Polynomial evaluation and interpolation on special sets of points

Alin Bostan and Éric Schost

Laboratoire STIX, École polytechnique, 91128 Palaiseau, France

Abstract

We give complexity estimates for the problems of evaluation and interpolation on various polynomial bases. We focus on the particular cases when the sample points form an arithmetic or a geometric sequence, and discuss applications, respectively to computations with differential operators and to polynomial matrix multiplication.

1 Introduction

Let k be a field and let $\mathbf{x} = x_0, \ldots, x_{n-1}$ be n pairwise distinct points in k. Given arbitrary values $\mathbf{v} = v_0, \ldots, v_{n-1}$ in k, there exists a unique polynomial F in $k[x]$ of degree less than n such that $F(x_i) = v_i$, for $i = 0, \ldots, n-1$. Having fixed a basis of the vector space of polynomials of degree at most $n-1$, interpolation and evaluation questions consist in computing the coefficients of F on this basis from the values $\mathbf{v}$, and conversely.

Well-known problems are that of interpolation and evaluation in the *monomial basis* $1, x, \ldots, x^{n-1}$:

- Given $x_0, \ldots, x_{n-1}$ and $v_0, \ldots, v_{n-1}$, *monomial interpolation* consists in determining the unique coefficients $f_0, \ldots, f_{n-1}$ such that the polynomial $F = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$ satisfies $F(x_i) = v_i$, for $i = 0, \ldots, n-1$.
- Given $x_0, \ldots, x_{n-1}$ and $f_0, \ldots, f_{n-1}$, *monomial evaluation* consists in computing the values $v_0 = F(x_0), \ldots, v_{n-1} = F(x_{n-1})$, where F is the polynomial $f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$.

Email addresses: Alin.Bostan@stix.polytechnique.fr (Alin Bostan), Eric.Schost@stix.polytechnique.fr (Éric Schost).

The *Newton basis* associated to the points $x_0, \ldots, x_{n-1}$ provides with an alternative basis of degree $n-1$ polynomials, which is defined as

$$1, (x - x_0), (x - x_0)(x - x_1), \ldots, (x - x_0) \cdots (x - x_{n-2}).$$

An important particular case is the *falling factorial basis*

$$1,\ x^{\underline{1}} = x,\ x^{\underline{2}} = x(x-1),\ x^{\underline{3}} = x(x-1)(x-2), \ldots,$$

which is used in many algorithms for symbolic summation, see for instance (**???**). Accordingly, Newton interpolation and Newton evaluation are defined as the problems of interpolation and evaluation with respect to the Newton basis:

- Given $x_0, \ldots, x_{n-1}$ and $v_0, \ldots, v_{n-1}$, *Newton interpolation* consists in determining the unique coefficients $f_0, \ldots, f_{n-1}$ such that the polynomial

$$F = f_0 + f_1(x - x_0) + f_2(x - x_0)(x - x_1) + \cdots + f_{n-1}(x - x_0) \cdots (x - x_{n-2}) \quad (1)$$

  satisfies $F(x_i) = v_i$, for $i = 0, \ldots, n-1$.
- Given $x_0, \ldots, x_{n-1}$ and $f_0, \ldots, f_{n-1}$, *Newton evaluation* consists in computing the values $v_0 = F(x_0), \ldots, v_{n-1} = F(x_{n-1})$, where F is the polynomial given by Equation (1).

Fast algorithms for evaluation and interpolation in the monomial basis were discovered in the seventies and have complexity in $O(\mathsf{M}(n)\log(n))$, where $\mathsf{M}(n)$ denotes the complexity of multiplying univariate polynomials of degree less than n. Using FFT-based multiplication algorithms, $\mathsf{M}(n)$ can be taken in $O(n\log(n)\log(\log(n)))$, so the complexity above is nearly optimal, up to logarithmic factors (note that naive algorithms are quadratic in n). These fast algorithms are nowadays classical topics covered by most of the computer algebra textbooks (**???**) and their practical relevance is recognized.

In contrast, fast algorithms for Newton evaluation and interpolation are quite recent, despite a potentially vast field of applications. The standard algorithms are based on divided differences and have a complexity quadratic in n (**?**). Yet, using a divide-and-conquer approach, the complexity of Newton evaluation and interpolation becomes essentially linear in n: indeed, the algorithms suggested in (**?**, Ex. 15, p. 67) have complexity in $O(\mathsf{M}(n)\log(n))$.

Such fast algorithms for Newton evaluation and interpolation rely on two additional tasks: the basis conversion between the monomial basis and the Newton basis, and conversely. Algorithms realizing these tasks are detailed for instance in (**?**, Theorems 2.4 and 2.5); they also have a complexity in $O(\mathsf{M}(n)\log(n))$.

Remark that monomial as well as Newton evaluation and interpolation can

also be regarded as base change problems, between monomial or Newton basis on one hand and Lagrange basis

$$\frac{\prod_{j\neq 0}(x-x_j)}{\prod_{j\neq 0}(x_0-x_j)}, \frac{\prod_{j\neq 1}(x-x_j)}{\prod_{j\neq 1}(x_1-x_j)}, \ldots, \frac{\prod_{j\neq n-1}(x-x_j)}{\prod_{j\neq n-1}(x_{n-1}-x_j)}$$

associated to the points $x_0, \ldots, x_{n-1}$, on the other hand.

Our contribution. The core of this paper is the study of the three operations mentioned up to now:

(1) conversions between Newton and monomial bases,
(2) monomial evaluation and interpolation,
(3) Newton evaluation and interpolation.

Our first goal is to propose improved algorithms for some of these operations: specifically, we describe a faster conversion algorithm, from which we deduce an improved Newton interpolation algorithm. When the sample points bear no special structure, our algorithms still have an asymptotic complexity belonging to the class $O(\mathsf{M}(n)\log(n))$, but they are faster than the previous ones by a constant factor. To establish comparisons, and for the sake of completeness, we thus will detail the constants hidden behind the Big-Oh notation in the complexity estimates, for our algorithms as well as for one of **?**.

Our second objective is to obtain better algorithms for special cases of evaluation points $\mathbf{x}$. Indeed, it is well known that the divided difference formulas used in Newton interpolation simplify when the points form an *arithmetic* or a *geometric* progression; we will show how to obtain improved complexity estimates based on such simplifications. We also discuss applications to computations with differential operators and polynomial matrix multiplication.

Table 1 summarizes the best results known to us on the three questions mentioned above; we now review its columns in turn and detail our contributions. In what follows, all results of type $O(\mathsf{M}(n)\log(n))$ are valid when n is a power of 2. The results for the arithmetic progression case require that the base field has characteristic 0 or larger than n.

The general case. The first column gives estimates for arbitrary sample points; we call this case the *general case*. In this situation, conversion algorithms and monomial evaluation / interpolation algorithms are designed first, and algorithms for Newton evaluation and interpolation are deduced by composition.

Question	general case
Newton to monomial basis (**NtoM**)	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$
Monomial to Newton basis (**MtoN**)	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$
Monomial evaluation (**MtoV**)	$3/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$
Monomial interpolation (**VtoM**)	$5/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$
Newton evaluation (**NtoV**)	$2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$
Newton interpolation (**VtoN**)	$3\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$

Question	arithmetic case	geometric case
NtoM	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$	$\mathsf{M}(n) + O(n)$
MtoN	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$	$\mathsf{M}(n) + O(n)$
MtoV	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$	$2\,\mathsf{M}(n) + O(n)$
VtoM	$\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$	$2\,\mathsf{M}(n) + O(n)$
NtoV	$\mathsf{M}(n) + O(n)$	$\mathsf{M}(n) + O(n)$
VtoN	$\mathsf{M}(n) + O(n)$	$\mathsf{M}(n) + O(n)$

Table 1
Complexity results on conversions between Newton and monomial bases, monomial evaluation and interpolation, Newton evaluation and interpolation.

The conversion algorithm from the Newton to the monomial basis (first line in the table) is from **?**, who also gives its bit complexity when the base field is $\mathbb{Q}$. As to the conversion from the monomial to the Newton basis (second line in the table), our algorithm is new, to the best of our knowledge. It relies on the so-called *transposition principle*, which finds its origins in Tellegen's theorem on electrical networks (**????**). **?** also presents an algorithm for this task, but its complexity is higher by a constant factor.

The results on monomial evaluation and interpolation (third and fourth lines)

appeared in (**?**) and improve the classical results by (**???**) and **?**.

