Approximate Evaluation of a Polynomial on a Set of Real Points

Victor Pan

TR-92-076

November 1992

Approximate Evaluation of a Polynomial on a Set of Real Points

Victor Pan

Department of Mathematics and Computer Science
Lehman College, CUNY
Bronx, NY 10468
and
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704-1105

Abstract

The previous best algorithm for approximate evaluation of a polynomial on a real set was due to Rokhlin and required the order of $mu+nu^3$ infinite precision arithmetic operations to approximate [on a fixed bounded set $X(m)$ of $m+1$ real points] a degree n polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$ within the error bound $2^{-u} \sum_{i=0}^{n} |p_i|$. We develop an approximation algorithm, which decreases Rokhlin's record estimate to $O(m \log^2 u + n \min \{u, \log n\})$. For $\log u = o(\log n)$, this result may also be favorably compared with the record bound $O((m+n)\log^2 n)$ on the complexity of the exact multipoint polynomial evaluation. The new algorithm can be performed in the fields (or rings) generated by the input values, which enables us to decrease the precision of the computations [by using modular (residue) arithmetic] and to simplify our computations further in the case where $u = O(\log n)$. Our algorithm allows NC and simultaneously processor efficient parallel implementation. Because of the fundamental nature of the multipoint polynomial evaluation, our results have further applications to numerical and algebraic computational problems. By passing, we also show a substantial improvement in the Chinese remainder algorithm for integers based on incorporating Kaminski's fast residue computation.

Key words: polynomial evaluation, interpolation, approximation algorithms, rational algorithms, algebraic and symbolic computing, numerical stability, computational complexity, Chinese remainder algorithm.

1991 Math. Subject Classification: 68Q25, 65D05, 65D15, 65Y20.

1 Introduction.

The techniques for algebraic computing and for numerical computing have been historically developing quite independently of each other, although the power of combining their advantages has been increasingly appreciated in recent years and has been advocated, for instance, in [15] and [2]. The present paper gives some new evidence of the advantages of such a combination: we improve the known results on the multipoint polynomial evaluation (on a real set) by computing approximations, rather than the exact solutions (which is a typical feature of numerical computing). We, however, obtain such approximations by applying (some new and some old) purely algebraic techniques.

The literature on efficient evaluation of a polynomial dates back to a linear time algorithm, using n additions and n multiplications, for the exact evaluation of a polynomial at a single point. The algorithm is usually called Horner's rule, due to the work of 1819 by Horner [7], but was actually discussed by Isaac Newton [12], already in 1669. For the evaluation at a single point, this algorithm has been proved to be optimum ([13]).

We will consider the fundamental problem of evaluation of a polynomial of degree n, simultaneously at m points, in less than nm time (measured by the number of arithmetic operations involved, hereafter referred to as *ops*). In 1971 Borodin and Munro [4] published an $O(n^{1.91})$ time algorithm for multipoint polynomial evaluation, for $m = n$, thus giving the first algorithm with the complexity below nm. The asymptotically fastest algorithm known for the multipoint exact polynomial evaluation problem is due to Moenck and Borodin [10], who reduced the problem to repeated polynomial divisions. Including here the Sieveking-Kung fast algorithm for polynomial division made this an $O((m+n)\log^2 n)$ algorithm. Strassen [20] proved the lower bound $\lceil m \log n \rceil$.

Our goal is to present efficient algorithms for multipoint *approximate* polynomial evaluation. Indeed, in most applications, the computation needs to be done up to some limited accuracy of the machine, say, 32 or 64 bits, whereas the number of points and the degree of the polynomial may grow arbitrarily large. Thus, we desire an algorithm for the approximate evaluation of a polynomial

$$p(x) = \sum_{i=0}^{n} p_i x^i \tag{1.1}$$

on a fixed set $X(m)$ of $m+1$ points,

$$X(m) = \{x_0, x_1, \ldots, x_m\} ; \tag{1.2}$$

this algorithm should achieve a fixed accuracy of the output and should support upper estimates for computational complexity that grow linearly (with a small constant multiple) with $m+n$, or at least noticeably slower than $(m+n)\log^2 n$.

The first major progress in this direction was made by Rokhlin [18]. Rokhlin's algorithm uses $O(mu+nu^3)$ (infinite precision) ops in order to compute (within the error bound $2^{-u}p$, $p \le \sum_{i=0}^{n} |p_i|$) the set of the values

$$V(m) = \{p(x_j),\ j = 0, 1, \ldots, m\}, \tag{1.3}$$

for a positive u and for a set $X(m)$ of (1.2) lying on a real interval $\{x, 0 \leq x \leq 1\}$. The number of ops is linear in $m+n$ if u is a fixed constant. The transition to larger linear intervals can be made by means of linear transformations of the variable x and applying the same algorithm on two or more than two adjacent real intervals. The algorithm of [18] is numerically stable and performs well with a finite precision. Although its computational complexity is linear in n and m, the multiplicative factor of u^3 is large even for a relatively small u, and this limits both practical application and theoretical value of the algorithm.

In the present paper we substantially modify the approach of [18] (see our Remark 3.1), making it conceptually simpler and, unlike [18], exploiting some algebraic tools.

Let us next compare our resulting estimate for the arithmetic computational cost with those of [4,18,20]. Whereas the algorithm of [4] exactly computes the values $p(x_j)$, for $j = 0, 1, \ldots, m$, in $O((m+n)\log^2 n)$ ops, we decrease this cost estimate to $O((m+u)\log^2 u + n \min\{u, \log n\})$ (infinite precision) ops (see Remark 6.1) because, as in [18], instead of computing exactly, we approximate $p(x_j)$ within the error bound $2^{-u}p$, $p \leq \sum_{i=0}^{n} |p_i|$. The complexity bound of [4] (for the exact evaluation) exceeds this our bound (for approximation), by more than a constant factor, whenever $\log u = o(\log n)$. Our upper bound (on the complexity of the approximation) turns out to be strictly less than the lower bound of [20] (for the exact multipoint evaluation) already for u of the order of $\log n$. And finally, our bound is also superior to the cited bound of [18] for the approximate evaluation problem.

