

Variations on Computing Reciprocals of Power Series

Arnold Schönhage

Institut für Informatik, Universität Bonn (Germany)

February 5, 2001

Summary by Ludovic Meunier

Abstract

Fast algorithms for polynomial division with remainder are key tools in computer algebra. The power series domain defines a suitable framework where such algorithms can be efficiently constructed. While revisiting Kung’s article [5], Arnold Schönhage discusses algebraic complexity bounds for the computation of reciprocals of power series and describes a new algorithm for this task involving Graeffe’s root squaring steps.

1. Introduction

By means of Newton’s iteration, reciprocals of power series modulo x^{n+1} can be computed with complexity $O(M(n))$, where $M(n)$ denotes the complexity of multiplication (see, e.g., [6] for a survey). However, the Bachmann–Landau O -notation hides a multiplicative constant, which needs to be investigated, for instance in order to determine cross-over points when a collection of algorithms is available.

Section 2 sets the required background by recalling a few definitions from algebraic complexity. Section 3 presents an algorithm for computing reciprocals of power series, while discussing complexity bounds. Section 4 describes a new algorithm and its implementation over \mathbb{Z} .

2. Algebraic Complexity

Let F be a field and let $A(x) = \sum_{i \geq 0} a_i x^i \in F[[x]]$ denote a formal power series of the indeterminate x . Here, *formal* means that convergence matters are out of concern. Let $D = F(a_0, a_1, \dots)$ define a domain where a_i ’s are regarded as indeterminates. If D is endowed with the four arithmetic operations $(+, -, *, /)$ and a scalar multiplication, then an algorithm that inputs the power series $A(x)$ consists of a finite sequence of operations in D . Counting these operations defines the algebraic complexity, which is an intuitive way of reflecting performances of the algorithm. Two models of complexity are worth considering. The *arithmetic* complexity, denoted by L ,¹ charges one unit of cost for each operation in D , while the *nonscalar* complexity, denoted by C , only counts nonscalar multiplications and divisions.

3. Kung’s Algorithm Revisited

The underlying algorithm used for the accurate cost calculation is based on Newton’s iteration for reciprocals, as discussed by Kung in [5].

¹For notational convenience, arithmetic complexity is also denoted by M (resp. Λ) for multiplication (resp. fast Fourier transform).

3.1. Kung's algorithm. Let $R(x)$ be the reciprocal of the unit $A(x)$ with respect to the field $D[[x]]$. Define the function f from the subdomain of $D[[x]]$ whose elements have nonzero constant term to $D[[x]]$ by $f(s) = s^{-1} - A(x)$. Thus $R(x)$ is just the zero of f .

Newton's iteration is a second-order iteration² and consists of a linear approximation of f . Newton's iteration function \mathcal{N} is given by:

$$(1) \quad \mathcal{N}(s) = s - \frac{f(s)}{f'(s)} = s(2 - A(x)s),$$

where f' denotes the derivative of f , which is defined algebraically (see [8]). Let n be a power of two and

$$(2) \quad A_{2n}(x) = A(x) \bmod x^{2n+1},$$

$$(3) \quad R_n(x) = 1/A(x) \bmod x^{n+1}.$$

Newton's iteration features a quadratic convergence (see [3, Chap. 4]): the number of accurate terms doubles at each iteration. This may be expressed by

$$(4) \quad R_{2n}(x) = \mathcal{N}(R_n(x)) \bmod x^{2n+1}.$$

From (2) and (3), there exists a polynomial P of degree at most $n-1$ such that

$$(5) \quad R_n(x)A_{2n}(x) = 1 + x^{n+1}P(x) \bmod x^{2n+1}.$$

Combining (1), (5) and the expansion (4) leads to a recursive formula that computes the reciprocal of $A(x)$ modulo x^{2n+1} :

$$(6) \quad \frac{1}{A(x)} = R_{2n}(x) = R_n(x)(1 - x^{n+1}P(x)) \bmod x^{2n+1}.$$

Equations (5) and (6) both charge $M(n) + O(n)$ units of cost. Therefore, the overall arithmetic complexity of Kung's algorithm is bounded by

$$(7) \quad L(2n) \leq L(n) + 2M(n) + O(n).$$

Unfolding this recurrence leads to $L(n) = O(M(n))$ for all known multiplications.