The last two operations (fifth and sixth line of the upper table) are Newton evaluation and interpolation; the algorithms for these tasks are easily deduced from those mentioned before (lines 1–4), by composition. Note however that the constants do not add up, due to some shared precomputations.

Recall that these complexity results were already known to lie in the class $O(\mathsf{M}(n)\log(n))$, see for instance **?**; the precise estimate in the fifth line is obtained by combining algorithms of (**?**) and (**?**), whereas that in the last line relies on our improved conversion algorithm.

The arithmetic progression case. In the arithmetic progression case, a sharp complexity statement was given in (**?**, Th. 3.2 and Th. 3.4), which proves that Newton evaluation and interpolation at an arithmetic progression can be done within $\mathsf{M}(n)+O(n)$ operations. That article also analyzes the bit complexity when the base field is $\mathbb{Q}$.

In the arithmetic progression case, the basis conversion algorithms developed in the general case remain unchanged. Using Gerhard's result and these conversion algorithms, we then deduce new, faster algorithms for monomial evaluation and interpolation on an arithmetic progression.

We apply these algorithms to conversions between the monomial and the falling factorial bases. This yields fast algorithms for computing Stirling numbers, which in turn are the basis for fast computation with linear differential operators. Also, we discuss the transpose of the algorithm of **?**, which is shown to be closely related to an algorithm of **?** for polynomial shift.

The geometric case. In the geometric progression case, we show that the complexities of Newton evaluation and interpolation drop to $\mathsf{M}(n)+O(n)$ as well. The improvements are obtained by (mainly) translating into equalities of generating series the formulas for divided q-differences, similarly to what is done by **?** for the arithmetic case. By considering the transposed problems, we deduce that the conversions between the Newton and the monomial bases can be done with the same asymptotic complexity of $\mathsf{M}(n)+O(n)$ operations in the base field.

These results have consequences for evaluation and interpolation in the monomial basis. It is known (**???**) that evaluating a polynomial of degree less than n on n points in geometric progression has cost $2\,\mathsf{M}(n)+O(n)$ (the algorithm given by **?** actually has complexity more than $2\,\mathsf{M}(n)+O(n)$, but using the middle product operation of **?** yields the announced complexity bound). An

analogue result for the inverse problem – that is, interpolation on the monomial basis at points in geometric progression – was previously not known. Using the Newton basis for intermediate computations, we show that this can also be done using $2\,\mathsf{M}(n) + O(n)$ operations.

Thus, this allows to exhibit special sequences of points, lying in the base field, for which both evaluation and interpolation are cheaper by a logarithmic factor than in the general case. Many algorithms using evaluation and interpolation for intermediate computations can benefit from this. We exemplify this by improving the known complexity results for polynomial matrix multiplication.

Organization of the paper. In Section 2 we recall basic tools for our subsequent algorithms, the construction of the *subproduct tree* associated to the sample points and the notion of transposed algorithms, and notably transposed multiplication.

In Section 3 we recall or improve known algorithms for the three operations mentioned up to now, evaluation and interpolation in the monomial and Newton bases, as well as base change algorithms, in the general case of arbitrary sample points.

In Section 4, we focus on the case when the sample points form an arithmetic sequence, and present an application to computations with differential operators. Section 5 is devoted to the special case of evaluation points in a geometric progression; we conclude this section by an application to polynomial matrix multiplication.

Technical assumptions. We suppose that the *multiplication time* function M fulfills 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 $\mathsf{M}(d/2) \leq 1/2\,\mathsf{M}(d)$ holds for all $d \geq 1$. We also make the hypothesis that $\mathsf{M}(cd)$ is in $O(\mathsf{M}(d))$, for all $c > 0$. The basic examples we have in mind are *classical* multiplication, for which $\mathsf{M}(n) \in O(n^2)$, Karatsuba's multiplication (**?**) with $\mathsf{M}(n) \in O(n^{1.59})$ and the FFT-based multiplication (**???**), which have $\mathsf{M}(n) \in O(n \log(n) \log(\log(n)))$. Our references for matters related to polynomial arithmetic are the books (**???**).

When designing transposed algorithm, we will impose additional assumptions on the polynomial multiplication algorithm; these assumptions are described in Subsection 2.2.

2 Preliminaries

In this section, we introduce two basic tools: the *subproduct tree* $\mathcal{T}$ associated to the points $x_0, \ldots, x_{n-1}$ and the notion of *transposed algorithm*.

2.1 The subproduct tree

In what follows, we suppose that n is a power of 2. The subproduct tree $\mathcal{T}$ associated to $x_0, \ldots, x_{n-1}$ is then a complete binary tree, all whose nodes contain polynomials in $k[x]$. Let $n = 2^m$; then $\mathcal{T}$ is defined as follows:

- If $m = 0$, then $\mathcal{T}$ reduces to a single node, containing the polynomial $x - x_0$.
- If $m > 0$, let $\mathcal{T}_0$ and $\mathcal{T}_1$ be the trees associated to $x_0, \ldots, x_{2^{m-1}-1}$ and $x_{2^{m-1}}, \ldots, x_{2^m-1}$ respectively. Let M_0 and M_1 be the polynomials at the roots of $\mathcal{T}_0$ and $\mathcal{T}_1$. Then $\mathcal{T}$ is the tree whose root contains the product $M_0 M_1$ and whose children are $\mathcal{T}_0$ and $\mathcal{T}_1$.

Alternately, one can represent the subproduct tree $\mathcal{T}$ as a 2-dimensional array $T_{i,j}$, with $0 \leq i \leq m$, $0 \leq j \leq 2^{m-i} - 1$. Then

$$T_{i,j} = \prod_{\ell=2^i j}^{2^i(j+1)-1} (x - x_\ell).$$

For instance, if $m = 2$ (and thus $n = 4$), the tree associated to x_0, x_1, x_2, x_3 is given by

$$\begin{gathered} T_{0,0} = x - x_0, \quad T_{0,1} = x - x_1, \quad T_{0,2} = x - x_2, \quad T_{0,3} = x - x_3, \\ T_{1,0} = (x - x_0)(x - x_1), \quad T_{1,1} = (x - x_2)(x - x_3), \\ T_{2,0} = (x - x_0)(x - x_1)(x - x_2)(x - x_3). \end{gathered}$$

The following result was first pointed out by **?**, see also (**?**).

Proposition 1 *The subproduct tree associated to $x_0, \ldots, x_{n-1}$ can be computed within $1/2\,\mathsf{M}(n)\log(n) + O(n\log(n))$ base field operations.*

Since in what follows a particular attention is payed to special sets of points $\mathbf{x}$, one may wonder whether the construction of the subproduct tree can be speeded-up in such *structured* situations. If the points $\mathbf{x}$ form a geometric sequence, we show that an acceleration (by a logarithmic factor) can indeed be obtained. This result is stated for completeness in Lemma 1 below; however, we will not make use of it in the sequel, since our fast algorithms in

Section 5 for evaluation and interpolation on geometric sequences do not rely on the use of the subproduct trees. In contrast, in the case of points in an arithmetic progression, we are not able to obtain a similar improvement upon Proposition 1. Note that a gain in the arithmetic case would be of real interest, as some of our algorithms in Section 4 share the construction of $\mathcal{T}$ as a precomputation.

Lemma 1 *Let k be a field and let n be a power of 2. Let $x_i = q^i, i = 0, \ldots, n-1$ be a geometric progression in k. Then, the nodes of the subproduct tree $\mathcal{T}$ associated to $\mathbf{x}$ can be computed within $\mathsf{M}(n) + O(n \log(n))$ operations in k.*

Proof. The fact that the points $\mathbf{x}$ form a geometric progression implies that, at every level i, the nodes $T_{i,j}$ can be deduced from one another by performing a homothety. Thus, our strategy is to determine the left-side nodes $T_{i,0}$ first, then to obtain all the other nodes by polynomial homotheties.

