Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves

Alin Bostan[1], Pierrick Gaudry[2], Éric Schost[3]

[1] Laboratoire STIX, École polytechnique, France and IMAR, Romania
Alin.Bostan@stix.polytechnique.fr
[2] Laboratoire LIX, École polytechnique, France
gaudry@lix.polytechnique.fr
[3] Laboratoire STIX, École polytechnique, France
Eric.Schost@polytechnique.fr

Abstract. We improve an algorithm originally due to Chudnovsky and Chudnovsky for computing one selected term in a linear recurrent sequence with polynomial coefficients. Using baby-steps / giant-steps techniques, the nth term in such a sequence can be computed in time proportional to $\sqrt{n}$, instead of n for a naive approach.
As an intermediate result, we give a fast algorithm for computing the values taken by an univariate polynomial P on an arithmetic progression, taking as input the values of P on a translate on this progression.
We apply these results to the computation of the Cartier-Manin operator of a hyperelliptic curve. If the base field has characteristic p, this enables us to reduce the complexity of this computation by a factor of order $\sqrt{p}$. We treat a practical example, where the base field is an extension of degree 3 of the prime field with $p = 2^{32} - 5$ elements.

1 Introduction

In this paper, we investigate some complexity questions related to linear recurrent sequences. Specifically, we concentrate on recurrences with polynomial coefficients; our main focus is on the complexity of computing one selected term in such a recurrence.

A well-known particular case is that of recurrences with constant coefficients, where the nth term can be computed with a complexity that is logarithmic in n, using binary powering techniques.

In the general case, there is a significant gap, as for the time being no algorithm with complexity polynomial in $\log(n)$ is known. Yet, in [10], Chudnovsky and Chudnovsky proposed an algorithm that allows to compute one selected term in such a sequence without computing all intermediate ones. This algorithm appears as a generalization of those of Pollard [23] and Strassen [30] 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.

Our main contribution is an improvement of the algorithm of [10]; for simplicity, we only give the details in the case when all coefficients are polynomials of degree 1, as the study in the general case would follow in the same manner. The complexity of our algorithm is still (roughly) linear in $\sqrt{n}$; Chudnovsky and Chudnovsky actually suggested that this bound might be essentially optimal. We improve the time and space complexities by factors that are logarithmic in n; in practice, this is far from negligible, since in the application detailed below, n has order 2^{32}. A precise comparison with Chudnovsky and Chudnovsky's algorithm is made in Section 3.

Along the way, we also consider a question of basic polynomial arithmetic: given the values taken by a univariate polynomial P on a set of points, how fast can we compute the values taken by P on a translate of this set of points? An obvious solution is to make use of fast interpolation and evaluation techniques, but we show that one can do better when the evaluation points form an arithmetic sequence.

Computing the Cartier-Manin operator. Our initial motivation is an application to point-counting procedures in hyperelliptic curve cryptography, related to the computation of the Cartier-Manin operator of curves over finite fields. We now present these matters in more detail.

The Cartier-Manin operator of a curve defined over a finite field, together with the Hasse-Witt matrix, are useful tools to study the arithmetic properties of the Jacobian of that curve. Indeed, the supersingularity, and more generally the p-rank, can be read from the invariants of the Hasse-Witt matrix. In the case of hyperelliptic curves, this matrix was used in [13, 21] as part of a point-counting procedure for cryptographic-oriented applications.

Indeed, thanks to a result of Manin, computing the Cartier-Manin operator gives the coefficients of the Zeta function modulo p; this partial information can then be completed by some other algorithms. However, in [13] and [21], the method used to compute the Hasse-Witt matrix has a complexity which is essentially linear in p.

It turns out that one can do better. The entries of the Hasse-Witt matrix of a hyperelliptic curve $y^2 = f(x)$ defined over a finite field of characteristic p are coefficients of the polynomial $h = f^{(p-1)/2}$, so they satisfy a linear recurrence with rational function coefficients. Using our results on linear recurrences, this remark yields an algorithm to compute the Hasse-Witt matrix whose complexity now grows like $\sqrt{p}$, up to logarithmic factors, instead of p.

We demonstrate the interest of these techniques by a point-counting example, for a curve of genus 2 defined over a finite field whose characteristic just fits in one 32-bit machine word; this kind of fields have an interest for efficiency reasons [3].

Note finally that other point-counting algorithms, such as the p-adic methods used in Kedlaya's algorithm [18], also provide efficient point-counting procedures in small characteristic, but their complexity remains at least linear in p [12]. On the other hand, Kedlaya's algorithm outputs the whole Zeta function and should

be preferred if available. Therefore, the range of application of our algorithm is when the characteristic is too large for Kedlaya's algorithm to be run.