Technically, we rely on the approximation to a given polynomial $p(x)$ on a real interval $(-a, a)$, $0 < a \leq 1/2$ (say), by means of its interpolation on the Chebyshev set of d points on this interval by the polynomial $v(x)$, of degree less than d, and in section 2 we prove that this polynomial approximates $p(x)$ within $O(2^{-d}p/\sqrt{d})$. Such an accuracy may satisfy the user, already for smaller values of d (say, for $d = 20$), and then we shall compute $v(x)$ [rather than $p(x)$] on the set $X(m)$ of (1.2).

First we need, however, to obtain the coefficients of $v(x)$. Theoretically, we may do this by computing $p(x)$ on the d point Chebyshev set, with the subsequent interpolation by the degree $d-1$ polynomial, and this approach can be applied even if we just have a black box subroutine for the evaluation of $p(x)$ [rather than the coefficients of $p(x)$], which can be convenient, for instance, if we seek the eigenvalues of a matrix, with the characteristic polynomial $p(x)$. However, this way to approximating $p(x)$ on the set $X(m)$ of (1.2) is inconvenient for symbolic computation (because Chebyshev's points are irrational) and generally is not good numerically [because interpolation is an ill-conditioned computational problem (see Appendix B)]. Due to this ill-conditioning, the roundoff errors of numerical computations may generally contaminate the output values $p(x_j)$ of (1.3) too much even if we apply slower but numerically more stable algorithms of [6,11,21] for the evaluation and interpolation, rather than the faster algorithms of [1,2,10,14].

Thus, we will devise some alternate, algebraic algorithms, which compute the coefficients of $v(x)$ in any ring containing the values $a/2, p_0, \ldots, p_n$ as its elements; to do this, we show that $v(x) = p(x) \bmod L(2x/a)$, where $L(y)$ is a fixed monic polynomial with integer coefficients. If the values $a, p_0, \ldots, p_n$ are rational, we may perform almost all of the computations (with $\lceil \log_2 M \rceil$-bit precision) in the ring (or field) $\mathbf{Z}_M$ of integers modulo an integer M and at the end output the coefficients of $v(x)$ with no errors (see section 6).

Our algorithms allow their parallel implementation in polylogarithmic time with full processor efficiency (see Remark 5.2).

We may decrease the precision of our computations and, consequently, their bit-complexity, by applying the Chinese remainder algorithm. Adding here Kaminski's algorithms [8] for computing the residues of polynomials and integers, we may decrease the bit-complexity of our computations. We estimate such complexity in section 6, and by passing, we decrease the known upper estimate for the bit-complexity of the Chinese remainder algorithm. (We achieve such a decrease by incorporating Kaminski's algorithm of [8].)

We organize this paper as follows. After some preliminaries in the next section, we present our algorithms (which can be performed over abstract rings or fields) in sections 3 and 4. We estimate the arithmetic cost of our computations in section 5 and their precision and bit-complexity in section 6. Section 7 is left for discussion and three appendices for some related and auxiliary material.

2 Auxiliary Results

For a fixed positive a, we identify the set $X(a,d)$ of the d Chebyshev points (nodes) on the real interval $\{x\colon -a \le x \le a\}$ as follows:

$$\begin{aligned} X(a,d) \; &= \{at_{2h+1} = a\cos\left(\tfrac{2h+1}{2d}\pi\right), \\ &\qquad h = 0,1,\ldots,d-1\} \, . \end{aligned} \tag{2.1}$$

We will need the following basic result:

Theorem 2.1 ([5], p. 120). *For real a and b, $a \ge b$, a natural d, and a real function $f(x)$, with a bounded derivative $f^{(d)}(x)$ of the d-th order on the interval $\{x\colon b \le x \le a\}$, such that*

$$M = \max_{b\le x\le a} |f^{(d)}(x)| \, ,$$

let a polynomial $v(x) = \sum_{i=0}^{d-1} v_i x^i$ interpolate to $f(x)$ on the set of the Chebyshev points of (2.1), that is, let

$$v(at_{2h+1}) = f(at_{2h+1}) \, , \qquad h = 0,1,\ldots,d-1 \, . \tag{2.2}$$

Denote that

$$E = E(f,a,b,d) = \max_{b\le x\le a} |f(x) - s(x)|.$$

Then

$$E \le \frac{2M}{d!}\left(\frac{a-b}{4}\right)^d \, . \tag{2.3}$$

Since we may shift from $f(x)$ to $f(-x)$, we will hereafter assume that $a \ge |b|$, so that

$$0 \le a - b \le 2a \, .$$

We will apply Theorem 2.1 in the case where $a < 2/3$, and we will further assume that $da \geq 1 - a$ (see Remark 2.3 in the case where $da < 1 - a$). Designate that $H = \lfloor \frac{d}{1-a} \rfloor$, $r = 1/a$ and observe that

$$r = 1/a > 3/2\,,\; H = \lfloor \frac{d}{1-a} \rfloor = d + \lfloor \frac{da}{1-a} \rfloor \geq d + 1\,. \tag{2.4}$$

The latter inequality follows due to the assumption that $da \geq 1 - a$.

Furthermore, let $f(x)$ be the polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$ of (1.1) with real coefficients. In this case we will next deduce that

$$E = E(f, a, b, d) \leq E^* = (\frac{8r-4}{\pi d})^{1/2} \sum_{i=d}^{n} |p_i| (\frac{1-br}{4r-4})^d \, e^{1/(12H)}\,, \tag{2.5}$$

where $H \geq d + 1$, $e = 2.71828\ldots$, $e^{1/(12H)} \leq e^{1/12} < 1.1$, $\frac{1-br}{4r-4} \leq \frac{1}{2r-2}$, due to (2.3), (2.4).

