A note on the fast power series' exponential

I. S. Sergeev

Abstract

It is shown that the exponential of a complex power series up to order n can be implemented via $(23/12+o(1))M(n)$ binary arithmetic operations over $\mathbb{C}$, where $M(n)$ stands for the (smoothed) complexity of multiplication of polynomials of degree $< n$ in FFT-model. Yet, it is shown how to raise a power series to a constant power with the complexity $(27/8 + o(1))M(n)$.

1 Introduction

It is very well known that the exponential of a power series (as well as some other elementary operations) has the same order of complexity as multiplication, see e.g. [2, 3]. (When we speak about complexity we consider a computational model of circuits or straight-line programs over arithmetic basis $\{\pm, *\} \cup \{ax \mid a \in \mathbb{C}\}$, see e.g. [3]. The field should not be necessary complex, it might be real or any algebraically closed, or any other which supports appropriate FFT.)

Previous papers set up a convention to estimate complexity of the basic power series' operations (including exponential) in the number of multiplications of the same size. The very last papers (since 2000) use the special FFT-way multiplications.

The complexity function $M(n)$ of FFT-way multiplication can be introduced as follows. Let $F^\circ(n)$ be the complexity of FFT of order n over $\mathbb{C}$. Let $F(n)$ be its smoothed version, that is, $F(n) = \min_{m \geq n} F^\circ(m)$. For simplicity we also assume $F(n) = \Omega(n \log n)$ (in fact, $F(n) = \omega(n)$ is sufficient) and $kF(n) = (1 + o(1))F(kn)$ for $1 \leq k \leq \log n$.[1] Then, let $M(n) = 6F(n)$. In any case $M(n)$ serves as an upper asymptotic estimate of the complexity of multiplication of polynomials of degree $< n$.

[1]These assumptions are for more convenient way of writing final complexity bounds only, it does not affect any other aspect of the proof. Notation $g = \omega(f)$ means $f = o(g)$.

Write an upper estimate of the complexity of computing exponent of a power series in $\mathbb{C}[[x]]$ modulo x^n in the form $(A+o(1))M(n)$. Let us list some previously known results: $A = 17/6$ [1], $A = 14/3$ [5, 6], $A = 13/6$ [8, 10] and [4]. Next, we will show that $A = 23/12$ is also valid. The method is a straightforward combination of methods [8, 10] and [4]. To be more precise, scheme of computation follows [4] and technique is inherited from [8, 10].

We will also show that one can compute a constant power of a power series in $\mathbb{C}[[x]]$ modulo x^n with the complexity $(B + o(1))M(n)$, where $B = 27/8$. It slightly improves the previously best known factor $B = 41/12$ [8, 10].

Some notation. Let $f \in \mathbb{C}[[x]]$, then $f_{..n}$ denotes $f \bmod x^n$ and $\lfloor f/x^n \rfloor$ stands for $(f - f_{..n})/x^n$. If $f = \sum_{i\geq 0} f_i x^i$, then Δf, $\mathbf{J} f$, $\ln f$ (when $f_0 = 1$) and e^f (when $f_0 = 0$) denotes formal derivative, formal integral, formal logarithm and formal exponent respectively:

$$\Delta f = \sum_{i\geq 1} i f_i x^{i-1}, \quad \mathbf{J} f = \sum_{i\geq 1} \frac{f_{i-1}}{i} x^i, \quad \ln f = -\sum_{i\geq 1} \frac{(1-f)^i}{i}, \quad e^f = \sum_{i\geq 0} \frac{f^i}{i!}.$$

2 Exponent

Consider a problem of computing exponent of a power series h, $h_{..1} = 0$. Denote $f = e^h$, $r = 1/f$. Recall that $\Delta h = \Delta f/f$.

