Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator

Alin Bostan, Pierrick Gaudry, Éric Schost

May 17, 2004

Abstract

We study the complexity of computing one or several terms (not necessarily consecutive) in a recurrence with polynomial coefficients. As applications, we improve the best currently known upper bounds for factoring integers deterministically, and for computing the Cartier-Manin operator for hyperelliptic curves.

1 Introduction

In this paper, we investigate complexity questions related to linear recurrent sequences. Our main focus is on the complexity of computing one or several *selected* terms, not necessarily consecutive, in a recurrence with polynomial coefficients. As applications, we improve the deterministic complexity of factoring integers and of computing the Cartier-Manin operator for hyperelliptic curves.

A well-known particular case of this question is that of linear recurrences with constant coefficients. In this case, the Nth term can be computed with a complexity that is logarithmic in N, by using binary powering techniques. In the general case, however, there is a significant gap, as for the time being no algorithm with a complexity polylogarithmic in N is known.

For the general case, Chudnovsky and Chudnovsky proposed in [11] an algorithm that allows one to compute one term in such a sequence without computing all intermediate ones. This algorithm is closely related to Strassen's algorithm [40] for integer factorization; using baby steps / giant steps techniques, it requires a number of operations which is roughly linear in $\sqrt{N}$ to compute the Nth term in the sequence.

To give a more precise complexity estimate, we let R be a commutative ring with unity, and we introduce a function $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ such that $\mathsf{M}(d)$ denotes the complexity of degree d polynomial multiplication, in terms of number of operations $(+, -, \times)$ in R. We next define a function $\mathsf{MM} : \mathbb{N} \to \mathbb{N}$, such that $\mathsf{MM}(n)$ denotes the complexity of multiplying $n \times n$ matrices over R, in terms of number of operations $(+, -, \times)$ in R; one can for instance take $\mathsf{MM}(n) \in O(n^3)$ (both notions are discussed more precisely below).

Let then M be an $n \times n$ matrix with entries in $R[X]$, all of them of degree at most 1. Given a vector of initial conditions $U_0 \in R^n$, we define a sequence (U_i) of elements in R^n

by the linear recurrence

$$U_{i+1} = M(i+1)U_i, \text{ for all } i \geq 0.$$

Under these assumptions, Chudnovsky and Chudnovsky showed that U_N can be computed using

$$O\big(\mathsf{MM}(n)\mathsf{M}(\sqrt{N}) + n^2\mathsf{M}(\sqrt{N})\log N\big)$$

operations $(+, -, \times)$ in R. Their algorithm relies on manipulations on polynomial matrices: both terms in the complexity estimate describe basic operations on such matrices, respectively multiplication and multi-point evaluation (we give a rough sketch of that algorithm at the end of this section). Using FFT-based multiplication, the above estimate becomes linear in $\sqrt{N}$, up to polylogarithmic factors.

In this article, we present improvements, generalizations and applications of this algorithm. Our contributions are the following:

- For computing the Nth term in the above recurrent sequence, the complexity of our algorithms improves that of [11], but it is still (roughly) linear in $\sqrt{N}$: we prove that the above question can be solved with the better cost of

  $$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\,\mathsf{M}\big(\sqrt{N}\big)\big)$$

  operations $(+, -, \times)$ in R, and 1 inversion. Compared to the result of [11], we save polylogarithmic factors. Chudnovsky and Chudnovsky actually suggested a lower bound of about $\sqrt{N}$ base ring operations for this problem; thus, our improvement gets us closer to the possible optimal bound. Furthermore, in practice, saving such polylogarithmic factors is far from negligible, since in some instances of an application detailed below (Cartier-Manin operator computation), N may be of order 2^{32}.

- We also present a generalization to the computation of *several* selected terms, of indices $N_1 < \cdots < N_r = N$. When the number r of terms to be computed does not exceed $\sqrt{N}$, we show that all of them can be obtained in a time complexity which is the same as above, that is, essentially linear in $\sqrt{N}$, so we are close to the optimal.

- Along the way, we consider a question of basic polynomial arithmetic: given the values taken by a univariate polynomial P on a large enough set of points, how fast can we compute the values taken by P on a shift of this set of points? An obvious solution is to appeal to fast interpolation and evaluation techniques, but we show that shifting evaluation values can be done faster when the evaluation points form an arithmetic progression.

For all the questions above, we will consider polynomial matrices with coefficients of degree at most 1, which is a quite frequent case in applications (for instance, in the two applications presented below). However, this is no real restriction: the case of coefficients of larger degree can be handled *mutatis mutandis*, at the cost of a more involved presentation.

A first application of these techniques is to improve the complexity of the deterministic factorization of integers. To find all the prime factors of an integer N, the original algorithm of Strassen [40] has a complexity of

$$O\big(\mathsf{M}(\sqrt[4]{N})\mathsf{M}_{\mathrm{int}}(\log N)\log N\big)$$

bit operations, where we denote by $\mathsf{M}_{\mathrm{int}}(d)$ the complexity (in terms of bit operations) of multiplying integers of bit-size d.

Now, as mentioned above, the idea of Chudnovsky and Chudnovsky's algorithm generalizes that of Strassen's for integer factorization. Thus, as a consequence, our improvements of their algorithm apply to Strassen's one as well. We prove that there exists a deterministic algorithm that outputs the complete factorization of any integer N using at most

$$O\big(\mathsf{M}(\sqrt[4]{N})\mathsf{M}_{\mathrm{int}}(\log N)\big)$$

bit operations.

In other words, our gain is logarithmic, when expressed in terms of N; to our knowledge, this gives the fastest deterministic integer factorization algorithm. It is customary to express the complexity of factoring algorithms in terms of $\log N$, which represents the bit-size of the input. From this point of view, the complexity estimate of our algorithm is then exponential in $\log N$, and our gain over Strassen's result is polynomial.

Prior to Strassen's algorithm, the fastest deterministic algorithm for integer factorization was Pollard's algorithm [30]. For any positive $\delta > 0$, the complexity of this method is in

$$O\big(\mathsf{M}(\sqrt[4]{N})N^{\delta}\mathsf{M}_{\mathrm{int}}(\log N)\big)$$

bit operations (see Section 5 for more details). Other deterministic factorization algorithms exist, see for instance [31, 26]. Some of them have a better conjectured complexity, whose validity relies on (yet) unproved number-theoretic conjectures.

For completeness, we mention that the fastest probabilistic algorithm for integer factorization, with fully established complexity bound, is due to Lenstra and Pomerance [23], with a bit complexity that is polynomial in $\exp(\sqrt{\log N \log\log N})$. The number field sieve [22] has a better conjectured complexity, which is expected to be polynomial in $\exp(\sqrt[3]{\log N(\log\log N)^2})$.

In accordance with these estimates, the latter algorithms are indeed better suited for practical computations than our deterministic variant, and all most recent record-sized computations rely on the number field sieve. However, as already pointed out by Pollard [30], proving unconditional, deterministic upper bounds for this question remains an important challenge.

Another motivation of our work is an application to point-counting procedures in hyperelliptic cryptography, related to the computation of the Cartier-Manin operator [10, 24] of curves over finite fields.

The basic ideas already appear for elliptic curves point counting; see [37, Chapter V]. Suppose we are to count the number n of solutions of the equation $y^2 = f(x)$ over the field $\mathbb{F}_p$, where p is a prime (larger than 2) and f has degree 3. Let $\chi : \mathbb{F}_p^* \to \{-1, 1\} \subset \mathbb{F}_p^*$ be the character $x \mapsto x^{(p-1)/2}$, extended to $\mathbb{F}_p$ by $\chi(0) = 0$. Since $\chi(x) = 1$ exactly when x is a non-zero square, it is easy to see that n coincides with $\sum_{x\in\mathbb{F}_p} \chi(f(x))$ modulo p.

Now, for $i \neq 0$, $\sum_{x \in \mathbb{F}_p} x^i = -1$ if $p-1$ divides i, and 0 otherwise; this implies that n modulo p coincides with the opposite of the coefficient of x^{p-1} in $f(x)^{(p-1)/2}$.

The generalization of these ideas to hyperelliptic curves leads to the notions of Hasse-Witt matrix and Cartier-Manin operator. As in the previous example, using this time a result of Manin [24], the Hasse-Witt matrix can be used as part of a point-counting procedure. Furthermore, in the case of a hyperelliptic curve given by the equation $y^2 = f(x)$, the entries of this matrix are suitable coefficients of the polynomial $h = f^{(p-1)/2}$.

It is easy to see that the coefficients of h satisfy a linear recurrence with rational function coefficients. Using our results on linear recurrences, we deduce an algorithm to compute the Hasse-Witt matrix whose complexity is essentially linear in $\sqrt{p}$. In fixed genus, for a curve defined over the finite field $\mathbb{F}_{p^d}$, the complexity of our algorithm is

$$O\big((\mathsf{M}(\sqrt{p}) + d \log p)\, \mathsf{M}(d) \mathsf{M}_{\mathrm{int}}(\log p) \big)$$

bit operations. This improves the methods used in [14] and [25] which have a complexity essentially linear in p. Note that when the characteristic p of the base field is small enough, other methods, such as the p-adic methods used in Kedlaya's algorithm [19], also provide very efficient point-counting procedures, but their complexity is actually at least linear in p, see [13].

As an illustration, in [6], we used our method to find the group order of the Jacobian of a hyperelliptic curve whose size would not have been possible to handle with previously known algorithms.

Main algorithmic ideas. To shed some light on this paper's structure, let us briefly recall Strassen's algorithm for integer factorization; we will then discuss Chudnovsky and Chudnovsky's generalization, and mention where our improvements fit in.

Suppose we are to factor an integer M. Factoring M by trial division with all integers smaller than $\sqrt{M}$ has a cost linear in $\sqrt{M}$. To speed up this computation, Strassen proposed to gather all integers smaller than $\sqrt{M}$ in $\sqrt[4]{M}$ blocks, each of them containing $\sqrt[4]{M}$ consecutive integers. Let thus c be of order $\sqrt[4]{M}$ and write $f_0 = 1 \cdots c \mod M$, $f_1 = (c+1) \cdots (2c) \mod M$, $\ldots$, $f_{c-1} = (c^2 - c + 1) \cdots (c^2) \mod M$. If the values $f_0, \ldots, f_{c-1}$ can be computed efficiently, then finding a prime factor of M becomes easy, using the gcd's of $f_0, \ldots, f_{c-1}$ with M. Thus, the main difficulty lies in computing the values above, whose cost will actually dominate the whole complexity.

To do so, let $R = \mathbb{Z}/M\mathbb{Z}$ and let F be the polynomial $(X+1) \cdots (X+c) \in R[X]$. The "baby steps" part of the algorithm consists in computing the polynomial F: using the subproduct tree algorithm [42, Chapter 10], this is done in $O(\mathsf{M}(c) \log c)$ operations in R — a faster algorithm that uses the special structure of F is proposed in [11], with a complexity in $O(\mathsf{M}(c))$. Then, the "giant steps" part of the algorithm consists in evaluating the polynomial F at $0, c, \ldots, (c-1)c$, since $F(ic)$ is the value f_i mentioned above. Using fast evaluation techniques, these values can be computed in $O(\mathsf{M}(c) \log c)$ operations in R. Since c has order $\sqrt[4]{M}$, the whole process has complexity of $O(\mathsf{M}(\sqrt[4]{M}) \log M)$ operations modulo M. This forms the core of Strassen's algorithm for factorization.