The left-side nodes $T_{i,0}$, for $0 \leq i \leq m$, have degree 2^i and can be determined (together with the nodes $T_{i,1}$) by a procedure based on the equalities

$$T_{i,1}(x) = T_{i,0}(x/q^{2^i}) \cdot q^{4^i}, \;\; T_{i+1,0}(x) = T_{i,0}(x) \cdot T_{i,1}(x), \quad 0 \leq i \leq m-1$$

within $\sum_{i=0}^{m-1} \Big(\mathsf{M}(2^i) + O(2^i)\Big) = \mathsf{M}(n) + O(n)$ operations in k. Then, starting from $T_{i,0}$, the remaining desired nodes can be determined using the equality

$$T_{i,j}(x) \;=\; T_{i,0}\Big(x/q^{2^i j}\Big)\, q^{4^i j}, \quad \text{for} \quad 1 \leq i \leq m-2, \;\; 2 \leq j \leq 2^{m-i} - 1,$$

for a total cost of at most $\sum_{i=1}^{m-2} 2^{m-i}\, O(2^i) = O(n \log(n))$ operations in k. This finishes the proof of the lemma. □

Remarks. In the general case, the *top polynomial* $T_{m,0} = (x - x_0)(x - x_1) \cdots (x - x_{n-1})$ (or, equivalently, all the elementary symmetric functions of $x_0, \ldots, x_{n-1}$) can be computed in $1/2\, \mathsf{M}(n) \log(n) + O(n \log(n))$ base field operations, using the subproduct tree construction recalled above. However, the proof of Lemma 1 shows that better can be done when the points $\mathbf{x}$ form a geometric progression. Namely, in this case the construction of the whole tree can be avoided and one can decrease the cost of computing $T_{m,0}$ to $\mathsf{M}(n) + O(n)$. Similarly, if the points $\mathbf{x}$ form an arithmetic progression, the polynomial $T_{m,0}$ can be computed within $2\, \mathsf{M}(n) + O(n)$ base field operations, provided k has characteristic zero or larger than n. Indeed, under this hypothesis, the node $T_{i+1,0}$ can be deduced from $T_{i,0}$ by a polynomial shift using the fast algorithm of **?** (which we recall in Section 4). These useful facts seem not to have been mentioned before in the classical computer algebra textbooks.

As a final remark, note that, for some algorithms presented later, only the *even nodes* $T_{i,2j}$ from the subproduct tree are necessary. If this is the case,

one is led to the natural question: can these polynomials be computed faster than all the subproduct tree? For the moment, we are unable to satisfactorily answer this question, even in the arithmetic and the geometric case.

2.2 Transposed algorithms

Introduced under this name by Kaltofen and Shoup, the *transposition principle* is an algorithmic theorem, with the following content: given an algorithm that performs an $(r+n) \times r$ matrix-vector product, one can deduce an algorithm with the same complexity, up to $O(n)$, and which performs the transposed matrix-vector product. See for instance **?** for a precise statement and **?** for historical notes and further comments on this question.

For this result to apply, some restrictions must be imposed on the computational model; for our purposes, it will suffice to impose a condition on the univariate polynomial multiplication algorithm. Namely, we require that to compute the product of a polynomial a by a polynomial b, only linear operations in the coefficients of b are done. This is no strong restriction, since all classical multiplication algorithms mentioned in the introduction satisfy this assumption.

We can then introduce our basic transposed operation. Let us denote by $k[x]_i$ the vector space of polynomials of degree at most i; then given $a \in k[x]$ of degree r, we denote by $\mathrm{mul}^t(n, a, .) : k[x]_{r+n} \to k[x]_n$ the transpose of the (k-linear) multiplication-by-a map $k[x]_n \to k[x]_{r+n}$.

Various algorithms for computing the transposed multiplication are detailed in **??**. The transposition principle implies that under the assumption above, whatever the algorithm used for polynomial multiplication is, the cost of the direct and of the transposed multiplication are equal, up to $O(r)$ operations in k; for instance, if $n < r$, the complexity of $\mathrm{mul}^t(n, a, .)$ is $\mathsf{M}(r) + O(r)$.

All algorithms to be transposed later rely mainly on polynomial multiplication, as well as other basic operations such as multiplication by diagonal matrices or additions. We now know how to perform transposed polynomial multiplications, and all other basic operations are easily transposed; following **?**, a basic set of mechanical transformations will then easily yield all required transposed algorithms.

3 Algorithms for arbitrary sample points

In this section, we treat the questions of evaluation and interpolation in the monomial and Newton bases, and base change algorithms, for an arbitrary choice of points $\mathbf{x} = x_0, \ldots, x_{n-1}$. Recall that n is supposed to be a power of 2.

3.1 Conversions between Newton basis and monomial basis

We first estimate the complexity for the conversions between monomial and Newton bases. The results are summarized in the theorem below:

Theorem 1 *Let k be a field, let $\mathbf{x} = x_0, \ldots, x_{n-1}$ be n elements of k and let $F \in k[x]$ of degree less than n. Suppose that n is a power of 2 and that the subproduct tree associated to the points $\mathbf{x}$ has been precomputed. Then:*

- *given the coefficients of F in the Newton basis, one can recover the coefficients of F in the monomial basis using $1/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k.*
- *given the coefficients of F in the monomial basis, one can recover the coefficients of F in the Newton basis using $1/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k.*

Taking into account the complexity of computing the subproduct tree stated in Proposition 1, we obtain the estimates given in the first column, first two rows, in Table 1.

From Newton basis to monomial basis. Let F be a polynomial of degree less than $n = 2^m$ and $\mathbf{f} = f_0, \ldots, f_{n-1}$ its coefficients in the Newton basis. Given $\mathbf{f}$, we want to recover the coefficients of F in the monomial basis. To this effect, we write the equality

$$\begin{aligned} F &= F_0 + (x - x_0)\cdots(x - x_{n/2-1})F_1, \quad \text{with} \\ F_0 &= f_0 + f_1(x - x_0) + \cdots + f_{n/2-1}(x - x_0)\cdots(x - x_{n/2-2}), \\ F_1 &= f_{n/2} + f_{n/2+1}(x - x_{n/2}) + \cdots + f_{n-1}(x - x_{n/2})\cdots(x - x_{n-2}). \end{aligned}$$

Using this decomposition, the following conversion algorithm can be deduced. On input the coefficients $\mathbf{f} = f_0, \ldots, f_{n-1}$ and the subproduct tree $\mathcal{T}$ associated to the points $x_0, \ldots, x_{n-1}$, it outputs the expansion of F on the monomial basis. The following algorithm works in place, since the input list $\mathbf{f}$ is modified

at each iteration (for type consistency we might see the input constants as polynomials of degree zero).

```
NewtonToMonomial(T, f)
    for i ← 0 to m − 1 do
        for j ← 0 to 2^{m−i−1} − 1 do
            f_j ← f_{2j} + T_{i,2j} f_{2j+1};
    return f_0;
```

Let us assume that the even nodes $T_{i,2j}$ of the subproduct tree associated to the points $x_0, \ldots, x_{n-1}$ have been precomputed. Then, the number of operations in k used by algorithm **NewtonToMonomial** is upper bounded by

$$\sum_{i=0}^{m-1} \left(\sum_{j=0}^{2^{m-i-1}-1} \Big(\mathsf{M}(2^i) + O(2^i) \Big) \right) = 1/2\, \mathsf{M}(n) \log(n) + O(n \log(n)).$$

This proves the first part of Theorem 1. This algorithm was already presented in (**?**, Th. 2.5), but the more precise complexity estimate given here is needed in the following.

Transposed conversion algorithm. The conversion algorithm described above computes a base change map, which is linear in F. In what follows, we are interested in computing the inverse of this map, that is, the converse base change. As a first step, we now discuss the transposed map.

Constructing the subproduct tree associated to the points $\mathbf{x}$ is a precomputation, which does not depend on the polynomial F, and is not transposed. As to the two nested loops, we use the transposed multiplication introduced previously. With this operation, we obtain by a mechanical transformation the following transposed conversion algorithm: increasing loop indices become decreasing indices, polynomial multiplications become transposed multiplications, and additions become duplications.

The direct version takes the subproduct tree and a list of coefficients as input and gives its output in the form of a polynomial; the transposed algorithm takes the subproduct tree and a polynomial as input and outputs a list of constant polynomials, that is, of constants.

TNewtonToMonomial$(\mathcal{T}, F)$
  $c_0 \leftarrow F$;
  for $i \leftarrow m-1$ **downto** 0 **do**
    for $j \leftarrow 2^{m-i-1}-1$ **downto** 0 **do**
      $c_{2j+1} \leftarrow \mathrm{mul}^t(2^i-1, T_{i,2j}, c_j)$;
      $c_{2j} \leftarrow c_j \mod x^{2^i}$;
  return $c_0, \ldots, c_{2^m-1}$;

It follows from either a direct analysis or the transposition principle that, if the subproduct tree is already known, the complexity of this algorithm is the same as that of the direct one, that is, $1/2\,\mathsf{M}(n)\log(n)+O(n\log(n))$ base field operations.