The next iterative formula [5] (derived as a solution of an equation $\Delta f = gf$ with $g = \Delta h$ in this case) is valid for $m \geq n$:

$$f_{..m+n} = f_{..m} + f_{..n}\, \mathbf{J} \left(x^{m-1} r_{..n} \left\lfloor \Delta(h_{..m+n}) f_{..m}/x^{m-1} \right\rfloor \right) \mod x^{m+n}. \qquad (1)$$

Let $E(n)$ and $I(n)$ denote the complexity of computation e^f and $1/f$ modulo x^n respectively. Then we can use (1) to compute $f_{..m}$ with the complexity

$$E(n) + I(n) + (13 + o(1))F(m) \sim (13/6 + o(1))M(m) \qquad (2)$$

for appropriately chosen parameters m and n, e.g. $n = o(m)$ and $m = O(n\sqrt{\log n})$. (That is one of the ways to obtain factor 13/6 in the complexity estimate for exponent [10].)

To achieve the complexity bound (2) split series into blocks of appropriate size k, e.g. $k \in o(n) \cap \Omega(n/\sqrt{\log n})$. Then use double DFT of order $(2k, k)$ to perform block multiplications in (1).

Double DFT of order (l, k) as a map from $\mathbb{C}[x]$ to $\mathbb{C}^{l+k}$ is defined so that its first l components are the components of DFT of order l, another k components are the components of composition of the variable substitution

$x \to \zeta x$ and DFT of order k, where ζ — an appropriate complex number. Multiple DFT can be defined in similar way [9]. Multiple DFTs are useful to perform multiplications of different sizes on the overlapping sets of inputs. Double DFT of order (l, k) or its inverse costs as much as DFT of order l and DFT of order k plus $O(l + k)$ extra operations, see [10] for details.

In the case $l = 2k$ (as we have) one can use an ordinary DFT of order $3k$ decomposed into outer DFTs of order 3 and inner DFTs of order k instead.

The main term of the complexity estimate (2) is contributed by: $3(m/k)F(k)$ (which we assume to be approximately $3F(m)$) operations to compute DFTs of blocks of f, the same number of operations to compute DFTs of blocks of Δh, the same number of operations to restore blocks of the triple product under the integral, $2F(m)$ operations to compute $2k$-order parts of DFTs of blocks of $\mathbf{J}(\ldots)$ and the same number of operations to restore the product $f_{..n}\,\mathbf{J}(\ldots)$. Other steps (precomputation of $f_{..n}$ and $r_{..n}$, additions, implementation of Δ and $\mathbf{J}$ operators, calculations in the DFT-image spaces) contribute $o(m \log m)$ in total complexity.

To provide a hint for verification we consider a subproblem of the triple product computation (this step seems to be less evident in the algorithm above) in Appendix. Other details (if necessary) see in [10].

Next we turn to introduce an improved version of the algorithm with the use of idea due to D. Harvey [4].

Suppose we are to compute $f_{..2m}$. Firstly we compute $f_{..m}$ acting as mentioned above. By the way we also have DFTs of blocks of $f_{..m-n}$ and $\Delta(h_{..m})$ been computed. At the second stage we use formula [2]

$$f_{..2m} = f_{..m} + f_{..m}(h - \ln f_{..m})_{..2m} \mod x^{2m} \tag{3}$$

derived by the discrete Newton—Raphson method as the solution of an equation $\varphi[f] = h$ with $\varphi[x] = \ln x$ in our case. This stage generally consists of the two essential parts: computation of $\ln f_{..m}$ up to order $2m$ and final multiplication $f_{..m}$ by $h - \ln f_{..m}$.

Denote $s = \Delta(f_{..m})/f_{..m}$. We compute $(\ln f_{..m})_{..2m}$ as $\mathbf{J}s_{..2m-1}$ via the iteration [7]

$$s_{..m'+n-1} = s_{..m'-1} - x^{m'-1} r_{..n} \left\lfloor s_{..m'-1} f_{..m} / x^{m'-1} \right\rfloor \mod x^{m'+n-1}, \tag{4}$$

where $m' \geq m$. We start from $s_{..m-1} = \Delta(h_{..m})$.

To perform the calculations by (4) we use $(6+o(1))F(m)$ extra operations: half of them to compute DFTs of remaining blocks of s, another half to compute blocks of the triple product (see Appendix for some details).