Independently of the factorization question, one sees that multiplying the values $f_0, f_1, \ldots, f_{c-1}$ yields the product $1 \cdots c^2$ modulo M. Thus, this algorithm can be used to compute factorials: the analysis above shows that in any ring R, for any $N \geq 0$, the

product $1 \cdots N$ can be computed using $O(\mathsf{M}(\sqrt{N}) \log N)$ operations in R (the case when N is not a perfect square is easily dealt with). Note the improvement obtained over the naive iterative algorithm, whose complexity is linear in N.

Now, the sequence $U_N = N!$ is a basic example of a solution of a linear recurrence with polynomial coefficients, since the values U_N satisfy the recurrence $U_N = NU_{N-1}$. Chudnovsky and Chudnovsky then generalized the above baby steps / giant steps process to compute the Nth term of an $n \times n$ matrix recurrence with polynomial coefficients (for simplicity, we discuss the case where all degrees are at most 1). The main computational tasks are the same as above: computing the matrix equivalent of the above polynomial F has cost $O(\mathsf{MM}(n)\mathsf{M}(\sqrt{N}))$; the subsequent multipoint evaluation has cost $O(n^2\mathsf{M}(\sqrt{N}) \log N)$. This gives the complexity estimate mentioned in the previous paragraphs.

Let us now describe our improvement, say for the computation of the factorial (the matrix case is handled similarly). Our first remark is that we are not interested in the expression of the polynomial F on the monomial basis, but merely in its values on suitable families of points. The second remark is that both F and the evaluation points have very special structures: on the one hand, F is the product of the linear polynomials $(X + 1)$, $(X + 2)$, $\ldots$, $(X + c)$; on the other hand, the evaluation points form the arithmetic progression $0, c, \ldots, (c - 1)c$. Our main objective is to use this additional structure to reduce the cost of evaluating F from $O(\mathsf{M}(c) \log c)$ to $O(\mathsf{M}(c))$.

At the moment, we do not know how to evaluate an arbitrary polynomial on the sequence $0, c, \ldots, (c-1)c$ in time $O(\mathsf{M}(c))$. However, this will be possible in our case due to the special shape of the polynomial F. We will use a divide-and-conquer approach, whose inductive step will consist in evaluating a polynomial akin to F, but with degree halved, on an arithmetic progression of halved size. Putting this idea to practice will involve the following operation, which is the basic building block of all algorithms in this paper: given the values of a polynomial P on an arithmetic progression, compute the values of P on a shift of this arithmetic progression.

In the general case of an $n \times n$ matrix recurrence, our fast solution to this question will enable us to dispense completely with polynomial matrix multiplications, and to reduce by a logarithmic factor all costs related to multipoint evaluation. However, it will impose suitable invertibility conditions in R; we will pay a special attention to such conditions, since in our two applications (integer factorization and Cartier-Manin operator computation) the base ring will contain zero-divisors.

Organization of the paper. After recalling useful notation and previous results in Section 2, we start in Section 3 with our algorithm for shifting a polynomial given by its values on special evaluation points. This building block is used in Section 4 to give an algorithm for evaluating some particular polynomial matrices at an arithmetic progression. As an application, we present our algorithm for integer factorization in Section 5.

Next, in Section 6 we describe our improvement on Chudnovsky and Chudnovsky's algorithm for computing one term in a recurrence with polynomial coefficients; a generalization to compute several terms is given in Section 7. In Section 8, we apply these results to the computation of the Cartier-Manin operator of a hyperelliptic curve.

A preliminary version of this paper was published in the proceedings of the Fq'7

conference [6]. With the exception of Theorem 1 below, all results in the present version are new, or improve those given in [6].

2 Notation and basic results

In this paper, we use two distinct computational models. Our algorithms for linear recurrent sequences apply over arbitrary rings, so their complexity will be expressed in an algebraic model, counting at unit cost the basic ring operations. On the other hand, our algorithms for integer factorization and Cartier-Manin operator computation naturally require that we count the number of bit-operations: for this purpose, we will use a standard Turing machine model.

Our algorithms use baby steps / giant steps techniques. As usual with such algorithms, the memory requirements essentially follow the time complexities (by opposition, note that naive iterative algorithms for solving linear recurrences run in constant memory). We will thus give memory estimates for all our algorithms.

In what follows, $\log x$ will always denote the logarithm of x in base 2; $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the largest integer less than or equal to x, and the smallest integer larger than or equal to x.

Algebraic complexity model. Let R be a commutative ring; we suppose that R has a unit element being denoted by 1_R, or simply 1. Let φ be the map $\mathbb{N} \to R$ sending n to $n \cdot 1_R = 1_R + \cdots + 1_R$ (n times); the map φ extends to a map $\mathbb{Z} \to R$. We will simply denote by $n \in R$ the image $\varphi(n)$.

The algorithms of Sections 3, 4, 6 and 7 apply over an arbitrary commutative ring with unity. To give complexity estimates for such algorithms, we use the straight-line program computational model, counting at unit cost the operations in R, see [8]. In particular, branchings are not present in this model. On the other hand, exact divisions are allowed. Divisions will however receive a particular treatment, since for many practical situations they are more costly than additions and multiplications. We will then indicate the number of divisions in a separate count, and use the term "ring operations" to denote the operations $(+, -, \times)$.

From a formal point of view, to assign a notion of space complexity to a straight-line program, we play a pebble game on the underlying directed acyclic graph. We refer to [4] for a detailed description. However, we shall not make an explicit use of such a detailed presentation: we will informally speak of the number of ring elements that have to be stored, or more simply of "space requirements"; such quantities actually correspond to the number of pebbles in the underlying pebble game.

Finally, we will use the following lemma, which shows how to trade inversions for multiplications. This result is due to Montgomery [28], see also [29, p. 66].

Lemma 1 *Let R be a commutative ring with unity and let $r_1, \ldots, r_n$ be units in R. Then one can compute $r_1^{-1}, \ldots, r_n^{-1}$ for $3n - 3 \in O(n)$ multiplications and 1 inversion in R, and using $O(n)$ space.*

Proof. We first compute the products $R_1 = r_1$, $R_2 = r_1 r_2$, ..., $R_n = r_1 r_2 \cdots r_n$; this requires $n - 1$ multiplications. Then we invert R_n, and by another $n - 1$ multiplications

we deduce $S_n = R_n^{-1}$, $S_{n-1} = r_n S_n$, ..., $S_1 = r_2 S_2$, so that S_i is the inverse of $r_1 \cdots r_i$ for all i. Thus, we deduce the inverse s_i of r_i by computing $s_1 = S_1$, $s_2 = R_1 S_2$, ..., $s_n = R_{n-1} S_n$ for $n-1$ additional operations. □

Algorithms for polynomials. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function that represents the complexity of polynomial multiplication, *i.e.* such that over any ring R, the product of two degree d polynomials can be computed within $\mathsf{M}(d)$ base ring operations. Using the algorithms of [35, 33, 9], $\mathsf{M}(d)$ can be taken in $O(d \log d \log\log d)$.

We suppose that the function M is super-additive, that is $\mathsf{M}(d_1) + \mathsf{M}(d_2) \leq \mathsf{M}(d_1 + d_2)$ for all integers d_1 and d_2. In particular, the inequality $\mathsf{M}(d) \leq \frac{1}{2}\mathsf{M}(2d)$ holds for all $d \geq 1$, and $\sum_i n_i \mathsf{M}(d_i) \leq \sum_i \mathsf{M}(n_i d_i)$ holds for all $n_i, d_i \geq 1$.

We assume that the product of two degree d polynomials can be computed in space $O(d)$; this is the case for all classical algorithms, such as naive, Karatsuba and Schönhage-Strassen multiplications.

We use the following classical results on polynomial arithmetic over R. The earliest references we are aware of are [17, 27, 39, 5], see [42] for a detailed account and [7] for more recent algorithms.

Multipoint evaluation. If P is a polynomial of degree d in $R[X]$ and $r_0, \ldots, r_d$ are points in R, then the values $P(r_0), \ldots, P(r_d)$ can be computed using $O(\mathsf{M}(d) \log d)$ operations in R and $O(d \log d)$ space.

Interpolation. In the converse direction, if $P(r_0), \ldots, P(r_d)$ are known and $r_i - r_j$ is a unit for all $i \neq j$, the coefficients of P can be recovered using $O(\mathsf{M}(d) \log d)$ base ring operations, 1 inversion, and $O(d \log d)$ space (the algorithms above use $O(d)$ inversions, but these inversions can be replaced by $O(d)$ multiplications and 1 inversion using Lemma 1).

Matrix multiplication. We denote by $\mathsf{MM} : \mathbb{N} \to \mathbb{N}$ a function that represents the complexity of matrix multiplication, *i.e.* such that over any ring R, the product of two $n \times n$ matrices with entries in R can be computed within $\mathsf{MM}(n)$ base ring operations. Thus, $\mathsf{MM}(n) \in O(n^3)$ for the classical matrix multiplication. Using Strassen's algorithm [38], we can take $\mathsf{MM}(n) \in O(n^{\log 7}) \subset O(n^{2.81})$. We assume that the product of two $n \times n$ matrices can be computed in space $O(n^2)$, which is the case for classical as well as Strassen's multiplications.

Bit-wise model. In Sections 5 and 8, we discuss the complexity of factoring integers, and of computing the Cartier-Manin operator on curves over finite fields. For these applications, the proper complexity measure is the bit (or boolean) complexity. For this purpose, our complexity model will be the multi-tape Turing machine, see for instance [34]. We will speak of *bit operations* to estimate time complexities in this model; storage requirements will be expressed in bits as well.

We will in particular have to apply the algorithms of Sections 4 and 7 over a ring R such as $\mathbb{Z}/N\mathbb{Z}$ or a p-adic ring. The bit complexity will then be estimated by multiplying the cost estimates of Sections 4 and 7 by the bit complexity of the basic operations in R. To do so, we will use the notation $\mathsf{M}_{\text{int}} : \mathbb{N} \to \mathbb{N}$ to denote the bit complexity of integer

multiplication, *i.e.* such that the product of two integers of bit-size d can be computed within $\mathsf{M}_{\text{int}}(d)$ bit-operations. Using the algorithm of Schönhage-Strassen [35], $\mathsf{M}_{\text{int}}(d)$ can be taken in $O(d \log d \log\log d)$; we suppose that the space complexity of size d integer multiplication is in $O(d)$.