Organization of the paper. We start in Section 2 with our algorithm for shifting a polynomial given by its values on some evaluation points. This building block is used in Section 3 to describe our improvement on Chudnovsky and Chudnovsky's algorithm. In Section 4 we apply these results to the computation of the Cartier-Manin operator of a hyperelliptic curve. We conclude in Section 5 with a numerical example.

Notation. In what follows, we give complexity estimates in terms of number of base ring operations (additions, subtractions, multiplications and inversions of unit elements) and of storage requirements; this last quantity is measured in terms of number of elements in the ring. We pay particular attention to polynomial and matrix multiplications and use the following notation.

- Let R be a commutative ring; we suppose that R is unitary, its 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 $\mathbb{Z} \to R$. When the context is clear, we simply denote the ring element $\varphi(n)$ by n.
- We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function that represents the complexity of univariate 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 [25, 24, 8], $\mathsf{M}(d)$ can be taken in $O(d \log(d) \log(\log(d)))$.
  We suppose that the function M verifies the inequality $\mathsf{M}(d_1) + \mathsf{M}(d_2) \leq \mathsf{M}(d_1 + d_2)$ for all positive integers d_1 and d_2; in particular, the inequality $\mathsf{M}(d) \leq \frac{1}{2}\mathsf{M}(2d)$ holds for all $d \geq 1$. On the other hand, we make the (natural) hypothesis that $\mathsf{M}(cd) \in O(\mathsf{M}(d))$ for all $c \geq 1$.
  We also 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 let ω be a real number such that for every commutative ring R, all $n \times n$ matrices over R can be multiplied within $O(n^\omega)$ operations in R. The classical multiplication algorithm gives $\omega = 3$. Using Strassen's algorithm [29], we can take $\omega = \log_2(7) \simeq 2.81$. 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.

In the sequel, we need the following classical result on polynomial arithmetic over R. The earliest references we are aware of are [22, 4], see [31] for a detailed account. We also refer to [6] for a solution that is in the same complexity class, but where the constant hidden in the $O(\ )$ notation is actually smaller than that in [31].

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.

2 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 $P(r_0+a), \ldots, P(r_d+a)$?

A reasonable condition for this question to make sense is that all differences $r_i - r_j$, $i \neq j$, are units in R; otherwise, uniqueness of the answer might be lost. Under this assumption, using fast interpolation and fast multipoint evaluation, the problem can be answered within $O(\mathsf{M}(d)\log(d))$ operations in R. We now show that the cost reduces to $\mathsf{M}(2d)+O(d)$ operations in R, in the particular case when $r_0, \ldots, r_d$ are in arithmetic progression, so we gain a logarithmic factor.

Our solution reduces to the multiplication of two suitable polynomials of degree at most $2d$; $O(d)$ additional operations come from additional pre- and post-processing operations. As mentioned in Section 1, all operations made below on integer values actually take place in R.

The algorithm underlying Proposition 1 is given in Figure 1; we use the notation $\texttt{coeff}(Q,k)$ to denote the coefficient of degree k of a polynomial Q. We stress the fact that the polynomial P is *not* part of the input of our algorithm.

Input $P(0), \ldots, P(d)$ and a in R
Output $P(a), \ldots, P(a+d)$

– Compute

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

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

– Let

$$\widetilde{P} = \sum_{i=0}^{d} \frac{P(i)}{\delta(i,d)} X^i, \quad S = \sum_{i=0}^{2d} \frac{1}{a+i-d} X^i, \quad Q = \widetilde{P}S.$$

– Return the sequence $\Delta(a,0,d)\cdot\texttt{coeff}(Q,d), \ldots, \Delta(a,d,d)\cdot\texttt{coeff}(Q,2d)$.

Fig. 1. Shifting evaluation values

Proposition 1 *Let R be a commutative ring with unity, and $d \in \mathbb{N}$ such that $1, \ldots, d$ are units in R. Let P be in $R[X]$ of degree d, such that the sequence*

$$P(0), \ldots, P(d)$$

is known. Let a be in R, such that $a-d, \ldots, a+d$ are units in R. Then the sequence

$$P(a), \ldots, P(a+d)$$

can be computed within $\mathsf{M}(2d) + O(d)$ base ring operations, using space $O(d)$.

Proof. Our assumption on R enables 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 $P(i)/\delta(i,d)$.