From monomial basis to Newton basis. We can now resume our study of conversion algorithms. Let F be a polynomial of degree less than n, whose coefficients on the monomial basis are known; we want to recover the coefficients of F on the Newton basis $1, x-x_0, \ldots, (x-x_0)\cdots(x-x_{n-2})$.

A natural way to do that is based on the following remark: if we write F as follows:

$$F = F_0 + (x-x_0)\cdots(x-x_{n/2-1})F_1,$$

then it is enough to recover the coefficients of F_0 and F_1 on the Newton bases $1, x-x_0, \ldots, (x-x_0)\cdots(x-x_{n/2-2})$ and $1, x-x_{n/2}, \ldots, (x-x_{n/2})\cdots(x-x_{n-2})$ respectively.

Using this remark, one can deduce a conversion algorithm based on (recursive) division with quotient and remainder by the even nodes $T_{i,2j}$, see (**?**, Theorem 2.4) for details. Using the idea in (**?**, p. 22–24) to save constant factors in the division by the tree nodes, the cost of the resulting algorithm (without counting the precomputation of the subproduct tree) is upper bounded by $2\,\mathsf{M}(n)\log(n)+O(n\log(n))$.

In what follows, we obtain an algorithm of better complexity by studying the transpose of this conversion algorithm. Indeed, we will show that this transposed map mainly amounts to a conversion from Newton to monomial basis on a modified set of points; transposing backwards will yield the desired result. Let us notice that the same approach – that is, looking at the dual problem – was already successfully applied by **?** to speed up algorithms for multipoint evaluation in the monomial basis.

Our starting point is the following result (where we take the usual convention that the empty sum is zero).

Lemma 2 *Let A be the matrix of change of base from the monomial basis to the Newton basis associated to $x_0, \ldots, x_{n-1}$. Then, for $1 \leq i, j \leq n$, the (i,j)th entry of A equals*

$$A_{i,j}(x_0, \ldots, x_{i-1}) = \sum_{\alpha_0 + \cdots + \alpha_{i-1} = j-i} x_0^{\alpha_0} x_1^{\alpha_1} \cdots x_{i-1}^{\alpha_{i-1}}.$$

Proof. The proof is an induction on $i = 1, \ldots, n$, with j fixed. For $i = 1$, $A_{1,j}$ is the constant coefficient of x^j in the Newton basis, that is, x_0^j, so our claim holds. For $i > 1$, $A_{i+1,j}$ is obtained by computing the ith divided difference of $A_{i,j}$, that is,

$$\begin{cases} \frac{A_{i,j}(x_0,\ldots,x_{i-2},x_{i-1}) - A_{i,j}(x_0,\ldots,x_{i-2},x_i)}{x_{i-1}-x_i} & \text{if } x_{i-1} \neq x_i \\ \frac{\partial A_{ij}}{\partial x_i} & \text{if } x_{i-1} = x_i, \end{cases}$$

see for instance (**?**). The conclusion follows by an easy computation. □

In this matricial formulation, our primary goal is thus to study the map of multiplication by A. As announced above, we start by considering the transposed map, of multiplication by A^t.

From Lemma 2, we see that modulo x^n, the generating series of the columns of A^t are all rational and respectively equal

$$\frac{1}{1 - xx_0}, \frac{x}{(1 - xx_0)(1 - xx_1)}, \frac{x^2}{(1 - xx_0)(1 - xx_1)(1 - xx_2)}, \ldots$$

Let then $f_0, \ldots, f_{n-1}$ be in k and $\mathbf{f} = f_0, \ldots, f_{n-1}$. A direct computation shows that the entries of the product between A^t and the vector $[f_0, \ldots, f_{n-1}]^t$ are the coefficients of $1, x, \ldots, x^{n-1}$ in the Taylor expansion of

$$G(x) = \sum_{j=0}^{n-1} \frac{f_j x^j}{\prod_{\ell=0}^{j}(1 - x_\ell x)} = \frac{\mathrm{rev}(n-1, Q(x))}{\mathrm{rev}(n, T_{m,0})},$$

where

$$Q(x) = f_{n-1} + f_{n-2}(x - x_{n-1}) + \cdots + f_0(x - x_{n-1}) \cdots (x - x_1)$$

and where $\mathrm{rev}(\ell, P(x)) = x^\ell P(1/x)$ for any $P \in k[x]$ and for all $\ell \geq \deg(P)$.

The polynomial Q can be obtained by applying the algorithm **Newton-ToMonomial** to the input values $\widetilde{\mathbf{f}} = f_{n-1}, \ldots, f_0$ and $\widetilde{\mathbf{x}} = x_{n-1}, \ldots, x_0$. Computing the Taylor expansion of S to recover G requires one additional power series inversion and one power series multiplication.

Let us denote by $\widetilde{\mathcal{T}}$ the subproduct tree associated to $\widetilde{\mathbf{x}}$. Then the algorithm above is summarized as follows.

TMonomialToNewton$(\widetilde{\mathcal{T}}, \mathbf{f})$
  $I \leftarrow 1/\mathrm{rev}(n, T_{m,0}) \mod x^n$;
  $Q \leftarrow$ **NewtonToMonomial**$(\widetilde{\mathcal{T}}, \widetilde{\mathbf{f}})$;
  $\widetilde{Q} \leftarrow \mathrm{rev}(n-1, Q)$;
  $G \leftarrow I\widetilde{Q} \mod x^n$;
return G;

By transposition, we deduce the following algorithm for computing the matrix-vector product by A. All operations that depend linearly in $\mathbf{f}$ are transposed, and their order is reversed; thus, we now use as a subroutine the algorithm **TNewtonToMonomial** presented in the previous paragraphs. Note that the computation of I is not transposed, since it does not depend on $\mathbf{f}$. The resulting algorithm takes as input a polynomial F and returns its coefficients in the Newton basis.

MonomialToNewton$(\widetilde{\mathcal{T}}, F)$
  $I \leftarrow 1/\mathrm{rev}(n, T_{m,0}) \mod x^n$;
  $\widetilde{Q} \leftarrow \mathrm{mul}^t(n-1, I, F)$;
  $Q \leftarrow \mathrm{rev}(n-1, \widetilde{Q})$;
  $f_{n-1}, \ldots, f_0 \leftarrow$ **TNewtonToMonomial**$(\widetilde{\mathcal{T}}, Q)$;
return $f_0, \ldots, f_{n-1}$;

This algorithm uses the subproduct tree $\widetilde{\mathcal{T}}$. However, this is not a strong limitation: if the subproduct tree $\mathcal{T}$ associated to the points $x_0, \ldots, x_{n-1}$ is already known, then $\widetilde{\mathcal{T}}$ is obtained by reversing the order of the siblings of each node in $\mathcal{T}$.

To conclude, using the first part of Theorem 1 and either the transposition principle or a direct analysis, we deduce that both algorithms require $1/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k, since the additional power series operations have cost in $O(\mathsf{M}(n))$. This is to be compared with the previous estimate of $2\,\mathsf{M}(n)\log(n) + O(n\log(n))$ for the algorithm based on Euclidean division. This estimate proves the second assertion in Theorem 1.

3.2 Completing the table

We conclude this section by filling the last entries of the first column in Table 1. Using the results above and the algorithms of **?**, this is an immediate task.

Evaluation and interpolation on the monomial basis. Let $F \in k[x]$ be a polynomial of degree less than n, let $x_0, \ldots, x_{n-1}$ be n pairwise distinct points in k and denote $v_i = F(x_i)$. The questions of multipoint evaluation and interpolation in the monomial basis consists in computing the coefficients of F in the monomial representation from the values $v_0, \ldots, v_{n-1}$, and conversely.

Fast algorithms for these tasks were given by (**??**), then successively improved by (**?**) and (**?**). All these algorithms are based on (recursive) polynomial remaindering and have complexity $O(\mathsf{M}(n)\log(n))$. Recently, different algorithms, based on the use of transposed operations, have been designed in (**?**, Section 6) and led to improved complexity bounds, by constant factors. For the sake of completeness, we summarize the corresponding results of **?** in the theorem below:

Theorem 2 *Let k be a field, let $\mathbf{x} = x_0, \ldots, x_{n-1}$ be pairwise distinct elements of k and let $F \in k[x]$ of degree less than n. Suppose that n is a power of 2 and that the subproduct tree associated to the points $\mathbf{x}$ has been precomputed. Then:*

- *the evaluation of F at the points $\mathbf{x}$ can be done using $\mathsf{M}(n)\log(n)+O(\mathsf{M}(n))$ operations in k.*
- *the interpolation of F at the points $\mathbf{x}$ can be done using $2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k.*

Taking into account the complexity of computing the subproduct tree, which is within $1/2\,\mathsf{M}(n)\log(n)+O(\mathsf{M}(n))$ operations, we obtain the estimates given in the first column, middle rows, in Table 1.