To formulate it alternately, we say that for any $N \in \mathbb{N}$, multiplying two integers bounded by N takes $\mathsf{M}_{\text{int}}(\log N)$ bit operations. Next, for $N \in \mathbb{N}$, the bit complexities of the operations $(+, -)$ and $\times$ in the ring $R = \mathbb{Z}/N\mathbb{Z}$ are respectively in $O(\log N)$ and $O(\mathsf{M}_{\text{int}}(\log N))$, see [42]; we will however simply use the upper bound $O(\mathsf{M}_{\text{int}}(\log N))$ for all three operations. The space complexities are in $O(\log N)$.

Similarly, still writing $R = \mathbb{Z}/N\mathbb{Z}$, if P is a monic degree d polynomial in $R[X]$, the bit complexity of the operations $(+, -, \times)$ in $R[X]/P$ are in $O(\mathsf{M}(d)\mathsf{M}_{\text{int}}(\log N))$, and the space complexity is in $O(d \log N)$.

Finally, we will use the fact that the extended gcd of two integers bounded by N can be computed for $O(\mathsf{M}_{\text{int}}(\log N) \log\log N)$ bit operations using the algorithms of Knuth and Schönhage [21, 32], with $O(\log N)$ storage.

3 Shifting evaluation values

In this section, we address a particular case of the question of *shifting evaluation values* of a polynomial. The question reads as follows: Let P be a polynomial of degree d in $R[X]$, where R is a commutative unitary ring. Let a and $r_0, \ldots, r_d$ be in R. Given $P(r_0), \ldots, P(r_d)$, how fast can we compute $P(r_0 + a), \ldots, P(r_d + a)$? We stress the fact that the coefficients of P on the monomial basis are *not* part of the input.

Suppose that all differences $r_i - r_j$, $i \neq j$ are units in R. Then using fast interpolation and fast multipoint evaluation, the problem can be answered within $O(\mathsf{M}(d) \log d)$ operations in R. We now propose an improved solution, in the special case when $r_0, \ldots, r_d$ form an arithmetic progression; its complexity is in $O(\mathsf{M}(d))$, so we gain a logarithmic factor.

Our algorithm imposes some invertibility conditions on the sample points, slightly more general than those mentioned above. See the remarks after the proof for an alternative solution that requires no invertibility condition at all, but works only in a special case; see also these remarks for mentions on how to lower the constant factor that appears in our complexity estimate.

Theorem 1 *Let R be a commutative ring with unity, $r, \beta \in R$, $d \in \mathbb{N}$ and let $\mathfrak{P}$ be in $R[X]$ of degree less than or equal to d, such that the sequence $\mathfrak{P}(r), \mathfrak{P}(r+\beta), \ldots, \mathfrak{P}(r+d\beta)$ is known.*

Let α in R, such that α is divisible by β, and suppose that $1, \ldots, d$ and $\alpha/\beta - d, \ldots, \alpha/\beta + d$ are units. Then the sequence $\mathfrak{P}(r+\alpha), \mathfrak{P}(r+\alpha+\beta), \ldots, \mathfrak{P}(r+\alpha+d\beta)$ can be computed using $2\mathsf{M}(d) + O(d)$ ring operations in R, 1 inversion, and space $O(d)$.

In the following section, we need a more explicit version of the complexity estimate; we will then let C be a constant such that the complexity of the algorithm above is bounded from above by $2\mathsf{M}(d) + \mathsf{C}d$.

PROOF. Our algorithm reduces to the multiplication of two suitable polynomials of degree at most d and $2d$; $O(d)$ additional operations come from additional pre- and post-processing operations. All operations made below on integer values actually take place in R.

First, we perform a change of variables, so as to normalize the situation. Let us define the polynomial $P(X) = \mathfrak{P}(\beta X + r)$. Then our assumption is that the values $P(0), P(1), \ldots, P(d)$ are known. Let us write $a = \alpha/\beta$; our objective is then to determine the values $P(a), \ldots, P(a+d)$. To this effect, our assumption on R enables us to write the Lagrange interpolation formula:

$$P = \sum_{i=0}^{d} P(i) \frac{\prod_{j=0, j\neq i}^{d} (X - j)}{\prod_{j=0, j\neq i}^{d} (i - j)}.$$

From now on, we denote by $\delta(i, d)$ the denominator $\prod_{j=0, j\neq i}^{d} (i - j)$ and by $\widetilde{P}_i$ the ratio $P(i)/\delta(i, d)$. Accordingly, we rewrite the formula above as

$$P = \sum_{i=0}^{d} \widetilde{P}_i \prod_{j=0, j\neq i}^{d} (X - j).$$

For k in $0, \ldots, d$, let us evaluate P at $a + k$:

$$P(a+k) = \sum_{i=0}^{d} \widetilde{P}_i \prod_{j=0, j\neq i}^{d} (a + k - j).$$

Using our assumption on a, we can complete each product by the missing factor $a+k-i$:

$$P(a+k) = \sum_{i=0}^{d} \widetilde{P}_i \frac{\prod_{j=0}^{d} (a+k-j)}{a+k-i} = \left(\prod_{j=0}^{d} (a+k-j) \right) \cdot \left(\sum_{i=0}^{d} \widetilde{P}_i \frac{1}{a+k-i} \right). \qquad (1)$$

Just as we introduced the sequence $\delta(i, d)$, we now introduce the sequence $\Delta(a, k, d)$ defined by $\Delta(a, k, d) = \prod_{j=0}^{d} (a+k-j)$ and define $Q_k = P(a+k)/\Delta(a, k, d)$. Using these values, Equation (1) reads

$$Q_k = \sum_{i=0}^{d} \widetilde{P}_i \frac{1}{a+k-i}. \qquad (2)$$

Let $\widetilde{P}$ and S be the polynomials:

$$\widetilde{P} = \sum_{i=0}^{d} \widetilde{P}_i X^i, \quad S = \sum_{i=0}^{2d} \frac{1}{a+i-d} X^i;$$

then by Equation (2), for $k = 0, \ldots, d$, Q_k is the coefficient of degree $k+d$ in the product $\widetilde{P}S$. From the knowledge of Q_k, it is clear how to recover $P(a+k)$.

Let us analyze the complexity of the underlying computations. First note the inductive formulas giving the values $\delta(i, d)$ and $\Delta(a, i, d)$, $i = 0, \ldots, d$:

$$\delta(0, d) = \prod_{j=1}^{d} (-j), \quad \delta(i, d) = \frac{i}{i-d-1} \delta(i-1, d),$$

$$\Delta(a,0,d)=\prod_{j=0}^{d}(a-j),\quad \Delta(a,k,d)=\frac{a+k}{a+k-d-1}\Delta(a,k-1,d).$$

Using Lemma 1, we deduce that all values $\Delta(a,i,d)$, the inverses of all $\delta(i,d)$ and all coefficients of S can be computed in $O(d)$ ring operations and 1 inversion. The sequence $\widetilde{P}_i, i=0,\dots,d$, can then be deduced for $O(d)$ ring operations from the input sequence $P(i)$.

The coefficients $Q_0,\dots,Q_d$ can be obtained by polynomial multiplication. Performing the product $\widetilde{P}S$ has a time complexity $2\mathsf{M}(d)$, and requires a space of $O(d)$. From the knowledge of $Q_0,\dots,Q_d$, we deduce $P(a),\dots,P(a+d)$ by multiplications with the coefficients $\Delta(a,i,d)$; this requires $O(d)$ ring operations. □

Remarks.

- The reader may note the similarity of our problem with the question of computing the Taylor expansion of a given polynomial P at a given point in R. The algorithm of [1] solves this question with a complexity of $\mathsf{M}(d)+O(d)$ operations in R and space $O(d)$. The complexity results are thus quite similar; it turns out that analogous generating series techniques are used in that algorithm.

- In [15], the operation called *middle product* is defined: Given a ring R, and A, B in $R[X]$ of respective degrees d and $2d$, write $AB=C_0+C_1X^{d+1}+C_2X^{2d+2}$, with all C_i of degree at most d; then the middle product of A and B is the polynomial C_1. This is precisely what is needed in the algorithm above.

  Up to considering the reciprocal polynomial of A, the middle product by A can be seen as the transpose of the map of multiplication by A. General program transformation techniques then show that it can be computed in time $\mathsf{M}(d)+O(d)$ (but with a possible loss in space complexity): this is the *transposition principle* for linear algorithms, which is an analogue of results initiated by Tellegen [41] and Bordewijk [3] in circuit theory. We refer to [18, Problem 6] for a longer discussion and [8, Theorem 13.20] for a proof; see also [15] for the independent discovery of the middle product, and [7] for additional applications.

  Thus, the time complexity of the algorithm above can be reduced to $\mathsf{M}(d)+O(d)$ ring operations, but possibly with an increased space complexity.

- Using the notation of the proof above, we mention an alternative $O(\mathsf{M}(d))$ algorithm which does *not* require any invertibility hypotheses, in the special case when $a=d+1$. The key fact is that for any degree d polynomial P, the sequence $P(0),P(1),\dots$ is linearly recurrent, of characteristic polynomial $Q(X)=(1-X)^{d+1}$. Thus, if the first terms $P(0),\dots,P(d)$ are known, the next $d+1$ terms $P(d+1),\dots,P(2d+1)$ can be recovered in $O(\mathsf{M}(d))$ using the algorithm in [36, Theorem 3.1].

4 Algorithms for polynomial matrix evaluation

Referring to the sketch of Strassen's algorithm given in the introduction, we see that an important part of the computational effort lies in evaluating polynomials on points

that form an arithmetic progression. A generalization of this question appears in Chudnovsky and Chudnovsky's algorithm for matrix recurrences, where one has to evaluate a polynomial matrix at points in an arithmetic progression.

We now present such an evaluation algorithm, in the special case when the polynomial matrix has the form

$$M_k(X) = M(X + \alpha k) \cdots M(X + \alpha),$$

where M is a given polynomial matrix with entries of degree at most 1: this is enough to handle both Strassen's and Chudnovsky and Chudnovsky's algorithms.

Using the result of the previous section, we propose a divide-and-conquer approach, which saves a logarithmic factor compared to classical multipoint evaluation techniques. To denote the successive steps of this process, we introduce a suitable notation. With a positive integer k, we associate $\mathsf{Ind}(k) = \lfloor \log k \rfloor$ and the sequence defined by $k_0 = k$ and $k_{i+1} = \lfloor k_i/2 \rfloor$, so that $k_{\mathsf{Ind}(k)} = 1$.

Theorem 2 *Let R be a ring and let M be an $n \times n$ polynomial matrix over R with entries of degree at most 1. Let α and β be in R, such that β divides α. For k in $\mathbb{N}$, let M_k be the polynomial matrix*

$$M_k(X) = M(X + \alpha k) \cdots M(X + \alpha)$$

and let $\mathsf{Ind}(k)$ and $k_0, \ldots, k_{\mathsf{Ind}(k)}$ be defined as above. Suppose that $1, \ldots, 2k_1 + 1$ and, for $i = 1, \ldots, \mathsf{Ind}(k)$,

$$k_i\frac{\alpha}{\beta} - k_i, \quad k_i\frac{\alpha}{\beta} - k_i + 1, \quad \ldots, \quad k_i\frac{\alpha}{\beta} + k_i$$

are units in R. Then the scalar matrices $M_k(0), M_k(\beta), \ldots, M_k(k\beta)$ can be computed in $O(\mathsf{MM}(n)k + n^2\mathsf{M}(k))$ ring operations and 1 inversion in R. The space requirement is $O(n^2k)$ elements in R.