First note that all $\delta(i,d), i = 0, \ldots, d$, can be computed in $O(d)$ operations in R. Indeed, computing the first value $\delta(0,d) = \prod_{j=1}^{d}(-j)$ takes d multiplications. Then for $i = 1, \ldots, d$, $\delta(i,d)$ can be deduced from $\delta(i-1,d)$ for two ring operations using the formula

$$\delta(i,d) = \frac{i}{i-d-1}\delta(i-1,d),$$

so their inductive computation requires $O(d)$ multiplications as well. Thus the sequence $\widetilde{P}_i, i = 0, \ldots, d$, can be computed in admissible time and space $O(d)$ from the input sequence $P(i)$. Accordingly, we rewrite the above formula 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 $a+k-i$:

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

Just as we introduced the sequence $\delta(i,d)$ above, we now introduce the sequence $\Delta(a,k,d)$ defined by $\Delta(a,k,d) = \prod_{j=0}^{d}(a+k-j)$. In a parallel manner, we

deduce that all $\Delta(a,k,d), k=0,\dots,d$ can be computed in time and space $O(d)$, using the formulas:

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

Let us denote $Q_k = P(a+k)/\Delta(a,k,d)$. We now show that knowing $\widetilde{P}_i, i=0,\dots,d$, we can compute $Q_k, k=0,\dots,d$ in $\mathsf{M}(2d)$ base ring operations and space $O(d)$; this is enough to conclude, by the above reasoning.

Using the coefficients $\Delta(a,k,d)$, Equation (1) reads

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

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

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

then by Equation (2), for $k=0,\dots,d$, Q_k is the coefficient of degree $k+d$ in the product $\widetilde{P}S$. This concludes the proof. □

We will conclude this section by an immediate corollary of this proposition; we first give a few comments.

- An alternative $O(\mathsf{M}(d))$ algorithm which does *not* require any inversibility hypotheses can be designed 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 [27, Theorem 3.1].
- The general case when the evaluation points form an arbitrary arithmetic progression reduces to the case treated in the above proposition. Indeed, suppose that $r_0,\dots,r_d$ form an arithmetic progression of difference δ, that $P(r_0),\dots,P(r_d)$ are known and that we want to compute the values $P(r_0+a),\dots,P(r_d+a)$, where $a\in R$ is divisible by δ. Introducing the polynomial $Q(X)=P(\delta X+r_0)$, we are under the hypotheses of the above proposition, and it suffices to determine the shifted evaluation values of Q by a/δ.
- 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 [2] 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], an 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 above algorithm.
  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 [7, 15] then show that it can be computed in time $\mathsf{M}(d) + O(d)$, but with a possible loss in space complexity. In [6], it is shown how to keep the same space complexity, at the cost of a constant increase in time complexity. Managing both requirements remains an open question, already stated in [17, Problem 6].

Corollary 1 *Let R be a commutative ring with unity, and $d \in \mathbb{N}$ such that $1, \ldots, 2d+1$ are units in R. Let P be a degree d polynomial in $R[X]$ such that the sequence*

$$P(0), \ldots, P(d)$$

is known. For any s in $\mathbb{N}$, the sequence

$$P(0), P(2^s), \ldots, P(2^s d)$$

can be computed in time $s\,\mathsf{M}(2d) + O(sd) \in O(s\,\mathsf{M}(d))$ and space $O(d)$.

Proof. For any $s \in \mathbb{N}$, let us denote by $P_s(X)$ the polynomial $P(2^sX)$. We prove by induction that all values $P_s(0), \ldots, P_s(d)$ can be computed in time $s\,\mathsf{M}(2d) + O(sd)$ and space $O(d)$, which is enough to conclude. The case $s = 0$ is obvious, as there is nothing to compute. Suppose then that $P_s(0), \ldots, P_s(d)$ can be computed in time $s\,\mathsf{M}(2d) + O(sd)$ and using $O(d)$ temporary space allocation.

Under our assumption on R, Proposition 1 shows that the values $P_s(d+1), \ldots, P_s(2d+1)$ can be computed in time $\mathsf{M}(2d) + O(d)$, using again $O(d)$ temporary space allocation. The values $P_s(0), P_s(2), \ldots, P_s(2d)$ coincide with $P_{s+1}(0), P_{s+1}(1), \ldots, P_{s+1}(d)$, so the corollary is proved. □

3 Computing one selected term of a linear sequence

In this section, we recall and improve the complexity of an algorithm due to Chudnovsky and Chudnovsky [10] for computing selected terms of linear recurrent sequences with polynomial coefficients. The results of the previous section are used as a basic subroutine for these questions.

As in the previous section, R is a commutative ring with unity. Let A be a $n \times n$ matrix of polynomials in $R[X]$. For simplicity, in what follows, we only treat the case of degree 1 polynomials, since this is what is needed in the sequel. Nevertheless, all results extend *mutatis mutandis* to arbitrary degree.