Remark 2.1. Observe that $1 - br \leq 2$ for $|b| \leq a$ and $1 - br = 2$ for $b = -a$. Denote that $p = \sum_{i=d}^{n} |p_i|$ and deduce from (2.5) that

$$E^* = ((8r-4)/(\pi d))^{1/2} \, (p/(2r-2)^d) \; e^{1/(12H)}\,,$$

for $b = -a$. (Note that $2r - 2 > 1$ since $r > 3/2$.) In particular, recall that $e^{1/(12H)} < 1.1$ and obtain that

$$E^* < 1.1(12/(\pi d))^{1/2} 2^{-d} p\,,\; \text{for } r = 2,\, a = -b = 1/2\,,$$
$$E^* < 1.1(20/(\pi d))^{1/2} 4^{-d} p\,,\; \text{for } r = 3,\, a = -b = 1/3\,,$$
$$E^* < 1.1(28/(\pi d))^{1/2} 6^{-d} p\,,\; \text{for } r = 4,\, a = -b = 1/4\,.$$

Remark 2.2. Since the error bound E^* rapidly decreases as a and/or $a - b$ decrease, we may greatly decrease the approximation error if we partition the original interval $\{x\colon b \leq x \leq a\}$ into two or several smaller subintervals and approximate $p(x)$ separately on each of them. By shifting and scaling the variable x, we may turn each such a subinterval into the interval of the form $\{x\colon -a \leq x \leq a\}$, where $0 < a < 2/3$, say, $a = 1/2$ or $a = 1/4$.

To prove (2.5), apply Theorem 2.1, take into account (2.3), and deduce that

$$\begin{aligned} E &\leq 2(\tfrac{a-b}{4a})^d \,|\textstyle\sum_{i=d}^{n} p_i a^i \binom{i}{d}| \\ &\leq 2m(a,d)(\tfrac{a-b}{4a})^d \textstyle\sum_{i=d}^{n} |p_i|\,, \end{aligned} \tag{2.6}$$

where $\dfrac{a-b}{4a} = \dfrac{1-br}{4}$ [see (2.4)],

$$m(a,d) = \max_{d \leq h \leq n} \left\{ a^h \binom{h}{d} \right\}\,.$$

Since $a^{h+1}\binom{h+1}{d} < a^h \binom{h}{d}$ for $h \geq d$ if and only if $(h+1)a < h + 1 - d$, it follows that

$$m(a,d) = a^H \binom{H}{d} = (1/r^H)\, H!/(d!(H-d)!)$$

where $H - d \geq 1$, $r = (1/a) > 3/2$ [see (2.4)], and

$$H = \lfloor \frac{d}{1-a} \rfloor = \frac{dr}{r-1} - q\,,\quad 0 \leq q < 1\,. \tag{2.7}$$

Applying Stirling's formula, we obtain that

$$m(a,d) < \frac{(H/r)^H}{d^d(H-d)^{H-d}}\left(\frac{H}{2\pi\, d(H-d)}\right)^{1/2} e^{1/(12H)}\ ,\ e = 2.71828\ldots,\ e^{1/(12H)} < 1.1\ ,\ \text{for } H \geq d+1\ .$$

We deduce from (2.7) that

$$H-d = \frac{d}{r-1} - q\ ,\ \frac{H}{H-d} = \left(1+\frac{(1-a)q}{H-d}\right)r\ .$$

It follows that $\frac{H}{H-d} < (2-a)r$, since $H-d \geq 1 > q$. Therefore,

$$\left(\frac{H}{H-d}\right)^{H-d} = \left(1+\frac{(1-a)q}{H-d}\right)^{H-d} r^{H-d} \leq e^{(1-a)q}\, r^{H-d}\ .$$

On the other hand,

$$\left(\frac{H}{d}\right)^d = \left(\frac{1}{1-a} - \frac{q}{d}\right)^d = \frac{1}{(1-a)^d}\left(1-\frac{(1-a)q}{d}\right)^d \leq \frac{1}{(1-a)^d\, e^{(1-a)q}}\ .$$

Substitute the latter expressions for $(\frac{H}{H-d})^{H-d}$, $(\frac{H}{d})^d$, and the upper bound $(2-a)r$ on $H/(H-d)$ and transform our previous bound on $m(a,d)$ as follows:

$$m(a,d) < ((1-a)r)^{-d}\left((2-a)r/(2\pi d)\right)^{1/2} e^{1/(12H)}\ .$$

Then replace a by $1/r$ and obtain that

$$m(a,d) < (r-1)^{-d}\left((2r-1)/(2\pi d)\right)^{1/2} e^{1/(12H)}\ .$$

Combine this estimate with (2.6) and obtain (2.5).

Remark 2.3. If we relax our earlier assumption and consider the case where $da < 1-a$, then we obtain from the equations of (2.4) that $H = d$, and therefore, $m(a,d) = a^d$, $E \leq 2(\,(a-b)/4)^d \sum_{i=d}^{n} |p_i|$.

3 Reduction to the Evaluation of a Residue

Based on the results of section 2, we arrive at an algorithm for real multipoint approximate polynomial evaluation.

Algorithm 3.1.

Input: a positive rational $a < 2/3$, natural m, n and d, real $p_0, p_1, \ldots, p_n$ and $x_0, x_1, \ldots, x_m$ [compare (1.1) and (1.2)], such that (2.4) holds (compare Remark 2.3) and

$$|x_j| \leq a\ , \qquad j = 0, 1, \ldots, m\ .$$

Output: real values $V_0, V_1, \ldots, V_m$ such that

$$|V_j - p(x_j)| \leq E^*\ , \qquad j = 0, 1, \ldots, m\ ,$$

for $p(x)$ of (1.1) and E^* of (2.5).

Computations.

Stage 0, *choice of the degree $d-1$ of the approximation polynomial.* Choose $d-1$ based on the estimates of Remark 2.1.

Stage 1, *interpolation via modular reduction.* Evaluate the coefficients of the polynomial

$$v(x) = \sum_{i=0}^{d-1} v_i x^i = p(x) \bmod L(2x/a) \tag{3.1}$$

where

$$L(x) = \sum_{i=0}^{d} \ell_i x^{d-i} = \prod_{h=0}^{d-1} (x - 2t_{2h+1}) \tag{3.2}$$

and where t_{2h+1} are defined by (2.1).

Stage 2, *multipoint evaluation* [1,2]. Compute and output the values

$$V_j = v(x_j)\ , \qquad j = 0, 1, \ldots, m\ .$$

Correctness of Algorithm 3.1 follows from (2.5) since, clearly, $L(2x/a) = 0$ on the set $X(a,d)$ of (2.1), and therefore, $p(at_{2h+1}) = v(at_{2h+1})$ for all h.