Proof. We first deal with inversions: using Lemma 1, we see that all elements $1, \ldots, 2k_1 + 1$ and, for $i = 1, \ldots, \mathsf{Ind}(k)$,

$$k_i\frac{\alpha}{\beta} - k_i, \quad k_i\frac{\alpha}{\beta} - k_i + 1, \quad \ldots, \quad k_i\frac{\alpha}{\beta} + k_i$$

can be inverted using $O(k)$ products and one inversion in R. So for what follows, we assume that all these inverses are available.

We will give an estimate on the complexity of computing the values of M_{k_i} on $0, \beta, \ldots, k_i\beta$, for decreasing values of i. The case $i = \mathsf{Ind}(k)$ is obvious, since then $k_i = 1$ and $M_{k_i} = M(X + \alpha)$, which we can be evaluated on the points 0 and β in $O(n^2)$ operations.

Then, for some $i = \mathsf{Ind}(k), \ldots, 1$, suppose that the values of M_{k_i} are known on $0, \beta, \ldots, k_i\beta$. We now show how to deduce the values of $M_{k_{i-1}}$ on $0, \beta, \ldots, k_{i-1}\beta$. To this effect, we will use Theorem 1, using the fact that all entries of M_{k_i} have degree at most k_i.

To keep control on the $O(\ )$ constants used at each induction level, we make them explicit using the constant C introduced in the previous section.

- The elements $1, \ldots, 2k_i + 1$ are supposed to be units in R. Applying Theorem 1 to each entry of M_{k_i} to perform a shift by $(k_i + 1)\beta$, the values of M_{k_i} on $(k_i + 1)\beta, \ldots, (2k_i + 1)\beta$ can be computed for $n^2(2\mathsf{M}(k_i) + \mathsf{C}k_i)$ ring operations, since all required inverses are known. We then have at our disposal the values of M_{k_i} at $0, \beta, \ldots, (2k_i + 1)\beta$.

- Furthermore, $k_i\frac{\alpha}{\beta} - k_i, k_i\frac{\alpha}{\beta} - k_i + 1, \ldots, k_i\frac{\alpha}{\beta} + k_i$ are units in R. Applying Theorem 1 to each entry of M_{k_i} to perform shifts by $k_i\alpha$ and then by $(k_i + 1)\beta$, the values of $M_{k_i}(X + k_i\alpha)$ on $0, \beta, \ldots, (2k_i + 1)\beta$ can be computed for $2n^2(2\mathsf{M}(k_i) + \mathsf{C}k_i)$ ring operations.

From these values, it is easy to deduce the values of $M_{k_{i-1}}$ at $0, \beta, \ldots, k_{i-1}\beta$. We distinguish two cases, according to the parity of k_{i-1}.

- Suppose first that k_{i-1} is even, so that $k_{i-1} = 2k_i$. Using the equality

$$M_{k_{i-1}}(X) = M_{k_i}(X + \alpha k_i) \cdot M_{k_i}(X),$$

  we obtain the values of $M_{k_{i-1}}$ at $0, \beta, \ldots, k_{i-1}\beta$ by multiplying the (already computed) values of $M_{k_i}(X + \alpha k_i)$ and $M_{k_i}(X)$ at $0, \beta, \ldots, k_{i-1}\beta$. This requires $(k_{i-1} + 1)\mathsf{MM}(n)$ ring operations.

- Suppose now that k_{i-1} is odd, so that $k_{i-1} = 2k_i + 1$. Using the equality

$$M_{k_{i-1}}(X) = M(X + \alpha k_{i-1}) \cdot M_{k_i}(X + \alpha k_i) \cdot M_{k_i}(X),$$

  we obtain the values of $M_{k_{i-1}}$ at $0, \beta, \ldots, k_{i-1}\beta$ as follows. We first multiply the values of $M_{k_i}(X + \alpha k_i)$ and $M_{k_i}(X)$ at $0, \beta, \ldots, k_{i-1}\beta$, for $(k_{i-1} + 1)\mathsf{MM}(n)$ ring operations. Then we evaluate $M(X + \alpha k_{i-1})$ at $0, \beta, \ldots, k_{i-1}\beta$ for $2n^2(k_{i-1} + 1)$ operations, from which we finally deduce the values of M_{2k_i+1} at $0, \beta, \ldots, k_{i-1}\beta$ for $(k_{i-1} + 1)\mathsf{MM}(n)$ ring operations.

Let us denote by $T(k_{i-1})$ the cost of these operations. From the analysis above, we deduce the inequality

$$T(k_{i-1}) \leq T(k_i) + 2(k_{i-1} + 1)\mathsf{MM}(n) + 6n^2\mathsf{M}(k_i) + n^2\mathsf{C}'k_i,$$

for a constant C' that can be taken to be $\mathsf{C}' = 3\mathsf{C} + 4$. Using the definition of the sequence k_i and our assumptions on the function M, we deduce the estimate $T(k) = T(k_0) \in O(\mathsf{MM}(n)k + n^2\mathsf{M}(k))$. The space complexity estimate is easily dealt with, since all computations at step i can be done in space $O(n^2k_i)$. □

The case $\alpha = \beta = 1$ is not covered in this last theorem. However, this case is actually easier to handle, as we now demonstrate (note also that there are no invertibility conditions).

Proposition 1 *Let R be a ring and let M be an $n \times n$ polynomial matrix over R with entries of degree at most 1. For k in $\mathbb{N}$, let M_k be the polynomial matrix*

$$M_k(X) = M(X + k) \cdots M(X + 1).$$

Then the scalar matrices $M_k(0), M_k(1), \ldots, M_k(k)$ can be computed within $O(\mathsf{MM}(n)k)$ ring operations. The space requirement is $O(n^2k)$ elements in R.

PROOF. We first evaluate all matrices $M(1), \ldots, M(2k)$; this requires $O(n^2k)$ operations, and takes space $O(n^2k)$. The conclusion is now similar to the proof of Lemma 1. Denoting by I the $n \times n$ identity matrix, we first compute the products

$$R_k = I, \quad R_{k-1} = R_k M(k), \quad R_{k-2} = R_{k-1} M(k-1), \quad \ldots \quad R_0 = R_1 M(1)$$

and

$$L_0 = I, \quad L_1 = M(k+1)L_0, \quad L_2 = M(k+2)L_1, \quad \ldots \quad L_k = M(2k)L_{k-1}.$$

This takes $O(\mathsf{MM}(n)k)$ ring operations, and $O(n^2k)$ space. We conclude by computing the matrices $M_k(i) = L_i R_i$, for $0 \leq i \leq k$. This also takes $O(\mathsf{MM}(n)k)$ ring operations, and $O(n^2k)$ space. □

In Section 7, we have to deal with a similar evaluation problem, but with arbitrary sample points. The following corollary of Proposition 1 will then be useful; compared to the particular case of Theorem 2, we lose a logarithmic factor in the evaluation process.

Corollary 1 *Let R be a ring and let M be an $n \times n$ polynomial matrix over R with entries of degree at most 1. For k in $\mathbb{N}$, let M_k be the polynomial matrix*

$$M_k(X) = M(X+k) \cdots M(X+1).$$

Assume that $1, \ldots, k$ are units in R. Then for any set of $k+1$ elements β_i of R the scalar matrices $M_k(\beta_0), M_k(\beta_1), \ldots, M_k(\beta_k)$ can be computed in

$$O(\mathsf{MM}(n)k + n^2 \mathsf{M}(k) \log k)$$

ring operations and 1 inversion in R. The space requirement is $O(kn^2 + k \log k)$.

PROOF. We start by evaluating M_k on $0, \ldots, k$: by the previous proposition, this requires $O(\mathsf{MM}(n)k)$ ring operations and space $O(n^2k)$. Using our assumption on R, from these values, it is possible to interpolate each entry of $M_k(X)$ within $O(n^2 \mathsf{M}(k) \log k)$ ring operations and 1 inversion in R, and using space $O(k \log k)$. Then, we can evaluate all entries of this matrix at the points $\beta_0, \beta_1, \ldots, \beta_k$, with the same cost. All inversions can be grouped together using Lemma 1. □

5 Application to integer factorization

In this section, we apply the algorithm of Theorem 2 to reduce by a logarithmic factor the best known (proved) upper bound for deterministic integer factorization, which is due to Strassen [40]. Recall Strassen's result: any positive integer N can be completely factored using at most

$$O\big(\mathsf{M}(\sqrt[4]{N})\mathsf{M}_{\mathrm{int}}(\log N) \log N\big)$$

bit operations, see for instance [42, Corollary 19.4]. Prior to Strassen, Pollard gave a more complicated deterministic algorithm [30] and proved the following complexity result: for any positive $\delta > 0$, the integer N can be completely factored within

$$O\big(\mathsf{M}(\sqrt[4]{N})N^{\delta}\mathsf{M}_{\mathrm{int}}(\log N)\big)$$

bit operations (see below). The following theorem improves these complexity results.

Theorem 3 *There exists a deterministic algorithm that outputs the complete factorization of any positive integer N using at most*

$$O\big(\mathsf{M}(\sqrt[4]{N})\mathsf{M}_{\mathrm{int}}(\log N)\big)$$

bit operations. The space complexity is $O(\sqrt[4]{N}\log N)$ bits.

Our proof closely follows that of Strassen, in the presentation of [42], the main ingredient being now Theorem 2; some additional complications arise due to the non-trivial invertibility conditions required by that theorem. We start by the following lemma.

Lemma 2 *Let $F_1, \ldots, F_k$ be non-zero elements in $\mathbb{Z}/N\mathbb{Z}$. Then one can decide whether all F_i are invertible modulo N and, if not, find one F_i such that $\gcd(F_i, N) > 1$, within*

$$O(\mathsf{M}_{\mathrm{int}}(\log N)(k + \log k \log\log N))$$

bit operations. The space complexity is $O(k \log N)$ bits.

Proof. We first form the subproduct tree associated to the values F_i as follows. By completing with enough 1's, we can assume that k is a power of 2, *i.e.* $k = 2^m$. First, we define $F_{0,i} = F_i$ for $i = 1, \ldots, k$; then iteratively we let $F_{j,i} = F_{j-1,2i-1}F_{j-1,2i} \bmod N$, for $j = 1, \ldots, m$ and $i = 1, \ldots, k/2^j$. Computing this subproduct tree requires at most k multiplications modulo N. The space complexity is $O(k \log N)$ bits.