For r in R, we denote by $A(r)$ the matrix over R obtained by specializing all coefficients of A at r. In particular, for k in $\mathbb{N}$, $A(k \cdot 1_R)$ is simply denoted by

$A(k)$, following the convention used up to now. Given a vector of initial conditions $U_0 = [u_1, \ldots, u_n]^t \in R^n$ and given k in $\mathbb{N}$, we consider the question of computing the kth term of the linear sequence defined by the relation $U_i = A(i)U_{i-1}$ for $i > 0$, that is, the product

$$U_k = A(k)A(k-1)\cdots A(1)U_0.$$

For simplicity, we write

$$U_k = \left(\prod_{i=1}^{k} A(i)\right) U_0,$$

performing all successive matrix products, $i = 1, \ldots, k$, on the left side. We use this convention hereafter.

In the particular case when A is a matrix of constant polynomials, and taking only the dependence on k into account, the binary powering method gives a time complexity of order $O(\log(k))$ base ring operations.

In the general case, the naive solution consists in evaluating all matrices $A(i)$ and performing all products. With respect to k only, the complexity of this approach is of order $O(k)$ base ring operations. In [10], Chudnovsky and Chudnovsky propose an algorithm that reduces this cost to essentially $O\big(\sqrt{k}\big)$. We first recall the main lines of this algorithm; we then present some improvements in both time and space complexities.

The algorithm of Chudnovsky and Chudnovsky. The original algorithm uses baby-step / giant-step techniques, so for simplicity we assume that k is a square in $\mathbb{N}$. Let C be the $n \times n$ matrix over $R[X]$ defined by

$$C = \prod_{i=1}^{\sqrt{k}} A(X+i),$$

where $A(X+i)$ denotes the matrix A with all polynomials evaluated at $X+i$. By assumption on A, the entries of C have degree at most $\sqrt{k}$. For r in R, we denote by $C(r)$ the matrix C with all entries evaluated at r. Then the requested output U_k can be obtained by the equation

$$U_k = \left(\prod_{j=0}^{\sqrt{k}-1} C(j\sqrt{k})\right) U_0. \tag{3}$$

Here are the main steps of the algorithm underlying Equation (3), originally due to [10].

Baby steps. The "baby steps" part of the algorithm consists in computing the polynomial matrix C. In [10], this is done within $O(n^\omega \mathsf{M}(\sqrt{k}))$ base ring operations, as products of polynomial matrices with entries of degree $O(\sqrt{k})$ are required.

Giant steps. In the second part the matrix C is evaluated on the arithmetic progression $0, \sqrt{k}, 2\sqrt{k}, \ldots, (\sqrt{k}-1)\sqrt{k}$ and the value of U_k is obtained using Equation (3). Using fast evaluation techniques, all evaluations are done within $O(n^2\,\mathsf{M}(\sqrt{k})\log(k))$ base ring operations, while performing the $\sqrt{k}$ successive matrix-vector products in Equation (3) adds a negligible cost of $O(n^2\sqrt{k})$ operations in R.

Summing all the above costs gives an overall complexity bound of

$$O\big(n^\omega \mathsf{M}\big(\sqrt{k}\big) + n^2\,\mathsf{M}\big(\sqrt{k}\big)\log(k)\big)$$

base ring operations for computing a selected term of a linear sequence. Due to the use of fast evaluation algorithms in degree $\sqrt{k}$, the space complexity is $O(n^2\sqrt{k} + \sqrt{k}\log(k))$.

In the particular case when A is the 1×1 matrix $[X]$, the question reduces to the computation of $\prod_{j=1}^{k} j$ in the ring R. For this specific problem, note that the ideas presented above were already used in [23, 30], for the purpose of factoring integers.

Avoiding multiplications of polynomial matrices. In what follows, we show how to avoid the multiplication of polynomial matrices, and reduce the cost of the above algorithm to $O(n^\omega\sqrt{k} + n^2\,\mathsf{M}(\sqrt{k})\log(k))$ base ring operations, storing only $O(n^2\sqrt{k})$ elements of R.

Our improvements are obtained through a modification of the baby steps phase; the underlying idea is to work with the *values* taken by the polynomial matrices instead of their representation on the monomial basis. This idea is encapsulated in the following proposition.

Proposition 2 *Let A be a $n\times n$ matrix with entries in $R[X]$, of degree at most 1. Let $N \geq 1$ be an integer and let C be the $n \times n$ matrix over $R[X]$ defined by*

$$C = \prod_{i=1}^{N} A(X+i).$$

Then one can compute all scalar matrices $C(0), C(1), \ldots, C(N)$ within $O(n^\omega N)$ operations in R and with a memory requirement of $O(n^2 N)$ elements in R.

Proof. We first compute the scalar matrices $[A(1), A(2), \ldots, A(2N)]$. Since all entries of A are linear in X, the complexity of this preliminary step is $O(n^2 N)$, both in time and space.