Remark 3.1. The algorithm of [18] also uses Chebyshev points, but only in order to approximate each monomial x^i as an exponential function in i. This leads to a distinct algorithm and distinct complexity estimates.

4 Computing the Coefficients of $v(x)$ (Stage 1 of Algorithm 4.1) via Polynomial Division.

Let us first specify the polynomial $L(x)$ of (3.2).

Theorem 4.1. *The polynomial $L(x)$ of (3.2) has integer coefficients such that*

$$\ell_{2j+1} = 0\ ,\ j = 0, 1, \ldots, \tag{4.1}$$

$$\ell_{2j} = (-1)^j \binom{d-j}{j} d/(d-j)\ ,\ j = 0, 1, \ldots, s\ , \tag{4.2}$$

where $s = \lfloor d/2 \rfloor$.

Proof is given in Appendix C. □

Theorem 5.1 implies that $L(x)$ is a monic polynomial with integer coefficients. Thus, at stage 1 of Algorithm 3.1, we prefer to have reduction modulo $L(x)$, rather than modulo $L(2x/a)$. We will achieve this by scaling, which turns $p(x)$ into a polynomial with the coefficients $(a/2)^i p_i B$, for any nonzero scalar B. Specifically, fix such a scalar B at our convenience, denote that

$$y = 2x/a\ ,$$

$$g(y) = \sum_{i=0}^{n} g_i y^i = Bp(ay/2) = B \sum_{i=0}^{n} (a/2)^i p_i y^i \ , \tag{4.3}$$

$$w(y) = \sum_{i=0}^{d-1} w_i y^i = Bv(2y/a) = B \sum_{i=0}^{d-1} (2/a)^i v_i y^i \ , \tag{4.4}$$

and equivalently rewrite (3.1) as the following equation:

$$w(y) = g(y) \bmod L(y) \ . \tag{4.5}$$

Now, we may obtain $v(x)$ as follows:

a) compute the coefficients of the polynomial $g(y)$, from the equations

$$g_i = B(a/2)^i p_i \ , \quad i = 0, \ldots, n \ , \tag{4.6}$$

implied by (4.3);

b) compute the coefficents $w_0, \ldots, w_{d-1}$ of the polynomial $w(y)$ satisfying (4.5);

c) compute the coefficients $v_0, \ldots, v_{d-1}$ of the polynomial $v(x)$, from the equations

$$v_i = (a/2)^i w_i / B \ , \quad i = 0, \ldots, d-1 \ ,$$

implied by (4.4).

We only need to specify stage b).

The straightforward algorithm for polynomial division [1,2] (sometimes called "synthetic polynomial division") computes $w_0, \ldots, w_{d-1}$ remaining in the linear space with the basis $g_0, \ldots, g_n$ over the ring $\mathbf{Z}$ of integers or over any of its extensions (that is, over any ring with unity), since the divisor $L(x)$ is a monic polynomial with integer coefficients.

Besides the "synthetic division" algorithm, we may apply any algorithm for polynomial division in order to compute the coefficients of the polynomial $g(y) \bmod L(y)$. In particular, we may start with computing the coefficients of the auxiliary polynomial,

$$R(y) = \sum_{i=0}^{\lceil (n-d)/2 \rceil} r_i y^{2i} \ ,$$

of degree at most $n - d$, such that

$$L^*(y)\, R(y) = 1 \bmod y^{n-d+1} \tag{4.7}$$

where $L^*(y) = y^d L(1/y)$ denotes the reverse of the polynomial $L(y)$ (we refer to [1,2] and to our Remark 4.3 on computing the coefficients of the polynomial $R(y)$).

Then we deduce from (4.5) that

$$g(y) = L(y)\, q(y) + w(y) \ , \ \deg w(y) < d \ , \tag{4.8}$$

and consequently,

$$q^*(y) = y^{n-d} q(1/y) = R(y) g^*(y) \bmod y^{n-d+1} \ , \tag{4.9}$$

where $g^*(y) = y^n g(1/y)$ is the reverse of the polynomial $g(y)$. Given $g^*(y)$ and $R(y)$, we apply (4.9) and immediately compute the d trailing coefficients of $q(y)$. Then, from (4.8), we compute the coefficients of $w(y)$, by means of the subtraction of two polynomials modulo y^d. In the next section (in Table 5.1), we will refer to this entire computation of $w(y)$ as to **Algorithm 4.1**.

Remark 4.1. By choosing an appropriate integer value for B, we may make the coefficients $g_0, \ldots, g_n$ of the polynomial $g(y)$ integers if $a, p_0, \ldots, p_n$ are rational numbers (the value a can always be chosen rational, of course).

Remark 4.2. Due to (4.1), the polynomial $L(y)$ of (3.2) can be represented as follows:

$$L(y) = y^{d-2s} K(t) , \quad K(t) = \sum_{j=0}^{s} \ell_{2j} t^{s-j} , \quad t = y^2 , \quad s = \lfloor d/2 \rfloor .$$

Thus, the computation of $g(y) \bmod L(y)$ can be reduced to computing $g_0(t) \bmod K(t)$ and $g_1(t) \bmod K(t)$ where $g(y) = g_0(t) + y g_1(t)$, $g_0(t)$ and $g_1(t)$ are polynomials of degrees at most s.

Remark 4.3. The reader is challenged to deduce explicit expressions [in closed form, similar to (4.2)] for all or some of the coefficients of the polynomial $R(y) = \sum_{i=0}^{n-d} r_i y^{2i}$ of (4.7), in particular, to deduce that

$$r_i = \binom{d+2i}{i} d/(d+2i) , \quad \text{for} \quad i = 0, 1, 2, \ldots, s .$$

Furthermore, since $R(y)$ only depends on d and n, we may assume computational models that allow the cost-free evaluation of the integer coefficients of $R(y)$ even where their explicit expressions are not available.