We conclude by a remark. Interpolation requires to evaluate the derivative of $\prod_{i=0}^{n-1}(x - x_i)$ on all points $\mathbf{x}$, which contributes for $\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ in the estimate above. In the case of an arithmetic or a geometric progression, these values can be computed in linear time: we refer the reader to (**?**) for the arithmetic case and leave him the geometric case as an exercise. Thus the complexity of interpolation drops to

$$(1/2 + 1)\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n)) = 3/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$$

in these cases. In Sections 4 and 5 we show that one can actually do better in these two special cases.

Newton evaluation and interpolation. Combining the results of Sections 3.1 and 3.2, we deduce the following result concerning the complexities of Newton evaluation and interpolation on an arbitrary set of evaluation points.

Theorem 3 *Let k be a field, let $\mathbf{x} = x_0, \ldots, x_{n-1}$ be pairwise distinct elements*

of k and let $F \in k[x]$ of degree less than n. Suppose that n is a power of 2 and that the subproduct tree associated to the points x_i has been precomputed. Then:

- *Newton evaluation of F at the points $\mathbf{x}$ can be done in $3/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k.*
- *Newton interpolation of F at the points $\mathbf{x}$ can be done in $5/2\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ operations in k.*

Taking into account the complexity of computing the subproduct tree, this completes the entries of the first column of Table 1.

4 Special case of an arithmetic progression

In this section we focus on the special case of evaluation points in arithmetic progression, and show that many of the complexity estimates above can be improved in this case.

We begin by recalling a result taken from (**?**, Section 3), which shows that the complexities of Newton evaluation and interpolation drop to $\mathsf{M}(n) + O(n)$ in this special case, and we point out the link between these algorithms and the algorithm for shift of polynomials of **?**. Next, using the transposed algorithm of Section 3.1, we show how to improve (by constant factors) the complexities of evaluation and interpolation in the *monomial* basis on an arithmetic progression. We conclude by an application to computations with linear differential operators.

Newton interpolation and evaluation. We first recall the algorithm of (**?**, Section 3): this gives the last two entries of the second column, in Table 1. For further discussion, we detail the proof.

Proposition 2 *Suppose that k is a field of characteristic 0 or larger than n. Let h be a non-zero element in k. Then, Newton interpolation and evaluation of a polynomial of degree n on the arithmetic sequence $x_i = x_0 + ih$, for $i = 0, \ldots, n-1$ can be done using $\mathsf{M}(n) + O(n)$ operations in k.*

Proof. Let F be a polynomial of degree less than n, $\mathbf{v} = v_0, \ldots, v_{n-1}$ the values $F(x_i)$ and $\mathbf{f} = f_0, \ldots, f_{n-1}$ the coefficients of F on the Newton basis associated to the points $\mathbf{x} = x_0, \ldots, x_{n-1}$. Evaluating Formula (1) at $\mathbf{x}$, we deduce the

following equalities relating the values $\mathbf{v}$ and the coefficients $\mathbf{f}$:

$$\begin{aligned} v_0 &= f_0 \\ v_1 &= f_0 + hf_1 \\ v_2 &= f_0 + 2hf_1 + (2h \cdot h)f_2 \\ v_3 &= f_0 + 3hf_1 + (3h \cdot 2h)f_2 + (3h \cdot 2h \cdot h)f_3 \quad \ldots \end{aligned}$$

They suggests to introduce the auxiliary sequence $\mathbf{w} = w_0, \ldots, w_{n-1}$ defined by

$$w_i = \frac{v_i}{i!h^i}, \quad i = 0, \ldots, n-1.$$

Note that the sequences $\mathbf{v}$ and $\mathbf{w}$ can be deduced from one another for $O(n)$ base field operations. Using the sequence $\mathbf{w}$, the relations above become

$$w_i = \sum_{j+k=i} \frac{1}{h^k k!} f_j.$$

Introducing the generating series

$$W = \sum_{i=0}^{n-1} w_i x^i, \quad F = \sum_{i=0}^{n-1} f_i x^i, \quad S = \sum_{i=0}^{n-1} \frac{1}{i!h^i} x^i,$$

all relations above are summarized in the equation $W = FS$ modulo x^n. Since S is the truncation of $\exp(x/h)$, its inverse S^{-1} is the truncation of $\exp(-x/h)$, so multiplying or dividing by S modulo x^n can be done in $\mathsf{M}(n) + O(n)$ base field operations. We deduce that W and F can be computed from one another using $\mathsf{M}(n) + O(n)$ base field operations. This proves the proposition. □

Let us make a few comments regarding the previous algorithm. The problem of Newton evaluation on the arithmetic sequence $x_i = x_0 + ih$ is closely related to that of Taylor shift by $1/h$. More precisely, the matrix Newton_h of Newton evaluation is equal, up to multiplication by diagonal matrices, to the transpose of the matrix $\mathrm{Shift}_{1/h}$ representing the map $F(x) \mapsto F(x + 1/h)$ in the monomial basis. Indeed, the following matrix equality is easy to infer:

$$\mathrm{Newton}_h = \mathrm{Diag}\Big(1, h, h^2, \ldots, h^{n-1}\Big) \cdot \mathrm{Shift}_{1/h}^t \cdot \mathrm{Diag}\Big(0!, 1!, \ldots, (n-1)!\Big). \quad (2)$$

In the same vein, one can also interpret Newton interpolation as the transpose of Taylor shift by $-1/h$ (up to diagonal matrices). A simple way to see this is to take the inverse of Equation (2) and to use the equality between $\mathrm{Shift}_{1/h}^{-1}$ and $\mathrm{Shift}_{-1/h}$.

Now, over fields of characteristic zero or larger than n, a classical algorithm of **?** solves the Taylor shift problem within $\mathsf{M}(n) + O(n)$ operations. Given a degree $n-1$ polynomial $F(x) = \sum_{i=0}^{n-1} f_i x^i$, the algorithm in **?** computes the

coefficients of $\mathrm{Shift}_{1/h}(F) = F(x + 1/h)$ by exploiting Taylor's formula

$$\mathrm{Shift}_{1/h}(F) = \sum_{j=0}^{n-1} F^{(j)}(1/h)\,\frac{x^j}{j!}$$

and the fact that $F^{(j)}(1/h)$ is the coefficient of x^{n-j-1} in the product

$$\left(\sum_{i=0}^{n-1} i!\, f_i x^{n-i-1}\right) \cdot \left(\sum_{i=0}^{n-1} \frac{x^i}{i!\, h^i}\right).$$

In view of Equation (2), it is immediate to show that the algorithm for Newton evaluation on an arithmetic progression presented in Proposition 2 can be interpreted as the *transposition* of the algorithm in **?** (up to diagonal matrix multiplications) and thus could have been deduced automatically from that algorithm using the effective transposition tools in **?**.

Conversion between monomial and Newton bases. To fill the second column of Table 1, our next step is to consider the base change algorithms, which occupy the first and second rows. To perform these conversions, we use the same algorithms as in the case of arbitrary sample points; the complexity results are thus those given in the previous section.

Evaluation and interpolation on the monomial basis. We conclude this systematic exploration by studying evaluation and interpolation on the monomial basis, for points in an arithmetic progression. A surprising consequence of Proposition 2 is the following corollary: one can speed up both monomial evaluation and interpolation using the Newton basis for intermediate computations. This gives the middle entries of the second column in Table 1.

Corollary 1 *Let n be a power of 2 and let k be a field of characteristic 0 or larger than n. Let $F \in k[x]$ of degree less than n and let $x_0, \ldots, x_{n-1}$ be an arithmetic progression in k. Then:*

- *Given the coefficients of F on the monomial basis, $F(x_0), \ldots, F(x_{n-1})$ can be computed in $\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ base field operations.*
- *Given the values $F(x_0), \ldots, F(x_{n-1})$, all coefficients of F on the monomial basis can be computed in $\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ base field operations.*

The proof comes easily by combining the results of Proposition 1, Theorem 1 and Proposition 2.

Applications. Our initial interest in improving evaluation and interpolation on the points of an arithmetic progression was motivated by the study of linear recurrences with polynomial coefficients presented in (**??**): the algorithms therein can benefit from any improvement on evaluation and interpolation on an arithmetic progression. The cryptographic-sized record obtained by **?** requires to work in degree several tens of thousands, and gaining even a constant factor is interesting in such sizes.