The derivation of the exact arithmetic complexity from (7) depends on a specific algorithm for multiplication of polynomials. The next section describes a multiplication algorithm involving fast Fourier transform (FFT). Originally, Kung derived (7) for nonscalar complexity, where $M(n) = 2n + 1$, and found $C(n) < 4n$. Actually, the lowest upper bound presently known for the nonscalar complexity is $C(n) < 3.75n$. Kalorkoti derived this latter result from Kung's third-order iteration [4] and taking advantage that squaring modulo x^{n+1} is less expensive than multiplying modulo x^{n+1} (see [2, Chap. 2]).

3.2. FFT and fast multiplication. The N -point FFT defines a ring isomorphism from the quotient $F[[x]]/(x^N)$ to F^N . It is an evaluation-interpolation map where the evaluation points, also called Fourier points, are the N th roots of unity. Actually, the FFT is the evaluation-interpolation map whose implementation yields the lowest known complexity. Indeed, the symmetry properties of the N th roots of unity allow a divide-and-conquer implementation [3, Chap. 4]. The arithmetic complexity of N -point FFT is bounded by $\Lambda(N) \leq 3/2N \log N - N + 1$ (see [2, Chap. 2]).

The FFT performs fast back and forth conversions from an evaluated form to its interpolated form. Thus, low complexity algorithms can be achieved by taking advantage of each representation. In particular, fast multiplication consists in converting both operands into their evaluation forms with two FFTs, performing a coefficient-wise multiplication, and delivering the result with one

²Third-order iteration is mentioned later and consists of a parabolic approximation.

backward FFT. Schönhage shows that multiplication of polynomials of degree n (some restrictions on n are needed and discussed later) according to this method has algebraic complexity

$$(8) \quad M(n) = (9 + o(1)) n \log n.$$

3.3. Kung’s algorithm revisited. Direct substitution of (8) into (7) leads to

$$L(n) \leq (18 + o(1)) n \log n.$$

However, Schönhage obtains a lower multiplicative constant by deferring the last backward FFT. R_n and A_{2n} are first converted into their evaluation forms, requiring two direct N -point FFTs, which cost $2\Lambda(N)$. Then, steps (5) and (6) compute the evaluation form of $R_n P$, involving two coefficient-wise multiplications and two subtractions, which add $4N$ units of cost. One ultimate backward N -point FFT interpolates $R_n P$ with $\Lambda(N)$ operations. Therefore, (7) becomes

$$L(2n) \leq L(n) + 3\Lambda(N) + 4N.$$

A typical value for N is the lowest power of two that is greater than $d = \deg(R_n(x)A_{2n}(x)) = 3n$. However, a significant overhead is expected when d is slightly greater than the nearest power of two. In this case, the arithmetic complexity for the N -point FFT is $\Lambda(N) < 3d \log(2d)$. Thus, Schönhage suggests for N a scaled power of two of the form $N = c 2^\nu$, where $\nu = \lceil \log(d) \rceil - \lfloor \log \log(d + 1) \rfloor$ and $c = \lceil d/2^\nu \rceil$. This latter choice for N yields a lower bound

$$\Lambda(N) \leq d(3/2 \log(d) + 13/5 \log \log(d + 1) + O(1)).$$

This precise count yields the arithmetic complexity for reciprocals

$$L(n) \leq (27/2 + o(1)) n \log n.$$

Surprisingly, Newton’s third-order iteration does not yield a better bound for arithmetic complexity, as opposed to the case of nonscalar complexity (see Section 3.1).

4. A New Algorithm over \mathbb{Z}

Algorithms for division of polynomials reduce the division task to multiplications. However, while featuring an attractive asymptotic complexity, such reductions may involve detours and tricks whose implementations lead to tremendous multiplicative constants. Indeed, earlier algorithms for division of polynomials shared this drawback. Therefore, Schönhage suggests a new fast algorithm by means of Graeffe’s root squaring with a low constant and ready for an immediate implementation due to its extreme simplicity.

4.1. Graeffe’s root squaring method. Graeffe’s squaring method originates in numerical analysis for solving polynomial equations [1]. This method proceeds from any polynomial $A(x)$ in $F[x]$ to the even polynomial $G(x^2) = A(x)A(-x)$.