5 The Estimates for the Arithmetic Computational Complexity

To estimate the sequential and parallel arithmetic complexity of our computations, we will assume the customary models of the arithmetic RAM and PRAM ([1,2]). The resulting estimates will be represented by the number T of arithmetic operations involved (under the sequential model) and by the pair $[t, w]$, under the parallel model, where t stands for the parallel arithmetic time and w for the product of t with the number of processors involved. This product is called the *(potential) work* of a parallel algorithm, [16]. The values of T, t and w will be defined within constant factors. (The choice of the PRAM models is for convenience only; the resulting estimates can be immediately translated to the cases of the arithmetic circuit model and of any reasonable model supporting fast polynomial arithmetic.)

In our estimates, we will use the notation $\log^{(h)} D$ and $\log^* D$ where

$$\log^{(0)} D = D , \quad \log^{(h)} D = \log^{(h-1)} D , \quad h = 1, 2, \ldots, \log^* D , \tag{5.1}$$

$$\log^* D = \max\{h \colon \log^{(h)} D > 0\} . \tag{5.2}$$

$M(s)$ will denote the sequential arithmetic time (that is, the number of arithmetic operations) required to multiply two polynomials of degrees at most s. We recall (from [1,2]) that

$$M(s) = O(s \log s) \tag{5.3}$$

over the fields (such as the complex field) that support discrete Fourier transform on 2^k points, for an integer k, such that $2^k > 2s$, $2^k = O(s)$; furthermore,

$$M(s) = O(s \log s \log\log s) \tag{5.4}$$

over any ring of constants.

By performing stage 1 of Algorithm 3.1 via polynomial division and by using the known estimates for the computational complexity of polynomial multiplication [see (5.3), (5.4)], of polynomial division, and of multipoint polynomial evaluation [1,2,14], we obtain the upper estimates of Table 5.1 for the sequential and parallel arithmetic cost of performing Algorithm 3.1. The estimates are shown within constant factors, using the definitions (5.1)–(5.4); some further comments are given in Remarks 5.1–5.3.

Table 5.1. Upper estimates for the arithmetic cost of performing Algorithm 3.1.

stage i	sequential cost	parallel cost $[t, w]$
stage 1 (via "synthetic division")	nd	$[n-d, d]$
stage 1 (via Algorithm 4.1)	$M(n)$	$[(\log^* n)\log n,\ M(n)]$
stage 1 [via Algorithm 4.1, with precomputing the reciprocal polynomial $R(y)$ of (4.7)]	$M(n)$	$[\log n,\ M(n)]$
stage 2	$T_2 = \frac{m+d}{d} M(d) \log d$	$[(\log^* d)\log^2 d,\ T_2]$

Remark 5.1. As follows from the estimates of (2.5) and Remark 2.1, we may ensure approximation within the errors $p\,2^{-u}$ to $p(x)$ on the set $X(m)$ of (1.2) if we apply Algorithm 3.1 with $d = O(u)$. Substituting this expression and (5.3) into the bounds of Table 5.1 on T_1 and T_2, we arrive at the bound $T_1 + T_2 = O((m+u)\log^2 u + n\min\{u, \log n\})$ on the overall arithmetic cost of performing Algorithm 3.1. We have cited this bound in the introduction and abstract.

Remark 5.2. The last column of Table 5.1 shows that Algorithm 3.1 runs in polylogarithmic parallel time and has the optimum potential work bound (equal to the record upper bound on the sequential time). In fact, the parallel time bounds t in lines 1 and 3 of Table 5.1 can be decreased by the factors of $\log^* n$ and $\log^* d$, respectively, at the cost of a slight increase of the potential work w, by the factors of $1 + 2^{-h}\log^{(h)} D$, for any fixed positive integer h, $h \le \log^* D$, where $D = n$ or $D = d$, respectively (see [1,2]).

Remark 5.3. We may perform stage 2 by using $O(nd)$ ops to ensure numerical stability [11].

6 Bounding the Precision and the Bit-Complexity of the Computations.

Suppose that the coefficients of $g(x)$ are integers (see Remark 4.1). Then so are the coefficients of $w(y)$. Due to Lemma 2 of [3], we have the following a priori bound:

$$W = \max_{0 \le i < d} \{|w_i|\} \le \max_{0 \le i < d} \{|g_i|\} + \sum_{i=d}^{n} |g_i| \left(1 + \max_{0 \le j \le s} |\ell_{2j}|\right).$$

Combining this bound with (4.2) and (4.6), we obtain that

$$W \le |B| \left[\max_{0 \le i < d} \{ |p_i (a/2)^i| \} + \sum_{i=d}^{n} |p_i| (a/2)^i \left(1 + d \max_{0 \le j \le s} \left\{ \binom{d-j}{j} / (d-j) \right\} \right) \right] . \qquad (6.1)$$

Since the coefficients of $w(y)$ are integers, bounded according to (6.1), we may compute them by applying any algorithm for polynomial division in the ring $\mathbf{Z}_M$ of integers modulo M for any integer

$$M > 2W .$$

Then we may recover w_i from $w_i \bmod M$ according to the equations

$$w_i = w_i \bmod M \text{ if } w_i \bmod M < M/2 , \qquad (6.2a)$$

$$w_i = -M + w_i \bmod M \text{ otherwise.} \qquad (6.2b)$$

We may choose M in the form

$$M = 2^r \pm 1 ,$$

for an integer r, to simplify the reduction modulo M. Then the number T_B of bit-operations required for such a polynomial division will be bounded by

$$T_B = O(T\mu(\log M)) = O(M(n)\mu(\log W)) \qquad (6.3)$$

where $T = O(M(n))$ denotes the arithmetic complexity of computing the coefficients of $g(y) \bmod L(y)$ (see Table 5.1), $\mu(r)$ denotes the number of bit-operations required for multiplication of two integers modulo 2^r or $2^r + 1$, and W is bounded according to (6.1). Theoretically, we may reach

$$\mu(r) = O(r \log r \log\log r) , \qquad (6.4a)$$

[1], but in practice the users mostly rely on the straightforward algorithm, which only supports the bound

$$\mu(r) = O(r^2) . \qquad (6.4b)$$

Example 6.1. Let $M = O(W)$, $W = n^{O(n^c)}$, for a nonnegative c, so that $\log W = O(n^c \log n)$. Assume (5.3) and obtain from (6.3) that

$$T_B = O(n^{c+1} (\log n)^3 \log\log n) \qquad (6.5a)$$

under (6.4a) and that

$$T_B = O(n^{2c+1} (\log n)^3) , \qquad (6.5b)$$

under (6.4b). Assuming (5.4), rather than (5.3), would have implied an extra factor $\log\log n$ in these estimates.

