Note on fast division algorithm for polynomials using Newton iteration

Zhengjun Cao∗, Hanyue Cao
Department of Mathematics, Shanghai University, Shanghai, China.
∗ caozhj@shu.edu.cn

Abstract

The classical division algorithm for polynomials requires \( O(n^2) \) operations for inputs of size \( n \). Using reversal technique and Newton iteration, it can be improved to \( O(M(n)) \), where \( M \) is a multiplication time. But the method requires that the degree of the modulo, \( x^l \), should be the power of 2. If \( l \) is not a power of 2 and \( f(0) = 1 \), Gathen and Gerhard suggest to compute the inverse, \( f^{-1} \), modulo \( x^\lceil l/2 \rceil, x^\lceil l/2 - 1 \rceil, \ldots, x^{l/2}, x^l \), separately. But they did not specify the iterative step. In this note, we show that the original Newton iteration formula can be directly used to compute \( f^{-1} \bmod x^l \) without any additional cost, when \( l \) is not a power of 2.

Keywords: Newton iteration, revisal, multiplication time

1 Introduction

Polynomials over a field form a Euclidean domain. This means that for all \( a, b \) with \( b \neq 0 \) there exist unique \( q, r \) such that \( a = qb + r \) where \( \deg r < \deg b \). The division problem is then to find \( q, r \), given \( a, b \). The classical division algorithm for polynomials requires \( O(n^2) \) operations for inputs of size \( n \). Using reversal technique and Newton iteration, it can be improved to \( O(M(n)) \), where \( M \) is a multiplication time. But the method requires that the degree of \( x^l \) should be the power of 2. If \( l \) is not a power of 2 and \( f(0) = 1 \), Gathen and Gerhard [1] suggest to compute the inverse, \( f^{-1} \), modulo \( x^\lceil l/2 \rceil, x^\lceil l/2 - 1 \rceil, \ldots, x^{l/2}, x^l \), separately. But they did not specify the iterative step. In this note, we show that the original Newton iteration formula can be directly used to compute \( f^{-1} \bmod x^l \) without any additional cost, when \( l \) is not a power of 2. We also correct an error in the cost analysis [1].

2 Division algorithm for polynomials using Newton iteration

The description comes from Ref.[1].
Let $D$ be a ring (commutative, with 1) and $a, b \in D[x]$ two polynomials of degree $n$ and $m$, respectively. We assume that $m \leq n$ and that $b$ is monic. We wish to find polynomials $q$ and $r$ in $D[x]$ satisfying $a = qb + r$ with $\deg r < \deg b$ (where, as usual, we assume that the zero polynomial has degree $-\infty$). Since $b$ is monic, such $q, r$ exist uniquely.

Substituting $1/x$ for the variable $x$ and multiplying by $x^n$, we obtain

$$x^n a \left( \frac{1}{x} \right) = \left( x^{n-m} q \left( \frac{1}{x} \right) \right) \cdot \left( x^m b \left( \frac{1}{x} \right) \right) + x^{n-m+1} \left( x^{m-1} r \left( \frac{1}{x} \right) \right) \quad (1)$$

We define the reversal of $a$ as $\text{rev}_k(a) = x^k a(1/x)$. When $k = n$, this is the polynomial with the coefficients of $a$ reversed, that is, if $a = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, then

$$\text{rev}(a) = \text{rev}_n(a) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_0$$

Equation (1) now reads

$$\text{rev}_n(a) = \text{rev}_{n-m}(q) \cdot \text{rev}_m(b) + x^{n-m+1} \text{rev}_{m-1}(r),$$

and therefore,

$$\text{rev}_n(a) \equiv \text{rev}_{n-m}(q) \cdot \text{rev}_m(b) \mod x^{n-m+1}.$$ Notice that $\text{rev}_m(b)$ has constant coefficient 1 and thus is invertible modulo $x^{n-m+1}$. Hence we find

$$\text{rev}_{n-m}(q) \equiv \text{rev}_n(a) \cdot \text{rev}_m(b)^{-1} \mod x^{n-m+1},$$

and obtain $q = \text{rev}_{n-m}(\text{rev}_{n-m}(q))$ and $r = a - qb$.

So now we have to solve the problem of finding, from a given $f \in D[x]$ and $l \in N$ with $f(0) = 1$, a $g \in D[x]$ satisfying $fg \equiv 1 \mod x^l$. If $l$ is a power of 2, then we can easily obtain the inversion by the following iteration step

$$g_{i+1} = 2g_i - fg_i^2$$

In fact, if $fg_i \equiv 1 \mod x^{2^i}$, then $x^{2^i} \mid 1 - fg_i$, $x^{2^{i+1}} \mid (1 - fg_i)^2$. Hence, $x^{2^{i+1}} \mid 1 - f(2g_i - fg_i^2)$. Using the above iteration method, we have the following result:

**Theorem 1.** Let $D$ be a ring (commutative, with 1), $f, g_0, g_1, \ldots \in D[x]$, with $f(0) = 1$, $g_0 = 1$, and $g_{i+1} \equiv 2g_i - fg_i^2 \mod x^{2^{i+1}}$, for all $i$. Then $fg_i \equiv 1 \mod x^{2^i}$ for all $i \geq 0$.