Now, by computing $\gcd(F_{m,1}, N)$, we can decide whether all of $F_1, \ldots, F_k$ are invertible modulo N. If this is not the case, finding one i for which $\gcd(F_i, N) > 1$ amounts to go down the tree from the root to one leave. Suppose indeed that we have determined that $\gcd(F_{j,i}, N) > 1$ for some $j > 0$. Computing $\gcd(F_{j-1,2i-1}, N)$ enables us to determine one of $F_{j-1,2i-1}$ or $F_{j-1,2i}$ which has a non-trivial gcd with N. Since the tree has depth $\log k$, we only do $\log k$ gcd computations, so our claim on the time complexity is proved. The space complexity does not increase, concluding the proof. □

Our second intermediate result is the cornerstone of the algorithm; we essentially improve the estimate of [42, Theorem 19.3] using Theorem 2.

Lemma 3 *Let b and N be positive integers with $2 \le b < N$. Then one can compute a prime divisor of N bounded by b, or prove that no such divisor exists, in*

$$O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\mathsf{M}(\sqrt{b}) + \log b \log\log N)\big)$$

bit operations. The space complexity is $O(\sqrt{b}\log N)$ bit operations.

Proof. Let $c = \lfloor\sqrt{b}\rfloor$ and let us consider the polynomials of $\mathbb{Z}/N\mathbb{Z}[X]$

$$f(X) = X + 1 - c$$

$$F(X) = \prod_{k=0}^{c-1}(X + ck + 1) = f(X + c^2)\cdots f(X + 2c)f(X + c).$$

Our first goal is to compute the values

$$F(0) = \prod_{k=0}^{c-1}(ck+1), \quad F(1) = \prod_{k=0}^{c-1}(ck+2), \quad \ldots, \quad F(c-1) = \prod_{k=0}^{c-1}(ck+c).$$

To this effect, we have in mind to apply Theorem 2 with $\alpha = c$, $\beta = 1$ and $k = c$. This theorem can be applied under suitable invertibility conditions; without getting into details, it suffices to note that these conditions impose that $3c + \log c \in O(\sqrt{b})$ integers bounded by $c^2 \leq b$ must be invertible modulo N. All these integers are non-zero, and less than N, thus non-zero in $\mathbb{Z}/N\mathbb{Z}$.

Applying Lemma 2, we deduce that we can test whether one of these integers is not invertible modulo N. If this is the case, we can find one such integer in

$$O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\sqrt{b} + \log b \log\log N)\big)$$

bit operations. Then by trial division, we can find a prime factor of N bounded by b in $O(\mathsf{M}_{\mathrm{int}}(\log N)\sqrt{b})$ bit operations; in this case, the theorem is proved.

Thus, we can now assume that all invertibility conditions are satisfied. Then by Theorem 2, all values $F(i)$, $i = 0, \ldots, c-1$ can be computed in

$$O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\mathsf{M}(c) + \log\log N)\big) \subset O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\mathsf{M}(\sqrt{b}) + \log\log N)\big)$$

bit operations (in the estimate above, the $\mathsf{M}_{\mathrm{int}}(\log N)\log\log N$ term stands for the cost of one inversion modulo N).

Suppose that N admits a prime factor bounded by c^2, then $\gcd(F(i), N) > 1$ for some i in $\{0, 1, \ldots, c-1\}$. By Lemma 2, such an i can be found in time

$$O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\sqrt{b} + \log b \log\log N)\big).$$

Now, $\gcd(F(i), N) > 1$ means that $\prod_{k=0}^{c-1}(ck+i+1)$ and N have a non-trivial gcd; thus, $\gcd(ck+i+1, N) > 1$ for some k. Applying Lemma 2 again, the element $ck+i+1$ can be found in time

$$O\big(\mathsf{M}_{\mathrm{int}}(\log N)(\sqrt{b} + \log b \log\log N)\big).$$

By definition, $ck + i + 1 \leq b$, so by trial division, we can then find a prime factor of N bounded by b in $O(\mathsf{M}_{\mathrm{int}}(\log N)\sqrt{b})$ bit operations.

At this point, if we have not found any prime divisor, then we have certified that N has no prime divisor in $2, \ldots, c^2$, so we finally inspect the range $c^2 + 1, \ldots, b$. Since $b - c^2 \in O(\sqrt{b})$, and since all these numbers are in $O(b)$, we apply the same strategy as above: we first apply Lemma 2 to test if a prime divisor of N exists in this range. If so, we find it by trial division; we obtain the same time complexity bounds as above.

The space complexity estimate follows from the space estimates of Theorem 2 and Lemma 2. □

The proof of Theorem 3 follows from successive applications of the lemma above with an increasing value for b. Starting with $b = 2$, the algorithm of Lemma 3 is then run with the same value of b until no more factors smaller than b remain in N; then the value of b is doubled. The algorithm stops as soon as $b \geq \sqrt{N}$.

The space complexity estimate follows immediately. However, analyzing the time complexity requires more work. Indeed, since there are at most $\log N$ prime factors of N, a rough upper bound on the runtime of the whole factorization would be $\log N$ times the runtime of Lemma 3 with $b = \sqrt{N}$. We show now that this $\log N$ factor can in fact be avoided, thus proving Theorem 3.

For a given parameter b, in the above process, when Lemma 3 is run with this b, all prime divisors of N less than $b/2$ have already been found and divided out. Therefore the primes that can be detected are greater than $b/2$; since their product is bounded by N, we deduce that the number of runs of Lemma 3 for the parameter b is upper-bounded by $O(\log N/\log b)$.

In the complexity estimate of Lemma 3, the sum of all $\mathsf{M}_{\text{int}}(\log N)\log b \log\log N$ terms is bounded by a polynomial in $\log N$, so its contribution in the total runtime is negligible. We are thus left with the problem of evaluating the following quantity:

$$\mathsf{M}_{\text{int}}(\log N)\sum_{i=1}^{\lceil 1/2\log N\rceil}\frac{\log N}{\log(2^i)}\mathsf{M}(2^{i/2}) = \mathsf{M}_{\text{int}}(\log N)\sum_{i=1}^{\lceil 1/2\log N\rceil}\frac{\log N}{i}\mathsf{M}(2^{i/2}).$$

A first obvious upper bound is

$$\mathsf{M}_{\text{int}}(\log N)\sum_{i=1}^{\lceil 1/2\log N\rceil}\left\lceil\frac{\log N}{i}\right\rceil\mathsf{M}(2^{i/2}).$$

Then, in the sum, the coefficients in front of the $\mathsf{M}(2^{i/2})$ are all at least 1; then, our hypothesis on the function M yields the following upper bound

$$\mathsf{M}_{\text{int}}(\log N)\mathsf{M}\left(\sum_{i=1}^{\lceil 1/2\log N\rceil}\left\lceil\frac{\log N}{i}\right\rceil 2^{i/2}\right).$$

The right hand side of the previous expression is then upper-bounded by a constant times $N^{1/4}$ using Cauchy's criterion. Putting all of this together we obtain the runtime announced in Theorem 3. □

For completeness, we briefly describe Pollard's algorithm [30], which is somehow similar to Strassen's. Pollard's algorithm chooses an integer α relatively prime with N and proceeds to search for an integer m less than $\sqrt{N}$ such that $\gcd(\alpha^m - 1, N) \neq 1$. To find such an m, it suffices to evaluate the polynomial $\prod_{i=0}^{c-1}(X-\alpha^i)$ on the geometric progression $\alpha^c, \alpha^{2c}, \ldots, \alpha^{c^2}$, where (as above) c is of order of $\sqrt[4]{N}$.

However, to ensure that a proper factor of N is found, one may have to run this procedure for many values of α (this complication arises when α has the same order modulo all the prime divisors of N). The number of such *bad* consecutive integers α can be bounded at best by $O\big(N^\delta\big)$, for each $\delta > 0$; this estimate can not be replaced by $O(\log(N))$. Since multipoint evaluation on the geometric progression α^{ci} can be done in $O(\mathsf{M}(c))$ operations modulo N [2, 1], this yields the estimate of

$$O\big(\mathsf{M}(\sqrt[4]{N})N^\delta\mathsf{M}_{\text{int}}(\log N)\big)$$

bit operations.

6 Computing one term in a linear recurrence

We now address the following problem: given an $n \times n$ matrix M with entries in $R[X]$, all of them of degree at most 1, and a vector of initial conditions U_0, define the sequence (U_i) of elements in R^n by the linear recurrence

$$U_{i+1} = M(i+1)U_i, \text{ for all } i \geq 0.$$

Given $N \in \mathbb{N}$, our question is then to compute the vector U_N. In the introduction, we gave the complexity of the algorithm of Chudnovsky and Chudnovsky for that task. We now present our improvement, which is quite similar to the one we obtained for integer factorization in the previous section.

Theorem 4 *Let M be a $n \times n$ matrix with linear entries in $R[X]$ and let U_0 be in R^n. Suppose that (U_i) is the sequence of elements in R^n defined by the linear recurrence*

$$U_{i+1} = M(i+1)U_i, \text{ for all } i \geq 0.$$

Let $N > 0$ be an integer and suppose that $1, \ldots, 2^s+1$ are units in R, where $s = \lfloor \log_4 N \rfloor$. Then the vector U_N can be computed within

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\,\mathsf{M}(\sqrt{N})\big)$$

ring operations in R and 1 inversion in R. The space requirement is $O(n^2\sqrt{N})$.

The proof of Theorem 4 is divided in two steps. We begin by proving the assertion in the particular case when N is a power of 4, then we treat the general case. The main reason for dealing with powers of 4 is that it makes it possible to control the list of elements that need to be units.

The case N is a power of 4. Let us suppose that $k = 2^s$ and $N = k^2$, so that $N = 4^s$. Let M_k be the $n \times n$ matrix over $R[X]$ defined by

$$M_k(X) = M(X+k)\cdots M(X+1),$$

where $M(X+i)$ denotes the matrix M with all polynomials evaluated at $X+i$. By assumption on M, the entries of M_k have degree at most k. The requested output U_N can be obtained by the equation

$$U_N = M_k(k(k-1))\cdots M_k(k)\,M_k(0)\,U_0. \tag{3}$$

With this choice of k, since $\beta = k$ is a unit, applying Theorem 2 with $\alpha = 1$ and $\beta = k$ shows that the values $M_k(0), M_k(k), \ldots, M_k((k-1)k)$ can be computed within the required time and space complexities. Due to the form of k, it is easy to check that the conditions of invertibility in Theorem 2 rewrite as $1, \ldots, k+1$ being units in R, which is true in our case.

Then, by Formula (3), the result is obtain after performing $\sqrt{N}$ successive matrix-vector products, which has a cost in both time and space of $O(n^2\sqrt{N})$.

THE GENERAL CASE. We now consider the general case. Let $N = \sum_{i=0}^{s} N_i 4^i$ be the 4-adic expansion of N, with $N_i \in \{0,1,2,3\}$ for all i. Given any $t \geq 0$, we will denote by $\lceil N \rceil^t$ the integer $\sum_{i=0}^{t-1} 4^i N_i$. Using this notation, we define a sequence $(V_t)_{0 \leq t \leq s}$ as follows: we let $V_0 = U_0$ and, for $0 \leq t \leq s$ we set

$$\begin{aligned} &V_{t+1} = M(\lceil N \rceil^t + 4^t N_t) \cdots M(\lceil N \rceil^t + 1) V_t, \quad \text{if } N_t \in \{1,2,3\} \\ &V_{t+1} = V_t, \quad \text{if } N_t = 0. \end{aligned} \tag{4}$$

It is easy to verify that $V_{s+1} = U_N$. Therefore, it suffices to compute the sequence (V_t) within the desired complexities.