In the remainder of this section, we will combine the above results with some customary techniques, based on the Chinese remainder algorithm [1,2] (hereafter referred to as C.r.a.) and usually applied in order to decrase the precision of algebraic computations. This will also lead us to a substantial decrease of the bit-complexity estimate (6.5b), under the assumption (6.4b).

Specifically, first choose a natural H and then H integers $m_1, \ldots, m_H$, all exceeding 1, all pairwise relatively prime, and satisfying the relations,

$$m_1 m_2 \cdots m_H = M > 2W \ . \tag{6.6}$$

Having chosen these integers, we

a) first compute $w_i \bmod m_h$ in $\mathbf{Z}_{m_h}$ for all h and i,

b) then recover $w_i \bmod M$ for all i, by applying the C.r.a.,

c) and finally, obtain the values w_i for all i, from (6.2a,b).

Hereafter, $m^* = \max_h m_h$, T denotes the arithmetic time required for the division of $g(y)$ by $L(y)$;

$$T = O(\min\{nd, M(n)\}) \tag{6.7}$$

(see Table 5.1);

$$m^* = \max_h \{m_h\} \ . \tag{6.8}$$

The bit-complexity of performing stage a) is bounded by $O(TH\mu(\log m^*))$. Stage c) is clearly less costly than stage a). To estimate the bit-cost of stage b), we recall the C.r.a. [1,2], reduced to the following steps:

1.° Compute $M = m_1 m_2 \cdots m_H$.

2.° Compute $M_h = M \bmod m_h^2$, for all h.

3.° Compute $F_h = M_h / m_h$, for all h.

4.° Compute $f_h = (1/F_h) \bmod m_h$, for all h (by means of the Euclidean algorithm applied to F_h and m_h).

5.° Compute

$$w_i \bmod M = \sum_{h=1}^{H} (((w_i \bmod m_h) f_h) \bmod m_h) F_h \ ,$$

for all i; for each i, we obtain such a value by recursively summing in h the partial fractions

$$(w_i \bmod m_h) f_h / m_h \ , \qquad h = 1, \ldots, H \ ,$$

and outputing the numerator of the resulting sum.

Let us now assume the bound $O(\mu(\log N))$ on the bit-complexity of performing an arithmetic operation with integers reduced modulo N [compare (6.4a,b)], and estimate the number of bit-operations requried in each of steps 1°–5°.

We need:

$O(\log H \, \mu(\log M))$ bit-operations at step 1°;

$O(H \, \mu(\log M))$ at step 2°;

$O(H \, \mu(\log m^*))$ at step 3°;

$O(H \mu(\log m^*) \log m^*)$ at step 4°, and

$O(d \log H \, \mu(\log M))$ at step 5°.

Summarizing, we need

$$\begin{aligned} T_B \ &= O(d \log H \, \mu(\log M) \\ &\quad + TH \, \mu(\log m^*) + H \, \mu(\log M) + H \, \mu(\log m^*) \log m^*) \end{aligned} \tag{6.9}$$

bit-operations in order to compute the d coefficients of $w(y) = g(y) \mod L(y)$, with $\mu(r)$, H, m^*, M, T satisfying (6.4)–(6.8).

If $d = O(\log n)$, we may replace the bound (6.7) on T by the bound $T = O(n)$, and if $\log m^* = O(\log\log M)$, then we may decrease the cost of step 2° to $O(H \log M)$; in both cases we achieve the cost decrease by using the algorithms of Kaminski [8]. The overall bit-complexity bound (6.9) will then decrease to the bound

$$T_B = O\left(H\,\mu(\log m^*)\,(n + \log m^*) + H \log M + d \log H\,\mu(\log M)\right) \ . \tag{6.10}$$

Remark 6.1. By incorporating Kaminski's algorithm of [8] for integer residues, we may decrease the known upper bound on the bit-complexity of the C.r.a. from

$$O\left(H\,\mu(\log M) + H\,\mu(\log m^*) \log m^*\right) \tag{6.11}$$

to

$$O\left(\log H\,\mu(\log M) + H \log M + H\,\mu(\log m^*) \log m^*\right) \tag{6.12}$$

[compare (6.9) and (6.10) for $d = 1$, $n = 0$], which is a substantial improvement under (6.4b) (see the next example).

Example 6.2. Under the assumption of Example 6.1 that $M = O(W)$, $W = n^{O(n^c)}$, choose the integers $H, m_1, \ldots, m_H$ of (6.6) such that

$$H = \log W / \log n = O(n^c) \ ,$$

$m^* = H^2 = O(n^{2c})$, $(m^*)^H > M$. Then at step 2° of the C.r.a. we may apply Kaminski's algorithm of [8] for computing integer residues. For any positive constant c, this will decrease the overall bit-cost estimate for the C.r.a. from $O(n^{3c} \log^2 n)$ [obtained from (6.11)] to $O(n^{2c} \log^3 n)$ [obtained from (6.12)].

Example 6.3. Let the assumptions of Example 6.2 hold and let $d = O(\log n)$. Then we may apply Kaminski's algorithm for computing polynomial residuals and arrive at (6.10), with M, H and m^* bounded according to the assumptions of our Examples. Thus,

$$T_B = O\left(n^{c+1} \mu(\log n) + n^{2c} \log n + (\log n)^2 \mu(n^c \log n)\right) \ .$$

The latter bound turns into the bounds $T_B = O\left(n^{c+1} \log^2 n + n^{2c} \log^4 n\right)$, under (6.4b) [and this improves (6.5b) roughly by the factor $n^{\min\{c,1\}}$], and $T_B = O(n^c \log n (n^c \log n + n \log\log n \log\log\log n))$, under (6.4a) [and this slightly improves (6.5a) for $c \le 1$ and exceeds the bound (6.5a) for $c > 1$].