Then, we construct the matrices $(C'_j)_{0\leq j\leq N}$ and $(C''_j)_{0\leq j\leq N}$, which are defined as follows: we let C'_0 and C''_0 equal the identity matrix I_n and we recursively define

$$\begin{aligned} C'_j &= \quad A(N+j)C'_{j-1} \quad \text{for } 1\leq j\leq N,\\ C''_j &= C''_{j-1}A(N-j+1) \text{ for } 1\leq j\leq N. \end{aligned}$$

Explicitly, for $0 \leq j \leq N$, we have

$$C'_j = A(N+j)\cdots A(N+1)$$

and

$$C''_j = A(N)\cdots A(N-j+1),$$

thus

$$C''_{N-j} = A(N)\cdots A(j+1).$$

Computing all the scalar matrices (C'_j) and (C''_j) requires $2N$ matrix multiplications with entries in R; their cost is bounded by $O(n^\omega N)$ in time and by $O(n^2 N)$ in space. Lastly, the formula

$$C(j) = A(N+j)\cdots A(N+1)A(N)\cdots A(j+1) = C'_j C''_{N-j}, \quad 0 \leq j \leq N$$

enables to recover $C(0), C(1), \ldots, C(N)$ in time $O(n^\omega N)$ and space $O(n^2 N)$. □

From this proposition, we deduce the following corollary, which shows how to compute the scalar matrices used in the giant steps.

Corollary 2 *Let A and C be polynomial matrices as in Proposition 2. If the elements $1, \ldots, 2N+1$ are units in R, then for any integer $s \geq 1$, the sequence*

$$C(0), C(2^s), \ldots, C(2^s(N-1))$$

can be computed using $O(n^\omega N + n^2 s\,\mathsf{M}(N))$ operations in R and $O(n^2 N)$ memory space.

Proof. This is an immediate consequence of Proposition 2 and Corollary 1. □

The above corollary enables us to perform the "giant steps" phase of Chudnovsky and Chudnovsky's algorithm in the special case when $N = 2^s$; this yields the 4^sth term in the recurrent sequence. Using this intermediate result, the following theorem shows how to compute the kth term, for arbitrary k, using the 4-adic expansion of k.

Theorem 1 *Let A 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*

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

Let $k > 0$ be an integer and suppose that $1, \ldots, 2\lceil\sqrt{k}\rceil + 1$ are units in R. Then the vector U_k can be computed within $O\big(n^\omega\sqrt{k} + n^2\,\mathsf{M}\big(\sqrt{k}\big)\log(k)\big)$ operations in R and using memory space $O(n^2\sqrt{k})$.

The proof of Theorem 1 is divided in two steps. We begin by proving the proposition in the particular case when k is a power of 4, then we treat the general case.

The case k is a power of 4. Let us suppose that $N = 2^s$ and $k = N^2$, so that $k = 4^s$. With this choice of k, Corollary 2 shows that the values $C(0), C(N), \ldots, C((N-1)N)$ can be computed within the required time and space complexities. Then we go on to the giant step phase described at the beginning of the section, and summarized in Equation (3). It consists in performing $\sqrt{k}$ successive matrix-vector products, which has a cost in both time and space of $O(n^2\sqrt{k})$.

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

$$V_{t+1} = A(\lceil k \rceil^t + 4^t k_t) \cdots A(\lceil k \rceil^t + 1) V_t. \tag{4}$$

It is easy to verify that $V_{s+1} = U_k$. 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 k_t is zero, we have nothing to do. Otherwise, we let $V_{t+1}^{(0)} = V_t$, and, for $1 \le j \le k_t$, we let $A^{(j)}(X) = A\left(X + \lceil k \rceil^t + 4^t(j-1)\right)$. Then we define $V_{t+1}^{(j)}$ by

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

By Equation (4), we have $V_{t+1}^{k_t} = V_{t+1}$. Thus, passing from V_t to V_{t+1} amounts to computing k_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 k_t are bounded by 3, the total cost of the general case is thus

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

Using the fact that $2^s \le \sqrt{k} \le 2^{s+1}$ and the assumptions on the function M, we easily deduce that the whole complexity fits into the bound $O\big(n^\omega\sqrt{k} + n^2\,\mathsf{M}\big(\sqrt{k}\big)\log(k)\big)$, as claimed. Similar considerations also yield the bound concerning the memory requirements. This concludes the proof of Theorem 1.

Comments. The question of a lower time bound for computing U_k is still open. The simpler question of reducing the cost to $O\big(n^\omega\sqrt{k} + n^2\,\mathsf{M}\big(\sqrt{k}\big)\big)$ base ring operations, that is gaining a logarithmic factor, already raises challenging problems.