We conclude this section by describing another application which comes from the domain of exact computations with linear differential operators. While computing with such operators, it is sometimes easier to work with the *Euler derivation* $\delta = x\frac{\partial}{\partial x}$ instead of the usual derivation $\mathcal{D} = \frac{\partial}{\partial x}$ (see below for an application example). We now estimate the complexity of performing this base change.

Corollary 2 *Let n be a power of 2 and k a field of characteristic zero or larger than n.*

Let $\mathcal{L} = \sum_{i=0}^{n-1} p_i(x)\delta^i$ be a linear differential operator with polynomial coefficients of degree at most d, and let $q_0, \ldots, q_{n-1}$ be the unique polynomials such that $\mathcal{L} = \sum_{i=0}^{n-1} q_i(x)\mathcal{D}^i$. Then all q_i can be computed in

$$d\,\mathsf{M}(n)\log(n) + O(d\,\mathsf{M}(n))$$

operations in k.

Let $\mathcal{L} = \sum_{i=0}^{n-1} q_i(x)\mathcal{D}^i$ be a linear differential operator with polynomial coefficients of degree at most d and let $p_0, \ldots, p_{n-1}$ be the unique Laurent polynomials in $k[x, x^{-1}]$ such that $\mathcal{L} = \sum_{i=0}^{n-1} p_i(x)\delta^i$. Then all p_i can be computed in

$$(n+d)\,\mathsf{M}(n)\log(n) + O((n+d)\,\mathsf{M}(n))$$

operations in k.

Proof. Converting from the representation in δ to that in $\mathcal{D}$, or backwards, amounts to compute several matrix-vector products Sv or $S^{-1}v$, where S is the $n \times n$ matrix

$$S = \begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & S_{1,n} \\ 0 & 1 & 3 & 7 & \ldots & S_{2,n} \\ 0 & 0 & 1 & 6 & \ldots & S_{3,n} \\ 0 & 0 & 0 & 1 & \ldots & S_{4,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & S_{n,n} \end{bmatrix}.$$

Indeed, one has the equalities, seemingly known by (**?**), see also **?**:

$$\begin{aligned}
\delta^1 &= x\mathcal{D},\\
\delta^2 &= x\mathcal{D} + x^2\mathcal{D}^2,\\
\delta^3 &= x\mathcal{D} + 3\,x^2\mathcal{D}^2 \;+ x^3\mathcal{D}^3,\\
\delta^4 &= x\mathcal{D} + 7\,x^2\mathcal{D}^2 \;+ 6\,x^3\mathcal{D}^3 \;+ x^4\mathcal{D}^4,\\
\delta^5 &= x\mathcal{D} + 15\,x^2\mathcal{D}^2 + 25\,x^3\mathcal{D}^3 + 10\,x^4\mathcal{D}^4 + x^5\mathcal{D}^5, \quad \ldots
\end{aligned}$$

Let thus $\mathcal{L} = \sum_{i=0}^{n-1} p_i(x)\delta^i$, with $p_i(x) = \sum_{j=0}^{d} p_{i,j}x^j$. Then in matrix form, $\mathcal{L}$ writes as the product

$$\begin{bmatrix} 1 & x & \ldots & x^d \end{bmatrix} \cdot \begin{bmatrix} p_{0,0} & p_{1,0} & \cdots & p_{n-1,0} \\ \vdots & \vdots & \vdots & \vdots \\ p_{0,d} & p_{1,d} & \cdots & p_{n-1,d} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \delta \\ \vdots \\ \delta^{n-1} \end{bmatrix}.$$

Thus, the previous equalities show that rewriting $\mathcal{L}$ in $\mathcal{D}$ amounts to perform d matrix-vector products by the Stirling matrix S, followed by the addition of d polynomials, each of which having at most n non-zero coefficients.

Conversely, let $\mathcal{L} = \sum_{i=0}^{n-1} q_i(x)\mathcal{D}^i$ be written as an operator of order $n-1$ in $\mathcal{D}$, with coefficients $q_i(x) = \sum_{j=0}^{d} q_{i,j}x^j$ that are polynomials of degree at most d. Expressing $\mathcal{L}$ in the matricial form:

$$\mathcal{L} = \begin{bmatrix} x^{-(n-1)} & x^{-(n-2)} & \ldots & x^d \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & \ldots & q_{n-1,0} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & q_{1,0} & \ldots & q_{n-1,d} \\ q_{0,0} & q_{1,1} & \ldots & \vdots \\ & \vdots & & \\ \vdots & q_{1,d} & & \vdots \\ q_{0,d} & 0 & \ldots & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x\mathcal{D} \\ \vdots \\ x^{n-1}\mathcal{D}^{n-1} \end{bmatrix}$$

shows that the problem of computing $\mathcal{L}$ as an operator in δ with rational function coefficients is reduced to $n+d$ matrix-vector multiplications by the inverse of the Stirling matrix.

Thus, it suffices to estimate the cost of performing a matrix-vector product by S or its inverse. The entries $(S_{i,j})$ of the matrix S are the *Stirling numbers of the second kind*; they satisfy the recurrence $S_{i,j+1} = S_{i-1,j} + iS_{i,j}$ (while the entries of S^{-1} are, up to sign, the Stirling numbers of the first kind). These numbers also represent the coordinates of ordinary powers $1, x, \ldots, x^{n-1}$ in the falling factorial basis $1, x^{\underline{1}} = x, \ldots, x^{\underline{n-1}} = x(x-1)\cdots(x-n+2)$. For

instance, for $j = 1, \ldots, 5$ these relations write

$$\begin{aligned}
x^1 &= x^{\underline{1}},\\
x^2 &= x^{\underline{1}} + x^{\underline{2}},\\
x^3 &= x^{\underline{1}} + 3\,x^{\underline{2}} \ + x^{\underline{3}},\\
x^4 &= x^{\underline{1}} + 7\,x^{\underline{2}} \ + 6\,x^{\underline{3}} \ + x^{\underline{4}},\\
x^5 &= x^{\underline{1}} + 15\,x^{\underline{2}} + 25\,x^{\underline{3}} + 10\,x^{\underline{4}} + x^{\underline{5}}, \quad \ldots
\end{aligned}$$

Hence, the entries of the vector Sv represent the coefficients of the polynomial $\sum_{i=0}^{n-1} v_i x^i$ in the Newton basis $1, x^{\underline{1}}, x^{\underline{2}}, \ldots$ Similarly, computing $S^{-1}v$ amounts to converting a polynomial from its Newton representation (in the falling factorial basis) to the monomial one. Using the conversion algorithms above, both conversions can be done in complexity $\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$, which concludes the proof. □

As an application, recall that the coefficients of a power series $\sum_{i\geq 0} s_i x^i$ which is a solution of a linear differential operator $\mathcal{L}$ satisfy a linear recurrence, whose coefficients can be read off the coefficients of $\mathcal{L}$ when it is written in δ. More precisely, if $\mathcal{L} = \sum_{i=0}^{n-1} p_i(x)\delta^i$ has coefficients $p_i(x) = \sum_{j=0}^{d} p_{i,j}x^j$, then letting $\widetilde{p_j}(x) = \sum_{i=0}^{n-1} p_{i,j}x^i$ for $0 \leq j \leq d$, the recurrence satisfied by the s_i writes

$$\widetilde{p_d}(i)s_i + \cdots + \widetilde{p_0}(i+d)s_{i+d} = 0, \qquad \text{for all } \ i \geq 0.$$

This remark yields a fast algorithm to convert a differential equation of order $n-1$ in $\mathcal{D}$ with polynomial coefficients of degree at most d to the recurrence satisfied by a power series solution. By the previous considerations, its complexity is asymptotic to $(n+d)\,\mathsf{M}(n)\log(n)$, up to lower order terms. In comparison, the classical method uses the recurrence

$$\sum_{j=0}^{n-1}\sum_{k=0}^{d} p_{k,j}(i-k+1)(i-k+2)\cdots(i-k+j)s_{n+j-k-1} = 0$$

and amounts to compute the $d(n-1)$ polynomials $(x-k+1)(x-k+2)\cdots(x-k+j)$, which can be done in complexity of $O(dn\,\mathsf{M}(n))$. If d and n are of the same order, our method saves a factor of about n.

5 The geometric progression case

Simplifications in Newton evaluation and interpolation formulas also arise when the sample points form a geometric progression; this was already pointed out in **?** and references therein, but that article makes no mention of asymptotically fast algorithms.