To complete the computation of $f_{..2m}$ we use another $(4 + o(1))F(m)$ operations: half of them to compute $2k$-order DFTs of the order of blocks of

$h - \ln f_{..m}$ and roughly the same number to restore blocks of the product in (3). Recall that $2k$-order DFTs of almost all blocks of $f_{..m}$ are also computed at the first stage of the algorithm since we use double DFTs (this is the only place we gain a benefit from exploiting double DFT).

To summarize, we can compute the exponent up to order $2m$ with the complexity $(23 + o(1))F(m)$.

3 Exponentiation

Consider a problem of raising of a power series h, $h_{..1} = 1$ to a power $C \in \mathbb{C}$. Denote $f = h^C$, $r = 1/f$, $\rho = 1/h$, $s = C\Delta h/h$.

The way of computing a power is just similar with that of computing an exponent. We will give only a sketch.

To compute the first half of the required coefficients of f we use the formula

$$f_{..m+n} = f_{..m} + f_{..n}\,\mathbf{J}\left(x^{m-1} r_{..n} \left\lfloor s_{..m+n-1} f_{..m}/x^{m-1} \right\rfloor\right) \mod x^{m+n} \qquad (5)$$

derived as a solution of the equation $\Delta f = sf$. Next, we switch to the formula

$$f_{..2m} = f_{..m} + f_{..m}(\mathbf{J}s - \ln f_{..m})_{..2m} \mod x^{2m} \qquad (6)$$

derived as a solution of the equation $\ln f = C \ln h$.

To solve a subproblem of computing s we use the iteration [7]

$$s_{..m+n-1} = s_{..m-1} + \rho_{..n}(\Delta(h_{..m+n}) - s_{..m-1}h_{..m+n}) \bmod x^{m+n}. \qquad (7)$$

As above all series are divided into blocks of size k, except that the first half of the series Δh is divided into blocks of size $2k$. We also use double DFTs of order $(2k, k)$.

Suppose we are given $f_{..n}$, $r_{..n}$, $\rho_{..n}$ and $s_{..n-1}$. Then we can compute $f_{..2m}$ using $(40.5 + o(1))F(m)$ operations. This bound is contributed by the following parts:

$(10.5 + o(1))F(m)$ operations to compute $s_{..m-1}$ and DFTs of blocks of $s_{..m-1}$ via (7). We use DFTs of blocks of h, s, Δh, ρ and inverse DFTs to restore triple multiplications (it is essential that the blocks of $\Delta(h_{..m})$ are double-sized);

$(10 + o(1))F(m)$ operations to compute another half of $s_{..2m-1}$. Here each iteration (7) is performed via two ordinary multiplications with the use of DFTs of order $2k$;

$(10 + o(1))F(m)$ operations to compute $f_{..m}$ and DTFs of its blocks by (5). The procedure is the same as in the first part of the exponential algorithm;

$(6+o(1))F(m)$ operations to compute $\ln f_{..m}$ up to order $2m$. This step coincides with that of the exponential algorithm above;

$(4+o(1))F(m)$ operations to perform final multiplication in (6) via DFTs of order $2k$.

Therefore, we have got the required complexity estimate.

Research supported in part by RFBR, grants 11–01–00508, 11–01–00792, and OMN RAS "Algebraic and combinatorial methods of mathematical cybernetics and information systems of new generation" program (project "Problems of optimal synthesis of control systems").

References

[1] Bernstein D. J. *Removing redundancy in high-precision Newton iteration.* Manuscript, 2004. http://cr.yp.to/papers.html#fastnewton.

[2] Brent R. *Multiple-precision zero-finding methods and the complexity of elementary function evaluation.* Analytic computational complexity. Academic Press, NY, 1975, 151–176.

[3] von zur Gathen J., Gerhard J. *Modern computer algebra.* Cambridge University Press, Cambridge, 1999.

[4] Harvey D. *Faster exponentials of power series.* Preprint. 2009. http://arxiv.org/abs/0911.3110.