By Theorem 1, we now obtain the following algorithm to compute the inverse of $f \mod x^l$. We denote by $\log$ the binary logarithm.
Algorithm 1: Inversion using Newton iteration

Input: \( f \in D[x] \) with \( f(0) = 1 \), and \( l \in N \).

Output: \( g \in D[x] \) satisfying \( fg \equiv 1 \mod x^l \).

1. \( g_0 \leftarrow 1 \), \( r \leftarrow \lceil \log l \rceil \)
2. for \( i = 1, \ldots, r \) do \( g_i \leftarrow (2g_{i-1} - fg_{2i-1}) \rem x^{2i} \)
3. Return \( g_r \)

From the algorithm 1, one can easily obtain the following.

Algorithm 2: Fast division with remainder

Input: \( a, b \in D[x] \), where \( D \) is a ring (commutative, with 1) and \( b \neq 0 \) is monic.

Output: \( q, r \in D[x] \) such that \( a = qb + r \) and \( \deg r < \deg b \).

1. if \( \deg a < \deg b \) then return \( q = 0 \) and \( r = a \)
2. \( m \leftarrow \deg a - \deg b \)
   call Algorithm 1 to compute the inverse of \( \rev_{\deg b}(b) \in D[x] \) modulo \( x^{m+1} \)
3. \( q^* \leftarrow \rev_{\deg a}(a) \cdot \rev_{\deg b}(b)^{-1} \rem x^{m+1} \)
4. return \( q = \rev_m(q^*) \) and \( r = a - bq \)

3 On the form of \( l \)

The authors [1] stress that “if \( l \) is not a power of 2, then the above algorithm computes too many coefficients of the inverse.” They suggest to compute the inverse modulo \( x^{\lceil l/2 \rceil}, x^{\lceil l/2 - 1 \rceil}, \ldots, x^{\lceil l/2 \rceil}, x^{l} \). For example, suppose \( l = 11 \), then \( x^{\lceil 11/2 \rceil} = x \), \( x^{\lceil 11/2 - 1 \rceil} = x^2 \), \( x^{\lceil 11/2 - 2 \rceil} = x^3 \), \( x^{\lceil 11/2 - 3 \rceil} = x^6 \). In such case, one has to compute \( f^{-1} \) modulo \( x, x^2, x^3, x^6, x^{11} \). It should be stressed that the authors did not specify the iterative step. More serious, the sequence 1, 2, 3, 6, 11 does not form an addition chain [2]. Given a chain \( \{a_i\} \) and \( f \), we can define the following iterative step

\[ g_{a_k} = g_{a_i} + g_{a_j} - fg_{a_i}g_{a_j} \mod x^{a_k}, \text{ if } a_k = a_i + a_j \]

In fact, the suggestion is somewhat misleading. If \( l \) is not a power of 2, the original algorithm 1 can be used to compute the inverse modulo \( x^l \) without any additional cost. It suffices to observe the following fact.

**Fact 1.** If \( 0 < l \leq t \) and \( x^l | 1 - fg \), then \( x^l | 1 - fg \).

The above fact is directly based on the divisibility characteristic. Based on the fact, we obtain the following algorithm.
Algorithm 3: Inversion using divisibility characteristic

| Input: | $f \in D[x]$ with $f(0) = 1$, and $l \in \mathbb{N}$.
| Output: | $g \in D[x]$ satisfying $fg \equiv 1 \mod x^l$. |

1. $g_0 \leftarrow 1, r \leftarrow \lceil \log l \rceil$
2. for $i = 1, \cdots, r - 1$ do $g_i \leftarrow g_{i-1} \cdot (2 - f \cdot g_{i-1}) \text{ rem } x^{2^i}$
3. $g_r \leftarrow g_{r-1} \cdot (2 - f \cdot g_{r-1}) \text{ rem } x^l$
4. Return $g_r$

**Correctness.** It suffices to observe that $l \leq 2^r$ where $r = \lceil \log l \rceil$. Hence $x^l \mid x^{2^r}$. Since $x^{2^r} \mid 1 - f(2g_{r-1} - fg_{r-1}^2)$, we have $x^l \mid 1 - f(2g_{r-1} - fg_{r-1}^2)$. That means $g_r$ is the inverse of $f$ modulo $x^l$, too.

4 On the cost analysis

To make a sound cost analysis, we need the following definition of multiplication time and its properties.