Supposing that the term V_t has been determined, we estimate the cost of computing the next term V_{t+1}. If N_t is zero, we have nothing to do. Otherwise, we let $V_{t+1}^{(0)} = V_t$, and, for $1 \leq j \leq N_t$, we let $M^{(j)}(X) = M\left(X + \lceil N \rceil^t + 4^t(j-1)\right)$. Then we define $V_{t+1}^{(j)}$ by

$$V_{t+1}^{(j)} = M^{(j)}(4^t) \cdots M^{(j)}(1) V_{t+1}^{(j-1)}, \quad j = 1, \ldots, N_t.$$

By Equation (4), we have $V_{t+1}^{(N_t)} = V_{t+1}$. Thus, passing from V_t to V_{t+1} amounts to computing N_t selected terms of a linear recurrence of the special form treated in the previous paragraph. Using the complexity result therein and the fact that all N_t are bounded by 3, the total cost of the general case is thus

$$O\left(\sum_{t=0}^{s} \left(\mathsf{MM}(n) 2^t + n^2\, \mathsf{M}(2^t)\right)\right) = O\left(\mathsf{MM}(n) 2^s + n^2 \left(\sum_{t=0}^{s} \mathsf{M}(2^t)\right)\right).$$

Using the fact that $2^s \leq \sqrt{N} \leq 2^{s+1}$ and the assumptions on the function M, we easily deduce that the whole complexity fits into the bound $O\big(\mathsf{MM}(n)\sqrt{N} + n^2\, \mathsf{M}\big(\sqrt{N}\big)\big)$, as claimed. The requirements of invertibility in R follow from the previous paragraph, and all the inversions that are going to occur in the computation can be predicted in advance, thus allowing the use of Lemma 1 to have only 1 inversion to compute.

Similar considerations also yield the bound concerning the memory requirements. This concludes the proof of Theorem 4. □

7 Computing several terms in a linear recurrence

In the previous section, we dealt with the computation of one term. We now study the case where several terms are asked; this question will find an application in the next section, for the computation of the Cartier-Manin operator.

We take the same notation as in Section 6. Let M be an $n \times n$ matrix whose entries are degree 1 polynomials over a ring R. We consider the linear recurrent sequence defined for all $i \geq 0$ by $U_{i+1} = M(i+1)U_i$, and let U_0 be a vector in R^n. Given r indices $N_1 < N_2 < \cdots < N_r$, we want to compute all the values $U_{N_i}, 1 \leq i \leq r$.

An obvious solution is to repeatedly apply Theorem 4 to each interval $[N_i, N_{i-1}]$, leading to a time complexity of

$$O\left(\mathsf{MM}(n) \sum_{i=1}^{r} \sqrt{N_i - N_{i-1}} + n^2 \sum_{i=1}^{r} \mathsf{M}\left(\sqrt{N_i - N_{i-1}}\right)\right)$$

and with a memory requirement of $O(n^2 \max_{i=1}^{r} \sqrt{N_i - N_{i-1}})$, where we have set $N_0 = 0$. In special cases, this might be close to the optimal, for instance if $N_1, \ldots, N_{r-1}$ are small compared to N_r. However, in most cases it is possible to improve on this: we now present an alternative algorithm, which improves on the time complexity, at the cost however of increased memory requirements.

Theorem 5 *Let $N_1 < N_2 < \cdots < N_r = N$ be positive integers. Suppose that $1, \ldots, 2^s+1$ are units in R, where $s = \lfloor \log_4 N \rfloor$. Suppose furthermore that $r < N^{\frac{1}{2}-\varepsilon}$, with $0 < \varepsilon < \frac{1}{2}$. Then the vectors $U_{N_1}, \ldots, U_{N_r}$ can be computed within*

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\mathsf{M}(\sqrt{N})\big)$$

operations $(+, -, \times)$ in R, 1 inversion, and using $O(n^2\sqrt{N})$ space.

In other words, the complexity of computing several terms is essentially the same as the complexity of computing that of largest index, as long as the total number of terms to compute is not too large. In the case where the N_i form an arithmetic progression, and if the multiplication function M is essentially linear, we gain a factor of $\sqrt{r}$ compared to the naive approach. Note also that in the limiting case where $r = \sqrt{N}$, and for fixed size n, our algorithm is optimal up to logarithmic factors, as it takes a time essentially linear in the size of the output.

Proof. The algorithm proceeds as follows: we start by applying Theorem 2 with $k \approx \sqrt{N}$, so as to compute k terms in the sequence with indices separated by intervals of size about $\sqrt{N}$. If all the indices N_i have been reached, then we are done. Else, we have reached r indices that are within a distance of about $\sqrt{N}$ of $N_1, \ldots, N_r$. A recursive refining procedure is then done simultaneously for all these r indices. When we get close enough to the wanted indices, we finish with a divide-and-conquer method. The refining and the final steps make use of Corollary 1.

Let us describe more precisely the first step (named Step 0), the ith refining step, and the final step. In what follows, given $k \in \mathbb{N}$, we denote by M_k the polynomial matrix

$$M_k(X) = M(X + k) \cdots M(X + 1).$$

Step 0. Put $k_0 = 2^{\lfloor \log_4 N \rfloor}$, and set $N_1^{(0)} = k_0 \left\lfloor \frac{N_1}{k_0} \right\rfloor, \ldots, N_r^{(0)} = k_0 \left\lfloor \frac{N_r}{k_0} \right\rfloor$. Note that for all $j = 1, \ldots, r$, we have $0 \leq N_j - N_j^{(0)} < k_0$ (that is, $N_j^{(0)}$ is the largest multiple of k_0 smaller than, or equal to N_j). We now show how to compute the values

$$U_{N_1^{(0)}}, \ldots, U_{N_r^{(0)}},$$

the idea being to compute the vectors $U_{k_0\ell}$, with ℓ starting from 1.

Applying Theorem 2 with $\alpha = 1$ and $\beta = k = k_0$, we compute the matrices

$$M_{k_0}(0), M_{k_0}(k_0), \ldots, M_{k_0}((k_0 - 1)k_0).$$

Due to the fact that k is a power of 2, the elements that have to be inverted are $1, \ldots, k_0 + 1$. The vector U_0 is then successively multiplied by these matrices, yielding

$$U_{k_0}, U_{2k_0}, \ldots, U_{k_0^2}.$$

Now, the effect of taking the floor part in the definition of k_0 implies that

$$\frac{N}{4} < k_0^2 \leq N,$$

Therefore, with our choice of k_0, we might cover only one fourth of the interval $[0, N]$. To complete the sequence, we repeat up to 3 times the same strategy, starting with $V_0 = U_{k_0^2}$ and shifting the matrix M accordingly. Among all the resulting vectors, we collect the requested values $U_{N_1^{(0)}}, \ldots, U_{N_r^{(0)}}$.

Step i**.** We assume that coming out from Step $(i-1)$, we are given an integer k_{i-1} and indices $N_1^{(i-1)}, \ldots, N_r^{(i-1)}$, where for all $j = 1, \ldots, r$, we have $0 \leq N_j - N_j^{(i-1)} < k_{i-1}$. We also suppose that the values $U_{N_1^{(i-1)}}, \ldots, U_{N_r^{(i-1)}}$ have been computed. In other words, at Step $(i-1)$, we have computed r vectors whose indices are within a distance of k_{i-1} of the wanted indices.

The ith refining step is as follows: we set

$$k_i = \left\lceil \sqrt{r k_{i-1}} \right\rceil,$$

and the new terms that we want to compute correspond to the indices

$$N_1^{(i)} = N_1^{(i-1)} + k_i \left\lfloor \frac{N_1 - N_1^{(i-1)}}{k_i} \right\rfloor, \ \ldots, \ N_r^{(i)} = N_r^{(i-1)} + k_i \left\lfloor \frac{N_r - N_r^{(i-1)}}{k_i} \right\rfloor.$$

This time, we apply Corollary 1 to evaluate the new polynomial matrix $M_{k_i}(X)$ at the points

$$\begin{array}{c} N_1^{(i-1)}, \ N_1^{(i-1)} + k_i, \ \ldots, \ N_1^{(i-1)} + \left(\left\lfloor \frac{k_{i-1}}{k_i} \right\rfloor - 1 \right) k_i, \\ N_2^{(i-1)}, \ N_2^{(i-1)} + k_i, \ \ldots, \ N_2^{(i-1)} + \left(\left\lfloor \frac{k_{i-1}}{k_i} \right\rfloor - 1 \right) k_i, \\ \vdots \\ N_r^{(i-1)}, \ N_r^{(i-1)} + k_i, \ \ldots, \ N_r^{(i-1)} + \left(\left\lfloor \frac{k_{i-1}}{k_i} \right\rfloor - 1 \right) k_i. \end{array}$$

Note that there are at most k_i points of evaluation. The vectors coming from Step $(i-1)$ are then multiplied by the appropriate matrices to get the values

$$U_{N_1^{(i)}}, \ldots, U_{N_r^{(i)}}.$$

Final step. The refining process stops when k_i is close to r, for instance $k_i < 2r$. In this situation, we have at our disposal r indices, say $N_1', \ldots, N_r'$, such that $0 \leq N_j - N_j' < 2r$ for all j, and such that all values $U_{N_1'}, \ldots U_{N_r'}$ are known.

Then another recursive algorithm is used: $M_r(X)$ is computed and used to reduce all gaps that were of size between r and $2r$, by applying Corollary 1. As a result all the gaps are now of size at most r. Then $M_{\frac{r}{2}}(X)$ is computed and used to reduce the gaps to at most $\frac{r}{2}$, followed by $M_{\frac{r}{4}}(X)$ to reduce the gaps to $\frac{r}{4}$, and so on until we get the result.

First we show that the algorithm stops in a finite number of refining steps. At Step i, the indices are at distance at most k_i of the wanted indices. But (neglecting the effect of the ceiling), the value of k_i is bounded by $rN^{\frac{1}{2^{i+1}}}$ for all i, which demonstrates that after $O(\log\log N)$ steps, k_i is close enough to r and we reach the final step.