As the above paragraphs reveal, this improvement could be obtained by answering the following question: Let P be a polynomial of degree d in $R[X]$. Given r in R, how fast can we compute $P(0), P(r), \ldots, P(rd)$ from the data of $P(0), P(1), \ldots, P(d)$? A complexity of order $O(\mathsf{M}(d))$ would immediately give the improved bound mentioned above. We leave it as an open question.

4 The Cartier-Manin operator on hyperelliptic curves

We finally show how to apply the above results to the computation of the Cartier-Manin operator, and start by reviewing some known facts on this operator.

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:

Definition 1 *Let h_k be the coefficient of degree 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*

$$H = (h_{ip-j})_{1 \le i,j \le g}.$$

This matrix was introduced in [16]; in a suitable basis, it represents the operator on differential forms that was introduced by Cartier in [9]. Manin then showed in [20] 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 [33] provides a complete survey about those facts; they can be summarized by the following theorem:

Theorem 2 (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 let $\chi(t)$ be 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-step / giant-step algorithm, it can lead to a full point-counting algorithm, in particular for genus 2 curves, as was demonstrated in [13, 21].

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 $O(\mathsf{M}(gp))$ base field operations, whence a time complexity within $O(\mathsf{M}(p))$, if g is kept constant. In what follows, regarding the dependence in p only, we show how to obtain a complexity of $O(\mathsf{M}(\sqrt{p})\log(p))$ base field operations, using the results of the previous sections.

We will make the assumption that the constant term of f is not zero. Note that if it is zero, 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 required degrees are slightly less.

Furthermore, for technical reasons, we assume that $g < p$. This is not a true restriction since for $g \geq p$, all the coefficients of $f^{(p-1)/2}$ up to degree $g(p-1)$ are needed to fill in the matrix H.

Introduction of a linear recurrent sequence. In [11], 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 above differential equation, for all k in $\mathbb{Z}$, we deduce that

$$\sum_{i=0}^{2g+1} \left(k + 1 - \frac{(p+1)i}{2} \right) f_i \, h_{k+1-i} \; = \; 0.$$

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

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

The initial vector $U_0 = [0, \ldots, 0, f_0^{(p-1)/2}]^t$ can be computed using binary powering techniques in $O(\log(p))$ base field 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-j} gives the coefficients h_{ip-j} for $j = 1, \ldots, g$ that form the ith row of the Hasse-Witt matrix of $\mathcal{C}$.

Yet, Theorem 1 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 will compute separately, first $V_{p-1}, V_{2p-1}, \dots V_{gp-1}$ and then the denominators $f_0^{p-1}(p-1)!, \dots, f_0^{gp-1}(gp-1)!$.

To this effect, we proceed iteratively. Let us for instance detail the computation of the sequence $V_{p-1}, V_{2p-1}, \dots V_{gp-1}$. Knowing V_0, we compute V_{p-1} using Theorem 1. Then we shift all entries of B by p, so another application of Theorem 1 yields V_{2p-1}. Iterating g times, we obtain $V_{p-1}, V_{2p-1}, \dots V_{gp-1}$ as requested; the same techniques are used to compute $f_0^{p-1}(p-1)!, \dots, f_0^{gp-1}(gp-1)!$. Then the vectors $U_{p-1}, U_{2p-1}, \dots U_{gp-1}$ are deduced from

$$U_k = \frac{1}{f_0^k k!} V_k.$$

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 residue class field is $\mathbb{F}_{p^d}$. The ring of integers of K will be denoted by O_K; 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 above paragraph 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.

Using the iteration described above, we separately compute the values in K of the vectors V_{ip-1} and the denominators $f_0^{ip-1}(ip-1)!$, for $i = 1, \dots, g$. To this effect, we apply g times the result given in Theorem 1; this requires to perform

$$O\left(g^{\omega+1}\sqrt{p} + g^3 \mathsf{M}(\sqrt{p}) \log(p)\right),$$

operations in K and to store $O(g^2\sqrt{p})$ 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 $i = 1, \dots, g$, we use Theorem 1. This requires that all integers up to $2\lceil\sqrt{p}\rceil + 1$ are invertible, which holds as soon as $p \geq 11$.

Then, for all $i = 1, \dots, g$, 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. With our assumption that $g < p$, we have $i < p$ and 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. Storing an element of O_K/p^gO_K requires $O(dg\log(p))$ bits, and multiplying two such elements can be done with $O(\mathsf{M}(dg\log(p)))$ bit-operations. From the results of Section 3, we then deduce the following theorem on the complexity of computing the Hasse-Witt matrix.