In $F[[x]]$ the reciprocal of $A(x)$ modulo x^{n+1} may be written as

$$(9) \quad \frac{1}{A(x)} = \frac{A(-x)}{A(-x)A(x)} \bmod x^{n+1}.$$

In equation (9), the denominator of the right hand-side contains at most $n + 1$ terms, but only half of them are significant when computing modulo x^{n+1} . Therefore, Graeffe’s rule reduces the task of inverting $n + 1$ terms to a half-sized problem. Thus, the corresponding algorithm works recursively as follows (notations are those of (2) and (3)). With $k = \lfloor n/2 \rfloor$, Graeffe’s step computes

$$G_k(x^2) = A_n(x)A_n(-x) \bmod x^{n+1},$$

charging at most $n + 1$ nonscalar units of cost. Indeed, typically, nonscalar complexity for such a multiplication is $C(n) = 2n + 1$ (see [2, Chap. 2]). However, the polynomial A_n may be rewritten as

$$A_n(x) = A_n^{(\text{even})}(x^2) + xA_n^{(\text{odd})}(x^2),$$

which shows that both $A_n(x_0)$ and $A_n(-x_0)$, for any x_0 lying in the ground field, can be computed together as follows

$$A_n(\pm x_0) = A_n^{(\text{even})}(x_0^2) \pm x_0 A_n^{(\text{odd})}(x_0^2).$$

Therefore, Graeffe's step requires at most $n + 1$ essential multiplications, by evaluation of A_n for $n + 1$ distinct squares. The reciprocal of $G_k(x)$ modulo x^{k+1} , denoted by $H_k(x)$, is determined by recursive calls. An ultimate multiplication

$$R_n(x) = A_n(-x)H_k(x^2) \bmod x^{n+1}$$

delivers the result, charging extra $n + 2k + 1$ units of nonscalar cost. Then, the nonscalar complexity is bounded by $C(n) \leq 6n + 2 \log(n/2)$, which is slightly weaker than Kalorkoti's (see Section 3.1) but the implementation of Graeffe's approach is straightforward.

4.2. Application to reciprocals over \mathbb{Z} . This section deals with units of the ring $\mathbb{Z}[[x]]$ of the form $A(x) = 1 + \sum_{i>0} a_i x^i$. This form naturally arises with divisions by monic polynomials computed via the substitution $x \mapsto 1/x$.

Basically, the implementation of Graeffe's method consists in mapping polynomials to integers expressed in some radix r_0 notation, so that multiplication of integers can be used. This idea is based on Kronecker's trick of encoding polynomials with bounded coefficients in a single integer. Let ϕ_{r_0} be a ring morphism from $\mathbb{Z}_n[x]$ (i.e., polynomials of $\mathbb{Z}[x]$ of degree less than n) to \mathbb{Z} that evaluates polynomials at $r_0 \in \mathbb{N}$. If there exists a constant β such that $|a_i| < \beta^i$ holds for each $i > 0$, then the bit size of the coefficients of R and G can be bounded. Thus, under this assumption, $r_0 \in \mathbb{N}$ can be chosen such that the evaluation map ϕ_{r_0} is a bijection and N can be optimally determined. The arithmetic complexity can easily be derived

$$L(n) = 6M(\tau n^2),$$

where $\tau = \log(3\beta)$ and where the Schönhage–Strassen algorithm for multiplication of integers, which features the lowest known complexity $M(m) = O(m \log(m) \log \log(m))$ [7], is likely to be used.

Bibliography

- [1] Bareiss (Erwin H.). – Resultant procedure and the mechanization of the Graeffe process. *Journal of the ACM*, vol. 7, 1960, pp. 346–386.
- [2] Bürgisser (Peter), Clausen (Michael), and Shokrollahi (M. Amin). – *Algebraic complexity theory*. – Springer-Verlag, Berlin, 1997, xxiv+618p. With the collaboration of Thomas Lickteig.
- [3] Geddes (K. O.), Czapor (S. R.), and Labahn (G.). – *Algorithms for computer algebra*. – Kluwer Academic Publishers, Boston, MA, 1992, xxii+585p.
- [4] Kalorkoti (K.). – Inverting polynomials and formal power series. *SIAM Journal on Computing*, vol. 22, n° 3, 1993, pp. 552–559.
- [5] Kung (H. T.). – On computing reciprocals of power series. *Numerische Mathematik*, vol. 22, 1974, pp. 341–348.
- [6] Salvy (Bruno). – *Asymptotique automatique*. – Research Report n° 3707, Institut National de Recherche en Informatique et en Automatique, 1999. 20 pages.
- [7] Schönhage (A.) and Strassen (V.). – Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, vol. 7, 1971, pp. 281–292.
- [8] van der Waerden (B. L.). – *Modern algebra*. – Frederick Ungar Publishing Co., New York, N. Y., 1949, vol. I, xii+264p.