy. Surprisingly (at least to us), these instabilities are not so much due to polynomial division with remainder alone; it is the multiplication of several real polynomials which leads to ill-conditioned input to the polynomial division.

The big open challenge thus consists in devising some variant of fast polynomial arithmetic which is numerically sufficiently mild to yield a net benefit in running time over the naive $\mathcal{O}(n^2)$ approach. As a first step, we suggest considering a replacement for repeated squaring to calculate polynomial powers.

References

[Abe80] Oliver Aberth. *Computable Analysis*. McGraw-Hill, 1980.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[BCSS98] Leonore Blum, Felipe Cucker, Mike Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1998.

[BG04] Dario A. Bini and Luca Gemignani. Bernstein–bezoutian matrices. *Theoretical Computer Science*, 315:319–333, 2004.

[BGS04] Alin Bostan, Pierrick Gaudry, and Éric Schost. Linear recurrences with polynomial coefficients and computation of the cartier-manin operator on hyperelliptic curves. In *Fq7, International Conference on Finite Fields and Applications*, volume 2948 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2004.

[BH98] Vasco Brattka and Peter Hertling. Feasible real random access machines. *J. of Complexity*, 14(4):490–526, 1998.

[BLS03] Alin Bostan, Grégoire Lecerf, and Éric Schost. Tellegen's principle into practice. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 37–44. ACM, 2003.

[Bre99] Richard P. Brent. Stability of fast algorithms for structured linear systems. In Thomas Kailath, editor, *Fast reliable algorithms for matrices with structure*, pages 103–116. SIAM, 1999.

[Bür00] Peter Bürgisser. The computational complexity to evaluate representations of general linear groups. *SIAM Journal on Computing*, 30(3):1010–1022, 2000.

[DDHK07] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg. Fast matrix multiplication is stable. *Numerische Mathematik*, 106(2):199–224, 2007.

[Far00] Rida T. Farouki. Convergent inversion approximations for polynomials in bernstein form. *Computer Aided Geometric Design*, 17(2):179–196, 2000.

[Fid85] Charles M. Fiduccia. An efficient formula for linear recurrences. *SIAM J. on Computing*, 14(1):106–112, 1985.

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[FLPR99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Ann. Symp. on Foundations of Comp. Sci. (FOCS)*, pages 285–297. IEEE Comput. Soc., 1999.

[Ger88] Apostolos Gerasoulis. A fast algorithm for the multiplication of generalized hilbert matrices with vectors. *Mathematics of Computation*, 50(181):179–188, 1988.

[GG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[Grz57] Andrzej Grzegorczyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.

[KLPY99] Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee K. Yap. A core library for robust numeric and geometric computation. In *Proc. 5th Annual Symposium on Computational Geometry (SoCG)*, pages 351–359. ACM, 1999.

[KLRK97] Vladik Kreinovich, Anatoly Lakeyev, Jiří Rohn, and Patrick Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations*. Applied Optimization. Springer, 1997.

[Lac58] Daniel Lacombe. Les ensembles récursivement ouverts ou fermés et leurs applications à l'analyse récursive ii. *Comptes rendus de l'Académie des sciences Paris*, 246:28–31, 1958.

[LMS07] Xin Li, Marc Moreno Maza, and Éric Schost. Fast arithmetic for triangular sets: from theory to practice. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 269–276. ACM, 2007.

[Moe76] Robert T. Moenck. Practical fast polynomial multiplication. In *Proceedings of the Symposium on Symbolic and Algebraic Computation*, pages 136–148. ACM, 1976.

[Mül00] Norbert Müller. The iRRAM: Exact arithmetic in C++. In *4th International Workshop on Constructivity and Complexity in Analysis*, pages 222–252. Springer, 2000.

[NW05] Milad Niqui and Freek Wiedijk. Many digits. friendly competition during the Small TYPES workshop on constructive analysis, types and exact real numbers, 2005. http://www.cs.ru.nl/ milad/manydigits/.

[OS88] Andrew M. Odlyzko and Arnold Schönhage. Fast algorithms for multiple evaluations of the riemann zeta function. *Transactions of the American Mathematical Society*, 309(2):797–809, 1988.

[Pan01] Victor Y. Pan. *Structured Matrices and Polynomials*. Birkhäuser/Springer, 2001.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*. Springer, 1989.

[PST02] Daniel Potts, Gabriele Steidl, and Manfred Tasche. Numerical stability of fast trigonometric transforms – a worst case study. *Concrete Appl. Math.*, 1:1–36, 2002.

[SGV94] Arnold Schönhage, Andreas F. Grotefeld, and Ekkehart Vetter. *Fast Algorithms—A Multitape Turing Machine Implementation*. BI Wissenschafts-Verlag, 1994.

[Tuc80] John V. Tucker. Computability and the algebra of fields. *Journal of Symbolic Logic*, 45:103–120, 1980.

[Tur36] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Procedings of the London Mathematical Society*, 42(2):230–265, 1936.

[Tur37] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem, A correction. *Procedings of the London Mathematical Society*, 43(2):544–546, 1937.

[Tur48] Alan M. Turing. Rounding-off errors in matrix processes. *Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 1948. doi:10.1093/qjmam/1.1.287.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[Wel97] Sri Welaratna. Thirty years of fft analyzers. *S&V Observer, Sound and Vibration*, january 1997.

[XW07] Hua Xiang and Yimin Wei. Structured mixed and componentwise condition numbers of some structured matrices. *Journal of Computational and Applied Mathematics*, 202:217–229, 2007.

[Yap04] Chee K. Yap. On guaranteed accuracy computation. In Falai Chen and Dongming Wang, editors, *Geometric Computation*, chapter 12, pages 322–373. World Scientific Publishing Co., Singapore, 2004.