**Definition 1.** Let $R$ be a ring (commutative, with 1). We call a function $M : \mathbb{N}_{>0} \rightarrow R_{>0}$ a multiplication time for $R[x]$ if polynomials in $R[x]$ of degree less than $n$ can be multiplied using at most $M(n)$ operations in $R$. Similarly, a function $M$ as above is called a multiplication time for $\mathbb{Z}$ if two integers of length $n$ can be multiplied using at most $M(n)$ word operations.

For convenience, we will assume that the multiplication time satisfies

$$M(n)/n \geq M(m)/m \text{ if } n \geq m, \quad M(mn) \leq m^2 M(n),$$

for all $n, m \in \mathbb{N}_{>0}$. The first inequality yields the superlinearity properties

$$M(mn) \geq mM(n), \quad M(m + n) \geq M(n) + M(m), \text{ and } M(n) \geq n$$

for all $n, m \in \mathbb{N}_{>0}$.

By the above definition and properties, the authors obtained the following result [1].

**Theorem 2.** Algorithm 1 correctly computes the inverse of $f$ modulo $x^l$. If $l = 2^r$ is a power of 2, then it uses at most $3M(l) + l \in O(M(l))$ arithmetic operations in $D$.

**Proof.** In step 2, all powers of $x$ up to $2^l$ can be dropped, and since

$$g_i \equiv g_{i-1}(2 - fg_i) \equiv g_{i-1} \mod x^{2^{i-1}}, \quad (2)$$

also the powers of $x$ less than $2^{i-1}$. The cost for one iteration of step 2 is $M(2^{i-1})$ for the computation of $g_{i-1}^2$, $M(2^i)$ for the product $fg_{i-1}^2 \mod x^{2^i}$, and then the negative of the
upper half of $fg_i^{2i-1}$ modulo $x^{2i}$ is the upper half of $g_i$, taking $2^{i-1}$ operations. Thus we have $M(2^i) + M(2^{i-1}) + 2^{i-1} \leq \frac{3}{2}M(2^i) + 2^{i-1}$ in step 2, and the total running time is

$$\sum_{1 \leq i \leq r} \left( \frac{3}{2}M(2^i) + 2^{i-1} \right) \leq \left( \frac{3}{2}M(2^r) + 2^{r-1} \right) \sum_{1 \leq i \leq r} 2^{i-r} < 3M(2^r) + 2^r = 3M(l) + l,$$

where we have used $2M(n) \leq M(2n)$ for all $n \in \mathbb{N}$.

There is a typo and an error in the above proof and theorem.

- In the above argument there is a typo (see Eq.(2)).
- The cost for one iteration of step 2 is $M(2^i)$ for the computation of $g_i^{2i-1}$ instead of the original $M(2^{i-1})$, because it is computed under the module $x^{2i}$, not $x^{2i-1}$. Since the upper half of $f(g_i^{2i-1})$ modulo $x^{2i}$ is the same as $g_i$ and the lower half of $g_i$ is the same as $g_{i-1}$, the cost for the computation of $f(g_i^{2i})$ modulo $x^{2i}$ only needs $M(2^{i-1})$. Therefore, according to the original argument the bound should be

$$\sum_{1 \leq i \leq r} \left( \frac{3}{2}M(2^i) + 2^{i-1} \right) \leq \left( \frac{3}{2}M(2^r) + 2^{r-1} \right) \sum_{1 \leq i \leq r} 2^{i-r} < 3M(2^r) + 2^r \leq 12M(l) + 2l,$$

The last estimation comes from $l \leq 2^r \leq 2l$.

Now, we make a formal cost analysis of algorithm 3.

**Theorem 3.** Algorithm 3 correctly computes the inverse of $f$ modulo $x^l$. It uses at most $5M(l) + l \in O(M(l))$ arithmetic operations in $D$.

**Proof.** The cost for step 2 is $3M(2^{r-1}) + 2^{r-1}$ (see the above cost analysis). The cost for step 3 is bounded by $2M(l)$. Since $2^{r-1} \leq l \leq 2^r$, the total cost is $5M(l) + l$.

### 5 Conclusion

In this note, we revisit the fast division algorithm using Newton iteration. We show that the original Newton iterative step can still be used for any arbitrary exponent $l$ without the restriction that $l$ should be the power of 2. We also make a formal cost analysis of the method. We think the new presentation is helpful to grasp the method entirely and deeply.

**Acknowledgements** We thank the National Natural Science Foundation of China (Project 60873227), and the Key Disciplines of Shanghai Municipality (S30104).

**References**

[1] J. Gathen, J. Gerhard: Modern computer Algebra (3 edition), Cambridge University Press, 2003
[2] D. Knuth: The Art of Computer programming, Vol. 2 (3 edition), Addison-Wesley, 1997