[5] van der Hoeven J. *Newton's method and FFT trading.* Tech. report 2006–17. Univ. Paris-Sud, Orsay, France, 2006.

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

[7] Karp A. H., Markstein P. *High-precision division and square root.* ACM Trans. Math. Softw. 1997. **23**(4), 561–589.

[8] Sergeev I. S. *Fast algorithms for elementary operations on power series.* Proc. IX Intern. Seminar "Discrete math. and its applications" (Moscow, June 2007). Moscow, MSU Mech. Math. Faculty, 2007, 123–126 (in Russian).

[9] Sergeev I. S. *Regularization of some estimates of complexity of the polynomial multiplication.* Proc. VII Sci. Workshop on Discrete math. and its applications (Moscow, May 2009). Part II. Moscow, Keldysh Inst. of Applied Math., 2009, 26–32 (in Russian).

[10] Sergeev I. S. *Fast algorithms for elementary operations on complex power series.* Discrete Math. and Appl. 2010. **20**(1), 25–60.

Appendix

Let $f, g, h \in \mathbb{C}[[x]]$. Consider a problem of computing

$$q = f\lfloor gh/x^m \rfloor \mod x^n,$$

where m is a multiple of n. Suppose series $f, g, h \in \mathbb{C}[[x]]$ are given divided into blocks of size k (we assume for simplicity that n is a multiple of k):

$$f = \sum_{i\geq 0} a_i x^{ik}, \quad g = \sum_{i\geq 0} b_i x^{ik}, \quad h = \sum_{i\geq 0} c_i x^{ik}, \quad \deg a_i, b_i, c_i < k.$$

Suppose we are also given DFTs of order $3k$ (or, alternatively, double DFTs of order (l_1, l_2) with $l_1 + l_2 = 3k$) of all necessary blocks a_i, b_i, c_i. We are to show how one can compute q via approximately $3(n/k)F(k)$ extra operations.

Flooring makes some complication. We avoid it as following. Let

$$u_i = \sum_{\mu+\nu=m/k+i} b_\mu c_\nu, \qquad \theta = \lfloor u_{-1}/x^k \rfloor.$$

Then

$$\lfloor gh/x^m \rfloor = \theta + \sum_{i\geq 0} u_i x^{ik}.$$

Finally we have

$$q = \sum_{i\geq 0} d_i x^{ik} \mod x^n, \qquad d_i = a_i\theta + \sum_{\lambda+\mu=i} a_\lambda u_\mu.$$

Note that d_i are the polynomials of degree $< 3k$.

Turn to calculations. Let a^* to denote the vector of DFT (or double DFT) of polynomial $a(x)$.

(i) Compute u_i^* for $i = -1, \ldots, n/k - 1$. It costs $O(k)F((m + n)/k) = O((m+n)\log((m+n)/k))$ since u_i^* are the components of the convolution of vectors $(b_0^*, b_1^*, \ldots, b_{(m+n)/k-1}^*)$ and $(c_0^*, c_1^*, \ldots, c_{(m+n)/k-1}^*)$. Recall that b_i^* and c_i^* are the elements of the space $\mathbb{C}^{3k}$ of DFT-images with component-wise multiplication.

(ii) Compute u_{-1} and hence θ via inverse DFT. It costs $3F(k) + O(k)$ as far as u_{-1}^* is known (in fact, $2k$-point inverse DFT suffices here).

(iii) Compute θ^*. it costs $3F(k) + O(k)$.

(*iv*) Compute d_i^* for $i = 0, \ldots, n/k - 1$. It costs $O(k)F(n/k) = O(n \log(n/k))$ contributed by a convolution of order n/k in $\mathbb{C}^{3k}$ and $O(n/k)$ extra additions and multiplications in the same space.

(*v*) Compute all d_i and hence q. It costs $(n/k)(3F(k)+O(k))$ to compute d_i and $O(n)$ to restore q.

Finally we have an upper bound

$$3(n/k+2)F(k) + O((m+n)\log((m+n)/k))$$

for the total complexity.