Theorem 3 *Let p a prime, $d \geq 1$ 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 $g < p$, one can compute the Hasse-Witt matrix of $\mathcal{C}$ with a complexity of*

$$O\left(\left(g^{\omega+1}\sqrt{p} + g^3\mathsf{M}(\sqrt{p})\log(p)\right)\ \mathsf{M}(dg\log(p))\right)$$

bit-operations and $O\left(dg^3\sqrt{p}\ \log(p)\right)$ 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 [33, Corollary 2.3]. However, as stated in Theorem 2, 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. Thus, we finally 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 the computation of $H_{\pi,i+1}$ 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 $H_\pi = H_{\pi,\log_2(d)}$. Hence the cost of computing H_π is

$$O\left(dg^2\log(p) + g^\omega\log(d)\right)$$

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 [19] 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, we get a time complexity for computing $\chi(t) \mod p$ in

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

Case of large genus. In case of large genus, the algorithm of Theorem 1 is 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. The cost in bit-operations of Theorem 3 is at least $g^4\sqrt{p}d$ whereas the cost of the naive algorithm is linear in gpd. If $g > p^{1/6}$, then $g^4\sqrt{p} > gp$, and therefore the naive algorithm is faster.

5 Point-counting numerical example

We have implemented our algorithm using Shoup's NTL C++ library [26]. NTL does not provide any arithmetic of local fields or rings, but allows to work in finite extensions of rings of the form $\mathbb{Z}/p^g\mathbb{Z}$, as long as no division by p occur; the divisions by p are well isolated in the algorithm, so we could handle them separately. Furthermore, NTL multiplies polynomials defined over this kind of structure using an asymptotically fast FFT-based algorithm.

To illustrate that our method can be used as a tool in point-counting algorithms, we have computed the Zeta function of a (randomly chosen) genus 2 curve defined over $\mathbb{F}_{p^3}$, with $p = 2^{32} - 5$. Such a Jacobian has therefore about 2^{192} elements and should be suitable for cryptographic use if the group order has a large prime factor. Note that previous computations were limited to p of order 2^{23} [21].

The characteristic polynomial χ of the Frobenius endomorphism was computed modulo p in 3 hours and 41 minutes, using 1 GB of memory, on an AMD Athlon MP 2200+. Then we used the Schoof-like algorithms of [13] and [14] to compute χ modulo $128 \times 9 \times 5 \times 7$, and finally we used the modified baby-step / giant-step algorithm of [21] to finish the computation. These other parts were implemented in Magma [5] and were performed in about 15 days of computation on an Alpha EV67 at 667 MHz. We stress that this computation was meant as an illustration of the possible use of our method, so little time was spent optimizing our code. In particular, the Schoof-like part and the final baby-step / giant-step computations are done using a generic code that is not optimized for extension fields.

Numerical data. The irreducible polynomial $P(t)$ that was used to define $\mathbb{F}_{p^3}$ as $\mathbb{F}_p[t]/(P(t))$ is

$$t^3 + 1346614179t^2 + 3515519304t + 3426487663.$$

The curve $\mathcal{C}$ has equation $y^2 = f(x)$ where f is given by

$$\begin{aligned} f(x) = x^5 &+ (2697017539t^2 + 1482222818t + 3214703725)x^3 + \\ &(676673546t^2 + 3607548185t + 1833957986)x^2 + \\ &(1596634951t^2 + 3203023469t + 2440208439)x + \\ &2994361233t^2 + 3327339023t + 862341251. \end{aligned}$$

Then the polynomial characteristic $\chi(T)$ of the Frobenius endomorphism is given by $T^4 - s_1T^3 + s_2T^2 - p^3s_1T + p^6$, where

$$s_1 = 332906835893875, \quad s_2 = 142011235215638946167187570235.$$

The group order of the Jacobian is then

$$\begin{aligned} &627710169154160539591778508077182588386018946581362599397 7 \\ &= 3^3 \times 13 \times 67 \times 639679 \times 417268068727536370810010172344236025455933953139. \end{aligned}$$

This number has a large prime factor of size 2^{158}, therefore that curve is cryptographically secure.

Measure of the complexity in *p*. To check the practical asymptotic behaviour of our algorithm, we ran our implementation on a genus 2 curve defined over $\mathbb{F}_{p^3}$ with $p = 2^{34} - 41$. We performed only the Cartier-Manin step, and not the full point-counting algorithm. As the characteristic is about 4 times larger than in the previous example, a complexity linear in $\sqrt{p}$ means a runtime multiplied by about 2. On the same computer, the runtime is 8 hours and 48 minutes. Hence the ratio of the runtimes is about 2.39. The defect of linearity can be explained by taking into account the logarithmic factors. Assuming that $\mathsf{M}(n)$ is $O(n\log(n)\log(\log(n)))$, and neglecting the multi-logarithmic factors, the complexity announced in Theorem 3 is in $O(\sqrt{p}(\log(p))^3)$. With this estimate, the expected ratio between the runtimes becomes about 2.40, that is very close to the measure. This validates our analysis.