7 Discussion

a) Our algorithms can be applied recursively, to simplify the multipoint evaluation of $v(x)$ at stage 2 of Algorithm 3.1 if $p = \sum_{i=0}^{n} |p_i|$ substantially exceeds $v = \sum_{h=0}^{d-1} |v_h|$.

b) Instead of using approximation via interpolation on the Chebyshev set, one may try to use any other good approximation by a polynomial or by a ratio of two polynomials of lower degrees, as long as the coefficients of the latter polynomials are easy to compute.

c) Of course, it is interesting to consider approximate interpolation and multipoint evaluation with complex input nodes, although this case is much harder to treat than the real case (compare [17]).

d) A major challenge is an extension of our results to interpolation. As a heuristic approach, we suggest replacing the input set $X(n)$ of $n+1$ nodes of interpolation (1.2) by the set $X(a,d)$ of d Chebyshev nodes (2.1). The input values of $p(x)$ at $x = x_i$, $i = 0,1,\ldots,m$, are then used in order to approximate the values $p(at_{2h+1})$, for $h = 0,1,\ldots,d-1$. Then interpolation to $p(x)$ is replaced by the simpler interpolation to a polynomial of degree at most $d-1$ that approximates $p(x)$ at $x = at_{2h+1}$, for $h = 0,1,\ldots,d-1$. The approach is promising if all the points of $X(m)$ are closely approximated by the nodes at_{2h+1} of $X(a,d)$. Some difficulties with this heuristic may stem from the fact of ill-conditioning of the interpolation problem (see Appendix B).

Appendix A. Proof of Theorem 4.1.

Recall that $\cos(\pi - t) = -\cos t$, for any t. Therefore,

$$t_{2h+1} = -t_{2d-2h-1}\ , \quad h = 0,1,\ldots,d-1\ ,$$

due to (2.1). Deduce from these equations and from (3.2) that

$$L(x) = x^{d-2s}\prod_{h=0}^{s-1}(x^2 - 4t_{2h+1}^2)\ , \quad s = \lfloor d/2 \rfloor\ ,$$

which implies the equations (4.1).

To prove the equations (4.2), we need some auxiliary results.

Hereafter, $\omega_g = \exp(2\pi\sqrt{-1}/g)$ denotes a primitive g-th root of 1. We will first represent t_{2h+1} of (2.1) as follows:

$$2t_{2h+1} = \omega_{4d}^{2h+1} + \omega_{4d}^{-(2h+1)}\ , \quad h = 0,1,\ldots,d-1\ . \tag{A.1}$$

Next, we will prove the following identity in z [where $L(x)$ is the polynomial of (3.2)]:

Fact A.1.

$$L(z + z^{-1}) = z^d + z^{-d}\ . \tag{A.2}$$

Proof. Represent the right side of (A.2) as a polynomial in $z + z^{-1}$. Observe that such a polynomial is monic and has degree d. It remains to show that this polynomial has the same zeros as $L(x)$, that is, vanishes where

$$z + z^{-1} = 2t_{2h+1}\ , \qquad h = 0,1,\ldots,d-1\ .$$

But the latter equations imply that $z = \omega_{4d}^{\pm(2h+1)}$ [due to (A.1)], and consequently, that

$$z^d + z^{-d} = \omega_{4d}^{(2h+1)d} + \omega_{4d}^{-(2h+1)d} = (\omega_{4d}^{2(2h+1)d} + 1)\,\omega_{4d}^{-(2h+1)d} = 0\ . \ \square$$

Based on Fact A.1, we will now prove

Fact A.2. *The coefficients $\ell_0, \ell_2, \ldots$ of $L(x)$ satisfy the following unit triangular linear system of equations:*

$$\sum_{j=0}^{i}\binom{d-2j}{i-j}\ell_{2j} = \delta_{0i}\ , \quad i = 0,1,\ldots,s\ , \tag{A.3}$$

where $\delta_{00} = 1$, $\delta_{0i} = 0$ *for* $i > 0$, $s = \lfloor d/2 \rfloor$.

Proof. Substitute (3.2) and (4.1) into (A.2) and obtain that

$$L(z+z^{-1}) = \sum_{j=0}^{s} (z+z^{-1})^{d-2j}\, \ell_{2j} = z^d + z^{-d}\ , \quad s = \lfloor d/2 \rfloor\ .$$

Expand the powers of $z + z^{-1}$ in the latter expression (to represent them as polynomials in z and z^{-1}) and obtain that

$$(z+z^{-1})^{d-2j} = \sum_{k=0}^{d-2j} \binom{d-2j}{k} z^{d-2j-2k}\ ,$$

$$z^d + z^{-d} = \sum_{j=0}^{s} \ell_{2j} \sum_{k=0}^{d-2j} \binom{d-2j}{k} z^{d-2j-2k}\ , \quad s = \lfloor d/2 \rfloor\ .$$

Substitute $i = k + j$, $k = i - j$ and deduce that

$$z^d + z^{-d} = \sum_{j=0}^{s} \ell_{2j} \sum_{i=j}^{d-j} \binom{d-2j}{i-j} z^{d-2i}\ .$$

Therefore,

$$z^d = \sum_{i=0}^{s} \left(\sum_{j=0}^{i} \binom{d-2j}{i-j} \ell_{2j} \right) z^{d-2i}\ .$$

Equate the coefficients of the powers of z on both sides of the latter identity and arrive at (A.3). □

To prove Theorem 4.1, it now remains to replace ℓ_{2j} in (A.3) by their expressions from (4.2) and to verify that the resulting equations hold true. This is immediate for $i = 0$, and we may substitute $\ell_0 = 1$ into the remaining s equations of (A.3). We only need to prove that

$$\sum_{j=0}^{i} \binom{d-2j}{i-j} \binom{d-j}{j} (-1)^j\, d/(d-j) = 0\ , \quad j = 1, \ldots, s\ . \tag{A.4}$$

Towards this goal, we now represet $\binom{d-2j}{i-j}\binom{d-j}{j} / d-j$ as follows:

$$\frac{(d-j-1)!}{(d-i-j)!\,(i-j)!\,j!} = \frac{(d-j-1)!}{(i-1)!\,(d-i-j)!} \frac{i!}{(i-j)!\,j!} \frac{1}{i} = \binom{d-j-1}{i-1} \binom{i}{j} \frac{1}{i}\ .$$

Substitute the latter expressions into the equations of (A.4) and thus rewrite (A.4) as follows:

$$\frac{d}{i} \sum_{j=0}^{i} \binom{d-j-1}{i-1} \binom{i}{j} (-1)^j = 0\ , \quad i = 1, \ldots, s\ ,$$

where the factor d/i can be deleted. This brings us to the known equations (see [Kn], equation (2.5) on page 58, for $k = j$, $m = i - 1$, $r = d - 1$, $s = i$, $t = 0$), which completes the proof of Theorem 4.1. □

Appendix B. Error Magnification in the Interpolation.