It remains to perform the complexity analysis. The cost of Step 0 is dominated by that of evaluating the matrix $M_{k_0}(X)$; by Theorem 2, it can be done in

$$O\left(\mathsf{MM}(n)\sqrt{N} + n^2\mathsf{M}(\sqrt{N})\right)$$

operations in R, and space $O(n^2\sqrt{N})$. By Corollary 1, the ith refining step has a cost of

$$O\left(\mathsf{MM}(n)k_i + n^2\mathsf{M}(k_i)\log k_i\right)$$

operations, and requires $O(n^2k_i + k_i\log k_i)$ temporary memory allocation. We then have to evaluate the formula

$$\sum_{i=1}^{i_{\max}} \left(\mathsf{MM}(n)k_i + n^2\mathsf{M}(k_i)\log k_i\right),$$

where $i_{\max} = O(\log\log N)$. From the recursive definition of k_i, we deduce that

$$k_i = O\left(r^{\frac{2^i-1}{2^i}} N^{\frac{1}{2^{i+1}}}\right).$$

Using the upper bound $r < N^{\frac{1}{2}-\varepsilon}$, we get

$$k_i = O\left(N^{\frac{1}{2}}\left(N^{-\varepsilon}\right)^{\frac{2^i-1}{2^i}}\right).$$

But for any $0 < \ell \leq 1$ we have $\sum_{i=1}^{i_{\max}} \ell^{\frac{2^i-1}{2^i}} \leq i_{\max}\ell^{\frac{1}{2}}$; therefore we deduce

$$\sum_{i=1}^{i_{\max}} k_i = O\left(N^{\frac{1-\varepsilon}{2}}\log\log N\right).$$

On the other hand, our hypotheses on the function M show that the cost of all refining steps fits into the bound

$$O\left(\mathsf{MM}(n)\sum_{i=1}^{i_{\max}} k_i + n^2\mathsf{M}\left(\sum_{i=1}^{i_{\max}} k_i\right)\log\left(\sum_{i=1}^{i_{\max}} k_i\right)\right).$$

Thus, it is negligible compared with the cost of the first step, since the $N^{-\varepsilon/2}$ contribution compensates all the supplementary logarithmic factors. For the same reason, the memory requirement is negligible compared with that of Step 0.

Using Corollary 1, the cost of the first reduction in the final step is

$$O\left(\mathsf{MM}(n)r + n^2\mathsf{M}(r)\log r\right)$$

ring operations, 1 inversion and $O(n^2r + r\log r)$ temporary space. Due to our hypotheses on the function M, the second reduction costs at most half as much, the third reduction

costs at most $\frac{1}{4}$ of it, ... Summing up, we see that the whole cost of the final step is bounded by a constant times that of the first reduction, which is itself less than the cost of Step 0.

To conclude the proof of Theorem 5, we have to pay attention to what elements of R are to be inverted. As said before, in Step 0, the elements $1, \ldots, k_0 + 1$ need to be inverted. In the ith refining Step and in the final Step, only Corollary 1 is called, and it is always used with a value of k which is smaller than k_0. Therefore, no further element needs to be a unit. As before, all inversions can be grouped together using Lemma 1. □

8 Application to the Cartier-Manin operator on hyperelliptic curves

We conclude this paper by presenting an application of Theorems 4 and 5 to the computation of the Cartier-Manin operator on hyperelliptic curves.

Let $\mathcal{C}$ be a hyperelliptic curve of genus g defined over the finite field $\mathbb{F}_{p^d}$ with p^d elements, where p is the characteristic of $\mathbb{F}_{p^d}$. We suppose that $p > 2$ and that the equation of $\mathcal{C}$ is of the form $y^2 = f(x)$, where $f \in \mathbb{F}_{p^d}[X]$ is a monic squarefree polynomial of degree $2g + 1$. The generalization to hyperelliptic curves of the Hasse invariant for elliptic curves is the so-called Hasse-Witt matrix, which is defined as follows: Let h_k be the coefficient of degree x^k in the polynomial $f^{(p-1)/2}$. The Hasse-Witt matrix is the $g \times g$ matrix with coefficients in $\mathbb{F}_{p^d}$ given by $H = (h_{ip-j})_{1 \leq i,j \leq g}$.

This matrix was introduced in [16]; it represents the operator on differential forms introduced by Cartier [10] in a suitable basis. Manin then showed in [24] that this matrix is strongly related to the action of the Frobenius endomorphism on the p-torsion part of the Jacobian of $\mathcal{C}$. The article [43] provides a complete survey about those facts; they can be summarized by the following theorem:

Theorem 6 (Manin) *Let $\mathcal{C}$ be a hyperelliptic curve of genus g defined over $\mathbb{F}_{p^d}$. Let H be the Hasse-Witt matrix of $\mathcal{C}$ and let $H_\pi = HH^{(p)} \cdots H^{(p^{d-1})}$, where the notation $H^{(q)}$ means element-wise raising to the power q. Let $\kappa(t)$ be the characteristic polynomial of the matrix H_π and $\chi(t)$ the characteristic polynomial of the Frobenius endomorphism of the Jacobian of $\mathcal{C}$. Then $\chi(t) \equiv (-1)^g t^g \kappa(t) \mod p$.*

This result provides a quick method to compute the characteristic polynomial of the Frobenius endomorphism and hence the group order of the Jacobian of $\mathcal{C}$ modulo p, when p is not too large. Combined with a Schoof-like algorithm and / or a baby steps / giant-steps algorithm, it can lead to a full point-counting algorithm, in particular for genus 2 curves, as was demonstrated in [14, 25].

To compute the entries of the Hasse-Witt matrix, the obvious solution consists in expanding the product $f^{(p-1)/2}$. Using balanced multiplications, and taking all products modulo X^{gp} this can be done in $O(\mathsf{M}(gp) \log p)$ base ring operations, see [42, Chapter 10], whence a time complexity that is essentially linear in p, if g is kept constant. In what follows, we show how to obtain a complexity essentially linear in $\sqrt{p}$ using the results of the previous sections.

We make the additional assumption that the constant term of f is not zero. Otherwise, the problem is actually simpler: writing $f = Xf_1$, the coefficient of degree $ip - j$ in

$f^{(p-1)/2}$ is the coefficient of degree $ip-j-(p-1)/2$ in $f_1^{(p-1)/2}$. Hence we can work with a polynomial of degree $2g$ instead of $2g+1$ and the wanted degrees are slightly less.

Introduction of a linear recurrent sequence. In [12], Flajolet and Salvy already treat the question of computing a selected coefficient in a high power of some given polynomial, as an answer to a SIGSAM challenge. The key point of their approach is that $h = f^{(p-1)/2}$ satisfies the following first-order linear differential equation

$$fh' - \frac{p-1}{2} f'h = 0.$$

From this, we deduce that the coefficients of h satisfy a linear recurrence of order $2g+1$, with coefficients that are rational functions of degree 1.

Explicitly, let us denote by h_k the coefficient of degree k of the polynomial h, and for convenience, set $h_k = 0$ for $k < 0$. Similarly, the coefficient of degree k of f is denoted by f_k. From the differential equation above, for all k in $\mathbb{Z}$, we deduce that

$$(k+1) f_0 h_{k+1} + \left(k - \frac{p-1}{2}\right) f_1 h_k + \cdots + \left(k - 2g - \frac{(2g+1)(p-1)}{2}\right) f_{2g+1} h_{k-2g} = 0.$$

We set $U_k = [h_{k-2g}, h_{k-2g+1}, \ldots, h_k]^t$, and let $A(k)$ be the companion matrix:

$$A(k) = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 1 \\ \frac{f_{2g+1}((2g+1)(p-1)/2-(k-2g-1))}{f_0 k} & \cdots & \cdots & \cdots & \frac{f_1((p-1)/2-k+1)}{f_0 k} \end{pmatrix}.$$

The initial vector $U_0 = [0, \ldots, 0, f_0^{(p-1)/2}]^t$ can be computed using binary powering techniques in $O(\log p)$ operations; then for $k \geq 0$, we have $U_{k+1} = A(k+1)U_k$. Thus, to answer our specific question, it suffices to note that the vector U_{ip-1} gives the coefficients h_{ip-j} for $j = 1, \ldots, g$ that form the ith row of the Hasse-Witt matrix of $\mathcal{C}$.

Theorems 4 and 5 cannot be directly applied to this sequence, because $A(k)$ has entries that are rational functions, not polynomials. Though the algorithm could be adapted to handle the case of rational functions, we rather use the very specific form of the matrix $A(k)$, so only a small modification is necessary. Let us define a new sequence V_k by the relation

$$V_k = f_0^k k! U_k.$$

Then, this sequence is linearly generated and we have $V_{k+1} = B(k+1)V_k$, where

$$B(k) = f_0 k A(k).$$

Therefore, the entries of the matrix $B(k)$ are polynomials of degree at most 1. Note also that the denominators $f_0^k k!$ satisfy the recurrence relation

$$f_0^{k+1}(k+1)! = (f_0(k+1)) \cdot (f_0^k k!).$$

Thus, we can apply the results of the previous sections to compute separately the required vectors V_k, as well as the corresponding denominators: specifically, we will compute $V_{p-1}, V_{2p-1}, \ldots V_{gp-1}$ as well as $f_0^{p-1}(p-1)!, \ldots, f_0^{gp-1}(gp-1)!$. Then the vectors $U_{p-1}, U_{2p-1}, \ldots U_{gp-1}$ are deduced using the relation $f_0^k k!\, U_k = V_k$.

Lifting to characteristic zero. A difficulty arises from the fact that the characteristic is too small compared to the degrees we are aiming to, so $p!$ is zero in $\mathbb{F}_{p^d}$. The workaround is to do computations in the unramified extension K of $\mathbb{Q}_p$ of degree d, whose residual field is $\mathbb{F}_{p^d}$. The ring of integers of K will be denoted by O_K, so that any element of O_K can be reduced modulo p to give an element of $\mathbb{F}_{p^d}$. On the other hand, K has characteristic 0, so p is invertible in K.

We consider an arbitrary lift of f to $O_K[X]$. The reformulation in terms of linear recurrent sequence made in the paragraph above can be performed over K; the coefficients of $f^{(p-1)/2}$ are computed as elements of K and then projected back onto $\mathbb{F}_{p^d}$. This is possible, as they all belong to O_K.

We separately compute the values in K of the vectors V_{ip-1} and the denominators $f_0^{ip-1}(ip-1)!$, for $i = 1, \ldots, g$. To this effect, we can apply any of the strategies mentioned in Section 7: the first one is the plain iteration based on Theorem 4; iterating g times, it performs

$$O\left(\mathsf{MM}(g)g\sqrt{p} + g^3\mathsf{M}(\sqrt{p})\right)$$

operations in K and requires to store $O(g^2\sqrt{p})$ elements. The second strategy is to apply Theorem 5. In this case, we need to have $g \leq (gp)^{1/2-\varepsilon'}$, for some $0 < \varepsilon' < \frac{1}{2}$; this is equivalent to impose that $g \leq p^{1-\varepsilon}$ for some $0 < \varepsilon < 1$. Then the number of operations in K reduces to

$$O\left(\mathsf{MM}(g)\sqrt{gp} + g^2\mathsf{M}(\sqrt{gp})\right),$$

but the memory requirements grows to $O(g^2\sqrt{gp})$ elements of K.

Computing at fixed precision. Of course, we do not want to compute in the field K at arbitrary precision: for our purposes, it suffices to truncate all computations modulo a suitable power of p. To evaluate the required precision of the computation, we need to check when the algorithm operates a division by p.