In this section we show that the complexities of Newton evaluation and interpolation on a geometric progression of size n drop to $\mathsf{M}(n) + O(n)$. By transposition, we deduce that the conversion between monomial and Newton bases have the same asymptotic cost. Last, as in the previous section, we obtain as corollaries fast algorithms for evaluation and interpolation on the monomial basis: the complexities of both tasks is shown to be in $O(\mathsf{M}(n))$, *i.e.* better by a logarithmic factor than in the case of arbitrary samples points.

Thus, geometric progressions should be considered as interesting choices for algorithms relying on evaluation and interpolation techniques. We illustrate this in the case of polynomial matrix multiplication algorithms.

In all what follows, we actually assume for simplicity that the geometric progression we consider has the form $x_i = q^i$, $i = 0, \ldots, n-1$. Treating the general case $x_i = x_0 q^i$, with arbitrary x_0, does not alter the asymptotic estimates, and only burdens the notation. Finally, we mention that many formulas presented below can be thought as q-analogues of those presented in the previous section.

Newton interpolation and evaluation. Our first question is that of Newton interpolation and evaluation: the following proposition proves the estimates of the last entry in the third column of Table 1.

Proposition 3 *Let k be a field and let $q \in k$ such that the elements $x_i = q^i$ are different from 1, for $i = 0, \ldots n-1$. Then Newton interpolation and evaluation on the geometric sequence $1, q, \ldots, q^{n-1}$ can be done using $\mathsf{M}(n) + O(n)$ base field operations.*

Proof. Let F be a polynomial of degree less than n, let $\mathbf{v} = v_0, \ldots, v_{n-1}$ be the values $F(x_i)$ and $\mathbf{f} = f_0, \ldots, f_{n-1}$ the coefficients of F on the Newton basis associated to the points $\mathbf{x}$. As in the previous section, we evaluate Formula (1) on the points $\mathbf{x}$, yielding

$$
\begin{aligned}
v_0 &= f_0 \\
v_1 &= f_0 + (q-1)f_1 \\
v_2 &= f_0 + (q^2-1)f_1 + (q^2-1)(q^2-q)f_2 \\
v_3 &= f_0 + (q^3-1)f_1 + (q^3-1)(q^3-q)f_2 + (q^3-1)(q^3-q)(q^3-q^2)f_3 \quad \ldots
\end{aligned}
$$

Let us introduce the triangular numbers $t_i = 1 + 2 + \cdots + (i-1) = i(i-1)/2$, for $i \geq 0$ and the modified sequence $g_i = q^{t_i} f_i$, for $i = 0, \ldots, n-1$. Note that all coefficients q^{t_i} can be computed in $O(n)$ base field operations, since $q^{t_{i+1}} = q^i q^{t_i}$. Thus, $\mathbf{g} = g_0, \ldots, g_{n-1}$ and $\mathbf{f} = f_0, \ldots, f_{n-1}$ can be computed from one another for $O(n)$ base field operations. With this data, the relations

above become

$$
\begin{aligned}
v_0 &= g_0 \\
v_1 &= g_0 + (q-1)g_1 \\
v_2 &= g_0 + (q^2-1)g_1 + (q^2-1)(q-1)g_2 \\
v_3 &= g_0 + (q^3-1)g_1 + (q^3-1)(q^2-1)g_2 + (q^3-1)(q^2-1)(q-1)g_3 \quad \dots
\end{aligned}
$$

Next, we introduce the numbers w_i defined by

$$w_0 = v_0, \quad w_i = \frac{v_i}{(q-1)\cdots(q^i-1)}, \quad i = 1, \dots, n-1. \tag{3}$$

As above, $\mathbf{w}$ and $\mathbf{v}$ can be computed from one another for $O(n)$ base field operations. Using the modified values w_i, the relations above become

$$w_i = g_i + \sum_{j=0}^{i-1} \frac{1}{(q-1)\cdots(q^{i-j}-1)} g_j.$$

We conclude as in the arithmetic case. We introduce the generating series

$$W = \sum_{i=0}^{n-1} w_i x^i, \quad G = \sum_{i=0}^{n-1} g_i x^i, \quad T = 1 + \sum_{i=1}^{n-1} \frac{1}{(q-1)\cdots(q^i-1)} x^i, \tag{4}$$

so that the relations above become $W = GT$ modulo x^n. All coefficients of the power series T can be obtained in $O(n)$ base field relations. By a classical identity (see for instance **?** and the references therein) the inverse of T modulo x^n equals

$$1 + \sum_{i=1}^{n-1} \frac{q^{\frac{i(i-1)}{2}}(-1)^i}{(q-1)\cdots(q^i-1)} x^i,$$

thus its coefficients can also be obtained in $O(n)$ operations. The conclusion follows. □

For the sake of completeness, we summarize below the algorithm for Newton evaluation on the geometric sequence $1, q, \dots, q^{n-1}$. For a polynomial P, we denote by $\texttt{Coeff}(P, i)$ the coefficient of x^i in P. For simplicity, we take as input all powers of q; of course, given q only, they can be computed for $n-2$ additional multiplications.

NewtonEvalGeom$(1, q, \dots, q^{n-1}, f_0, \dots, f_{n-1})$
  $q_0 \leftarrow 1$; $u_0 \leftarrow 1$; $g_0 \leftarrow f_0$;
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $q_i \leftarrow q_{i-1} \cdot q^{i-1}$;
    $u_i \leftarrow u_{i-1} \cdot (q^i - 1)$;
    $g_i \leftarrow q_i f_i$;
  $W \leftarrow (\sum_{i=0}^{n-1} g_i x^i) \cdot (\sum_{i=0}^{n-1} u_i^{-1} x^i)$;
return $u_0 \,\texttt{Coeff}(W, 0), \dots, u_{n-1} \,\texttt{Coeff}(W, n-1)$;

The algorithm for Newton interpolation follow in an analogous manner from the proof of the proposition above. We give it for completeness.

NewtonInterpGeom$(1, q, \ldots, q^{n-1}, v_0, \ldots, v_{n-1})$
  $q_0 \leftarrow 1$; $u_0 \leftarrow 1$; $w_0 \leftarrow v_0$;
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $q_i \leftarrow q_{i-1} \cdot q^{i-1}$;
    $u_i \leftarrow u_{i-1} \cdot (q^i - 1)$;
    $w_i \leftarrow v_i / u_i$;
  $G \leftarrow (\sum_{i=0}^{n-1} w_i x^i) \cdot (\sum_{i=0}^{n-1} (-x)^i q_i / u_i)$;
return $\texttt{Coeff}(G, 0)/q_0, \ldots, \texttt{Coeff}(G, n-1)/q_{n-1}$;

Conversions between monomial and Newton bases. Our next step is to study the complexity of conversion between monomial and Newton bases. We prove the following result, which completes the first two entries in the last column of Table 1.

Proposition 4 *Let k be a field and let $q \in k$ such that the elements $x_i = q^i$ are different from 1, for $i = 0, \ldots n-1$. Then the conversion between the Newton basis associated to $1, q, \ldots, q^{n-1}$ and the monomial basis can be done using $\mathsf{M}(n) + O(n)$ base field operations.*

The proof comes from considering the transposed of the Newton evaluation and interpolation. Indeed, the following lemma relates these questions to those of conversions between monomial and Newton bases.

Lemma 3 *Let k be a field, $q \in k^*$ and $r = 1/q$. Suppose that $1, q, \ldots, q^{n-1}$ are pairwise distinct and define the following matrices:*

- *Let A be the matrix of base change from the Newton basis associated to $1, q, \ldots, q^{n-1}$ to the monomial basis.*
- *Let B be the matrix of Newton evaluation at $1, r, \ldots, r^{n-1}$.*
- *Let D_1 and D_2 be the $n \times n$ diagonal matrices*

$$D_1 = \mathrm{Diag}\left[\frac{q^{i(i-1)/2}}{\prod_{k=1}^{i-1}(q^k - 1)}\right]_{i=1}^{n} \quad \text{and} \quad D_2 = \mathrm{Diag}\left[(-1)^{j-1} q^{\frac{(j-1)(j-2)}{2}}\right]_{j=1}^{n}.$$

Then the matrix equality $A = D_1 \, B^t \, D_2$ holds.