6 Conclusion

In this paper, we have presented an improvement of an algorithm by Chudnovsky and Chudnovsky to compute selected terms in a linear sequence with polynomial coefficients. This algorithm is then applied to the computation of the Cartier-Manin operator of hyperelliptic curves, thus leading to improvements in the point-counting problems that occur in cryptography.

This strategy extends readily to curves of the form $y^r = f(x)$ with $r > 2$, for which the Hasse-Witt matrix has a similar form. For more general curves, Mike Zieve pointed to us the work of Stöhr and Voloch [28] that gives formulas that still fit in our context in some cases.

Finally, Mike Zieve pointed out to us the work of Wan [32] that relates Niederreiter's polynomial factorization algorithm to the computation of the Cartier-Manin operator of some variety. The link with our work is not immediate, as that variety has dimension zero. Nevertheless, this remains intriguing, especially if we think of Pollard-Strassen's integer factoring algorithm as a particular case of Chudnovsky and Chudnovsky's algorithm.

Acknowledgements

Part of our numerical experiments were done with the computer of the Medicis center [1]. We thank Takakazu Satoh and Mike Zieve for their interest in our work and their insightful comments and also Bruno Salvy for his numerous helpful suggestions.

References

1. Medicis. http://www.medicis.polytechnique.fr/.
2. A. V. Aho, K. Steiglitz, and J. D. Ullman. Evaluating polynomials at fixed sets of points. *SIAM J. Comput.*, 4(4):533–539, 1975.
3. D. Bailey and C. Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In *Advances in Cryptology – CRYPTO '98*, volume 1462 of *LNCS*, pages 472–485. Springer–Verlag, 1998.
4. A. Borodin and R. T. Moenck. Fast modular transforms. *Comput. System Sci.*, 8(3):366–386, 1974.
5. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symb. Comp.*, 24(3-4):235–265, 1997. See also `http://www.maths.usyd.edu.au:8000/u/magma/`.
6. A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *Proceedings of ISSAC'03*, pages 37–44. ACM Press, 2003.
7. P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren Math. Wiss.* Springer–Verlag, 1997.
8. D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.
9. P. Cartier. Une nouvelle opération sur les formes différentielles. *C. R. Acad. Sci. Paris*, 244:426–428, 1957.
10. 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.
11. P. Flajolet and B. Salvy. The SIGSAM challenges: Symbolic asymptotics in practice. *SIGSAM Bull.*, 31(4):36–47, 1997.
12. P. Gaudry and N. Gürel. Counting points in medium characteristic using Kedlaya's algorithm. To appear in *Experiment. Math.*
13. P. Gaudry and R. Harley. Counting points on hyperelliptic curves over finite fields. In *ANTS-IV*, volume 1838 of *LNCS*, pages 313–332. Springer–Verlag, 2000.
14. P. Gaudry and É. Schost. Cardinality of a genus 2 hyperelliptic curve over GF($5 \cdot 10^{24} + 41$). e-mail to the `NMBRTHRY` mailing list. Sept. 2002.
15. G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm, I. Speeding up the division and square root of power series. Preprint.
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. Kaltofen, R. M. Corless, and D. J. Jeffrey. Challenges of symbolic computation: my favorite open problems. *J. Symb. Comp.*, 29(6):891–919, 2000.
18. K. Kedlaya. Countimg points on hyperelliptic curves using Monsky-Washnitzer. *J. Ramanujan Math. Soc.*, 16:323–338, 2001.

19. W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theor. Comput. Sci.*, 36(2-3):309–317, 1985.
20. J. I. Manin. The Hasse-Witt matrix of an algebraic curve. *Trans. Amer. Math. Soc.*, 45:245–264, 1965.
21. 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 *LNCS*, pages 461–474. Springer–Verlag, 2002.
22. 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.
23. J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.
24. A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica*, 7:395–398, 1977.
25. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
26. V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net`.
27. V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *Proceedings of ISSAC'91*, pages 14–21. ACM Press, 1991.
28. K.-O. Stöhr and J. Voloch. A formula for the Cartier operator on plane algebraic curves. *J. Reine Angew. Math.*, 377:49–64, 1987.
29. V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.
30. V. Strassen. Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, 78(1):1–8, 1976/77.
31. J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.
32. D. Wan. Computing zeta functions over finite fields. *Contemp. Math.*, 225:131–141, 1999.
33. N. Yui. On the Jacobian varietes of hyperelliptic curves over fields of characteristic $p > 2$. *J. Algebra*, 52:378–410, 1978.