To compute the vectors V_{ip-1} and the denominators $f_0^{ip-1}(ip-1)!$, for $i = 1, \ldots, g$, we use either Theorem 4 or Theorem 5. In the worst case, it might be required to invert all the integers up to $\sqrt{gp}$. With the condition $g \leq p^{1-\varepsilon}$, all these numbers are strictly smaller than p, and they are therefore units in O_K. Hence no division by p occurs in this first phase.

Then, for all $i = 1, \ldots, g$, to deduce U_{ip-1} from V_{ip-1}, we need to divide by $f_0^{ip-1}(ip-1)!$. The element f_0 is a unit in O_K, so the only problem comes from the factorial term. If $i < p$, then the p-adic valuation of $(ip-1)!$ is exactly $i-1$. Therefore the worst case is $i = g$, for which we have to divide by p^{g-1}. Hence computing the vectors V_{ip-1} modulo p^g is enough to know the vectors U_{ip-1} modulo p, and then to deduce the Hasse-Witt matrix.

Overall complexity. From the results in Section 2, storing an element of O_K/p^gO_K requires $O(dg\log p)$ bits, and multiplying two elements of O_K/p^gO_K has a complexity of $O(\mathsf{M}(d)\mathsf{M}_{\text{int}}(g\log p))$ bit operations. From the results of Sections 6 and 7, we deduce the following theorem on the complexity of computing the Hasse-Witt matrix. We give two variants, that follow the two strategies described above.

Theorem 7 *Let p be a prime ≥ 3, $d \geq 0$ and $\mathcal{C}$ a hyperelliptic curve defined over $\mathbb{F}_{p^d}$ by the equation $y^2 = f(x)$, with f of degree $2g+1$. Then, assuming $g < p^{1-\varepsilon}$ for some $0 < \varepsilon < 1$, one can compute the Hasse-Witt matrix of $\mathcal{C}$ using one of the two following strategies:*

1. *A memory-efficient strategy gives a complexity of*

$$O\left(\left(\mathsf{MM}(g)g\sqrt{p} + g^3\mathsf{M}(\sqrt{p})\right)\mathsf{M}(g)\mathsf{M}_{\text{int}}(d\log p)\right)$$

   bit-operations and $O\left(dg^3\sqrt{p}\ \log p\right)$ storage.

2. *A time-efficient strategy gives a complexity of*

$$O\left(\left(\mathsf{MM}(g)\sqrt{gp} + g^2\mathsf{M}(\sqrt{gp})\right)\mathsf{M}(g)\mathsf{M}_{\text{int}}(d\log p)\right)$$

   bit-operations, with $O\left(dg^3\sqrt{gp}\ \log p\right)$ storage.

The matrix H by itself gives some information on the curve $\mathcal{C}$: for instance, H is invertible if and only if the Jacobian of $\mathcal{C}$ is ordinary [43, Cor 2.3]. However, as stated in Theorem 6, the matrix H_π and in particular its characteristic polynomial $\chi(t)$ tell much more and are required if the final goal is point-counting. We now concentrate on the cost of computing the characteristic polynomial of H_π.

The matrix H_π is the "norm" of H and as such can be computed with a binary powering algorithm. For simplicity, we assume that d is a power of 2. Then, denoting

$$H_{\pi,i} = HH^{(p)}\cdots H^{(p^{2^i-1})},$$

we have

$$H_{\pi,i+1} = H_{\pi,i}\cdot\left(H_{\pi,i}\right)^{(p^{2^i})}.$$

Hence computing $H_{\pi,i+1}$ from $H_{\pi,i}$ costs one matrix multiplication and 2^i matrix conjugations. A matrix conjugation consists in raising all the entries to the power p, therefore it costs $O(g^2\log p)$ operations in $\mathbb{F}_{p^d}$. The matrix we need to compute is $H_\pi = H_{\pi,\log d}$. Hence the cost of computing H_π is

$$O\left(dg^2\log(p) + \mathsf{MM}(g)\log d\right)$$

operations in $\mathbb{F}_{p^d}$. The general case where d is not a power of 2 is handled by adjusting the recursive step according to the binary expansion of d and yields the same complexity up to a constant factor. The cost of the characteristic polynomial computation is bounded by the cost of a matrix multiplication up to a logarithmic factor [20], [8, Chapter 16] and is therefore negligible compared to the other costs.

If we are interested only in the complexity in p and d, *i.e.* if we assume that the genus is fixed, then the two strategies of Theorem 7 become equivalent, up to constant factors, and we get a total time complexity for computing $\chi(t)$ mod p of

$$O\left(\left(\mathsf{M}(\sqrt{p}) + d\log p\right)\mathsf{M}(d)\mathsf{M}_{\text{int}}(\log p)\right)$$

bit operations and $O\left(d\sqrt{p}\ \log p\right)$ storage.

Case of large genus. If $g \geq p$, then our analysis is no longer valid. However, in that case, the algorithms of Theorems 4 and 5 are asymptotically not the fastest. In this paragraph, we assume that the function M is essentially linear and we do not take into account the logarithmic factors (adding appropriate epsilons in the exponents would yield a rigorous analysis). Then the cost in bit-operations of Strategy 1 of Theorem 7 is at least $g^4 d\sqrt{p}$, the cost of Strategy 2 is at least $g^3 d\sqrt{gp}$ and the cost of the naive algorithm is in $O(gpd)$.

Assume that $g > p^{\frac{1}{6}}$. Then it follows that $g^4\sqrt{p} > gp$, and therefore the naive algorithm is faster than Strategy 1 of Theorem 7. Assume now that $g > p^{\frac{1}{5}}$. Then it follows that $gp < g^3\sqrt{gp}$, and the naive algorithm is faster than Strategy 2 of Theorem 7.

As a conclusion, whatever the strategy used in Theorem 7, the parameter range for which our algorithms are interesting easily satisfies the technical condition $g < p^{1-\varepsilon}$.

Acknowledgements. We thank Bruno Salvy for his comments on this paper, and Joachim von zur Gathen and Jürgen Gerhard for answering our questions on the complexity of integer factorization.

References

[1] A. V. Aho, K. Steiglitz, and J. D. Ullman. Evaluating polynomials at fixed sets of points. *SIAM J. Comput.*, 4(4):533–539, 1975.

[2] L. I. Bluestein. A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Trans. Electroacoustics*, AU-18:451–455, 1970.

[3] J. L. Bordewijk. Inter-reciprocity applied to electrical networks. *Appl. Sci. Res. B.*, 6:1–74, 1956.

[4] A. Borodin. Time space tradeoffs (getting closer to the barrier?). In *ISAAC*, volume 762 of *Lecture Notes in Comput. Sci.*, pages 209–220. Springer–Verlag, 1993.

[5] A. Borodin and R. T. Moenck. Fast modular transforms. *Comput. System Sci.*, 8(3):366–386, 1974.

[6] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves. In *International Conference on Finite Fields and Applications (Toulouse, 2003)*, volume 2948 of *Lecture Notes in Comput. Sci.*, pages 40–58. Springer-Verlag, 2004.

[7] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM Press, 2003.

[8] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1997.

[9] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[10] P. Cartier. Une nouvelle opération sur les formes différentielles. *C. R. Acad. Sci. Paris*, 244:426–428, 1957.

[11] D. V. Chudnovsky and G. V. Chudnovsky. Approximations and complex multiplication according to Ramanujan. In *Ramanujan revisited (Urbana-Champaign, Ill., 1987)*, pages 375–472. Academic Press, Boston, MA, 1988.

[12] P. Flajolet and B. Salvy. The SIGSAM challenges: Symbolic asymptotics in practice. *SIGSAM Bulletin*, 31(4):36–47, 1997.

[13] P. Gaudry and N. Gürel. Counting points in medium characteristic using Kedlaya's algorithm. *Experiment. Math.*, 12:395–402, 2003.

[14] P. Gaudry and R. Harley. Counting points on hyperelliptic curves over finite fields. In *ANTS-IV*, volume 1838 of *Lecture Notes in Comput. Sci.*, pages 313–332. Springer-Verlag, 2000.

[15] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm, I. Speeding up the division and square root of power series. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[16] H. Hasse and E. Witt. Zyklische unverzweigte Erweiterungskörper vom Primzahlgrade p über einem algebraischen Funktionenkörper der Charakteristik p. *Monatsch. Math. Phys.*, 43:477–492, 1936.

[17] E. Horowitz. A fast method for interpolation using preconditioning. *Inf. Proc. Letters*, 1(4):157–163, 1972.

[18] E. Kaltofen, R. M. Corless, and D. J. Jeffrey. Challenges of symbolic computation: my favorite open problems. *J. Symbolic Comput.*, 29(6):891–919, 2000.

[19] K. S. Kedlaya. Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology. *J. Ramanujan Math. Soc.*, 16(4):323–338, 2001.

[20] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoret. Comput. Sci.*, 36(2-3):309–317, 1985.

[21] D. E. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens (Nice, 1970), Tome 3*, pages 269–274. Gauthier-Villars, Paris, 1971.

[22] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *ACM Symposium on Theory of Computing*, pages 564–572, 1990.

[23] H. W. Lenstra, Jr. and C. Pomerance. A rigorous time bound for factoring integers. *J. Amer. Math. Soc.*, 5(3):483–516, 1992.

[24] J. I. Manin. The Hasse-Witt matrix of an algebraic curve. *Trans. Amer. Math. Soc.*, 45:245–264, 1965.

[25] K. Matsuo, J. Chao, and S. Tsujii. An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. In *ANTS-V*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 461–474. Springer-Verlag, 2002.

[26] J. McKee and R. Pinch. Old and new deterministic factoring algorithms. In *Algorithmic number theory (Talence, 1996)*, volume 1122 of *Lecture Notes in Comput. Sci.*, pages 217–224. Springer-Verlag, 1996.

[27] R. T. Moenck and A. Borodin. Fast modular transforms via division. *Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972)*, pages 90–96, 1972.

[28] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.*, 48(177):243–264, 1987.

[29] P. L. Montgomery. *An FFT extension of the elliptic curve method of factorization*. PhD thesis, University of California, Los Angeles CA, 1992.

[30] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.

[31] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In *Computational methods in number theory, Part I*, volume 154 of *Math. Centre Tracts*, pages 89–139. Math. Centrum, Amsterdam, 1982.

[32] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Inform.*, 1:139–144, 1971.

[33] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inform.*, 7:395–398, 1977.

[34] A. Schönhage, A. F. W. Grotefeld, and E. Vetter. *Fast algorithms*. Bibliographisches Institut, Mannheim, 1994.

[35] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[36] V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *ISSAC'91*, pages 14–21. ACM Press, 1991.

[37] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996.

[38] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[39] V. Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.*, 20:238–251, 1972/73.

[40] V. Strassen. Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, 78(1):1–8, 1976/77.

[41] B. Tellegen. A general network theorem, with applications. *Philips Research Reports*, 7:259–269, 1952.

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

[43] N. Yui. On the Jacobian varietes of hyperelliptic curves over fields of characteristic $p > 2$. *Journal of Algebra*, 52:378–410, 1978.