Fix a set $X(m)$ of (1.2), denote

$$P(x) = \prod_{k=0}^{m} (x - x_k)\ ,$$

$$\ell_k(x) = P(x) / (x - x_k)\ , \quad k = 0, 1, \ldots, m\ ,$$

assume that $m = n$, and recall Lagrange's interpolation formula,

$$p(x) = \sum_{k=0}^{m} p(x_k)\,\ell_k(x) \,/\, \ell_k(x_k) \ .$$

Observe that an error ϵ in the value of $p(x_k)$ causes the output perturbation $\epsilon\,\ell_k(x)/\ell_k(x_k)$. This perturbation is large if $|\ell_k(x_k)|$ is small, which is frequently the case. For instance, if $x_k = -a + 2ak/m$, $k = 0, 1, \ldots, m$, then

$$\ell_m(x_m) = \max_k |\ell_k(x)| = (2a/m)^m m! \ ,$$

so that

$$(2a/e)^m m^{1/2} e^{3/4} \le \ell_m(x_m) \le (2a/e)^m e m^{1/2} \ . \qquad (B.1)$$

If $m = d - 1$ and if $X(m) = X(a, d)$ is the Chebyshev set (2.1), then the magnificaion of the errors at $x_{d-1} = a$ (and similarly at $x_0 = -a$) is greater than (B.1) indicates, since the nodes t_{2h+1} are accumulated near $x_{d-1} = a$ and $x_0 = -a$, so that the distances from x_k to a are of the order of $(2ak/d)^2$ if $d - k = o(d)$ (and similarly for x_k near $-a$).

Appendix C. Special Techniques for Chebyshev's Nodes.

To simplify the evaluation of a polynomial $p_n(x)$ of (1.1) on the set $X(1, d)$ of Chebyshev's nodes and the interpolation by $p(x)$ on the set $X(1, d)$ (provided in the latter case that $n = d - 1$), represent $p(x)$ as $p_0(x^2) + xp_1(x^2) = q_0(y) + xq_1(y)$ where $y = 1 - 2x^2$. The substitution of $y = 1 - 2x^2$ for x maps the Chebyshev set $X(1, d)$ of (1.2) into the half-size Chebyshev set $X(1, d/2)$. We may assume for simplicity that d and n are integer powers of 2 and recursively apply the above relations to obtain the divide-and-conquer algorithms for both evaluation and interpolation (see more details in [14]). The resulting algorithms slightly improve the parallel version of the algorithm of [10], incorporating the parallel polynomial division algorithm of [2].

Acknowledgements

The author has been supported by NSF Grant CCR 9020690 and PSC CUNY Award #662478. He is grateful to C. Engle for her assistance with typing this paper.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Mass., 1976.

[2] D. Bini, V. Pan, *Numerical and Algebraic Computations with Matrices and Polynomials*, Birkhauser, Boston, 1992 (to appear).

[3] D. Bini, V. Pan, Polynomial Division and its Computational Complexity, *J. of Complexity*, 2, 179–203, 1986.

[4] A. B. Borodin, I. Munro, Evaluating Polynomials at Many Points, *Information Processing Letters*, 1:2. 66–68, 1971.

[5] G. Dahlquist, A. Björck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[6] G. H. Golub, C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1989.

[7] W. G. Horner, *Philosophical Transactions, Royal Society of London*, 109, 308-335, 1819.

[8] M. Kaminski, Linear Time Algorithm for Residue Computation and a Fast Algorithm for Division with a Sparse Divisor, *J. of ACM*, 34, 4, 968-984, 1987.

[9] D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, **1**, Addison-Wesley, Reading, MA, 1973.

[10] R. Moenck, A. B. Borodin, Fast Modular Transforms via Division, *Conference Record, IEEE 13th Annual Symposium on Switching and Automata Theory*, 90–96, 1972.

[11] A. C. R. Newbery, Error Analysis for Polynomial Evaluation, *Math. Comp.*, 28, 127, 789–793, 1979.

[12] I. Newton, *Analysis per Quantitatem Series*, **10** London, 1711 (also see D. T. Whiteside, ed., *The Mathematical Papers of Isaac Newton*, **2**, 222, Cambridge Univ. Press, 1968).

[13] V. Y. Pan, Methods of Computing Values of Polynomials, *Russian Mathematical Surveys*, 21, 1, 105–136, 1966.

[14] V. Pan, Fast Evalulation and Interpolation at the Chebyshev Set of Points, *Applied Math. Letters*, 2, 3, 255–258, 1989.

[15] V. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Rev.*, 34, 2, 225–262, 1992.

[16] V. Y. Pan, F. P. Preparata, Supereffective Slow-down of Parallel Computations, *Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures*, 402–409, 1992.

[17] V. Pan, A. Sadikou, E. Landowne, O. Tiga, A New Approach to Fast Polynomial Interpolation and Multipoint Evaluation, submitted to *Computers and Math. (with Applications)*.

[18] V. Rokhlin, A Fast Algorithm for the Discrete Laplace transformation, *J. of Complexity*, 4, 12–32, 1988.

[19] A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, unpublished manuscript, 1982.

[20] V. Strassen, Die Berechnungskomplexität von elementarysymmetrischen Funktionen und von Interpolationskoeffizienten, *Numerische Math.*, 20, 3, 238–257, 1973.

[21] W. Werner, Polynomial Interpolation: Lagrange versus Newton, *Math. of Computation*, 43, 167, 205–217, 1984.