Proof. Given two integers n and k, the *q-binomial coefficient* (**???**) is defined as

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \begin{cases} \frac{1-q^n}{1-q} \cdot \frac{1-q^{n-1}}{1-q^2} \cdots \frac{1-q^{n-k+1}}{1-q^k}, & \text{for } n \geq k \geq 1, \\ 0, & \text{for } n < k \text{ or } k = 0. \end{cases}$$

The following generalization of the usual binomial formula holds:

$$\prod_{k=1}^{n}\left(1+q^{k-1}x\right) \;=\; \sum_{k=0}^{n}\begin{bmatrix} n \\ k \end{bmatrix}_q q^{\frac{k(k-1)}{2}}\, x^k. \tag{5}$$

From Equation (5), it is then easy to deduce that the entries of the matrix A are

$$A_{i,j} \;=\; (-1)^{j-i}\begin{bmatrix} j-1 \\ i-1 \end{bmatrix}_q q^{(j-i)(j-i-1)/2}.$$

On the other hand, the (i,j) entry of the matrix representing Newton evaluation with respect to $x_0,\ldots,x_{n-1}$ is zero if $j<i$ and equals $\prod_{k=1}^{j-1}\left(x_{i-1}-x_{k-1}\right)$ for all $j\geq i\geq 1$. Applying this to $x_i=1/q^i$, we get

$$B_{i,j} = (-1)^{j-1}\cdot\prod_{k=1}^{j-1}\frac{q^{i-k}-1}{q^{i-1}}, \;\text{ for all }\; j\geq i\geq 1.$$

Having the explicit expressions of the entries of A and B allows to write the equality

$$\frac{B^t_{i,j}}{A_{i,j}} = (-1)^{j-1}\frac{q^{\frac{i(i-1)}{2}+\frac{(j-1)(j-2)}{2}}}{(q-1)\cdots(q^{i-1}-1)},$$

from which the lemma follows. □

Thus, up to multiplications by diagonal matrices, the conversion maps between monomial and Newton bases are the transposes of those of Newton evaluation and interpolation, at the cost of replacing q by $1/q$. The proof of Proposition 4 is now immediate, since the two diagonal matrices involved can be computed in time $O(n)$.

For the sake of completeness, we give below the algorithm for the conversion from Newton to monomial basis on the geometric sequence $1,q,\ldots,q^{n-1}$. We obtain it using Lemma 3 above and by transposing the algorithm for Newton evaluation on a geometric sequence, as described in the proof of Proposition 3.

NewtonToMonomialGeom$(1,\ldots,q^{n-1},f_0,\ldots,f_{n-1})$
  $q_0\leftarrow 1$; $v_0\leftarrow f_0$; $u_0\leftarrow 1$; $w_0\leftarrow f_0$; $z_0\leftarrow 1$;
  for $i\leftarrow 1$ **to** $n-1$ **do**
    $q_i\leftarrow q_{i-1}\cdot q^{i-1}$;
    $v_i\leftarrow (-1)^i f_i q_i$;
    $u_i\leftarrow u_{i-1}q^i/(1-q^i)$;
    $w_i\leftarrow v_i/u_i$;
    $z_i\leftarrow (-1)^i u_i/q_i$;
  $G\leftarrow \mathrm{mul}^t(n-1,\sum_{i=0}^{n-1}u_ix^i,\sum_{i=0}^{n-1}w_ix^i)$;
return $z_0\,\mathtt{Coeff}(G,0),\ldots,z_{n-1}\,\mathtt{Coeff}(G,n-1)$;

Similarly, the algorithm for the conversion from monomial to Newton basis on a geometric sequence can be deduced using again Lemma 3 and the transposition of the algorithm for Newton interpolation on a geometric sequence given in the proof of Proposition 3. We state it below.

MonomialToNewtonGeom$(1, \ldots, q^{n-1}, v_0, \ldots, v_{n-1})$
  $q_0 \leftarrow 1$; $u_0 \leftarrow 1$; $w_0 \leftarrow v_0$; $z_0 \leftarrow 1$; $f_0 \leftarrow v_0$;
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $q_i \leftarrow q_{i-1} \cdot q^{i-1}$;
    $u_i \leftarrow u_{i-1} q^i / (1-q^i)$;
    $w_i \leftarrow v_i / u_i$;
    $z_i \leftarrow (-1)^i u_i / q_i$;
    $f_i \leftarrow (-1)^i w_i q_i$;
  $G \leftarrow \mathrm{mul}^t(n-1, \sum_{i=0}^{n-1} z_i x^i, \sum_{i=0}^{n-1} f_i x^i)$;
return $z_0\,\mathtt{Coeff}(G,0), \ldots, z_{n-1}\,\mathtt{Coeff}(G, n-1)$;

Evaluation and interpolation on the monomial basis. We now treat the question of fast monomial evaluation and interpolation on a geometric progression. As before, we take $x_i = q^i$, $i = 0, \ldots, n-1$, where $q \in k$ is such that the elements $1, q, \ldots, q^{n-1}$ be pairwise distinct.

It is known that evaluating a polynomial of degree less than n on the geometric progression $1, q, \ldots, q^{n-1}$ can be done using $O(\mathsf{M}(n))$ operations. This operation, generalizing the discrete Fourier transform, is called the *chirp transform* and has been independently studied by **?** and by **?**, see also **?**. In contrast, to the best of our knowledge, no algorithm for the inverse operation – interpolation at a geometric progression – has been given yet. Our aim is now to show that the inverse chirp transform can be performed in a similar asymptotic complexity. These results are gathered in the following proposition, which completes the entries of Table 1.

Proposition 5 *Let k be a field, let $n \geq 0$ and let $q \in k$ such that the elements $x_i = q^i$, for $i = 0, \ldots n-1$, are pairwise distinct. If $F \in k[x]$ has degree less than n then:*

- *Given the coefficients of F on the monomial basis, then all the values $F(x_i)$, for $0 \leq i \leq n-1$, can be computed in $2\mathsf{M}(n) + O(n)$ base field operations.*
- *Given the values $F(x_0), \ldots, F(x_{n-1})$, all coefficients of F on the monomial basis can be computed in $2\mathsf{M}(n) + O(n)$ base field operations.*

Proof. We briefly recall how the direct chirp transform works. Write $F =$

$f_0 + f_1 x + \cdots + f_{n-1} x^{n-1}$. Then the algorithm is based on the equalities:

$$F(q^i) = \sum_{j=0}^{n-1} f_j q^{ij} = q^{-i^2/2} \cdot \sum_{j=0}^{n} f_j q^{-j^2/2} q^{(i+j)^2/2}.$$

Suppose first that q is a square in k. Computing the values $b_i = q^{i^2/2}$, for $0 \leq i \leq 2(n-1)$ and $c_j = f_j q^{-j^2/2}$, for $0 \leq j \leq n-1$, takes linear time in n, using the recurrence $q^{(i+1)^2} = q^{i^2} q^{2i} q$. Then, the preceding formula shows that the values $F(q^i)$ are, up to constant factors, given by the *middle part* of the polynomial product

$$\Big(b_0 + b_1 x + \cdots + b_{2(n-1)} x^{2(n-1)}\Big)\Big(c_{n-1} + \cdots + c_0 x^{n-1}\Big).$$

Using standard polynomial multiplication, this algorithm requires $2\mathsf{M}(n)$ operations, but the complexity actually drops to $\mathsf{M}(n) + O(n)$, using the middle product of **?**.

In the general case when q is not a square, several possibilities are available. The first idea is to introduce a square root for q by computing in $K = k[x]/(x^2 - q)$. Another choice is to use the algorithms described previously: performing first a change of base to the Newton representation, and then a Newton evaluation. This way, we obtain the estimate of $2\mathsf{M}(n) + O(n)$ operations for the chirp transform.

Let us now focus on the computation of the inverse chirp transform. As above, we use the Newton basis for intermediate computations: first perform a Newton interpolation, then perform a conversion from the Newton basis to the monomial basis. Both steps have complexities $\mathsf{M}(n) + O(n)$, which gives the estimate of $2\mathsf{M}(n) + O(n)$ operations for the inverse chirp transform. □

Application to polynomial matrix multiplication. We finally apply the results above to improve the complexity of polynomial matrix multiplication. This problem is important, since polynomial matrix multiplication is a primitive of linear algebra algorithms dealing with polynomial matrices (determinant, inversion, system solving, column reduction, integrality certification, normal forms), see for instance (**???**). It also occurs during computations of matrix Padé-type approximants (**?????**), recurrences with polynomial coefficients (**??**) and linear differential operators.

Let $\mathsf{MM}(n, d)$ represent the number of base field operations required to multiply two $n \times n$ matrices with polynomial entries of degree less than d. For simplicity, the cost $\mathsf{MM}(n, 1)$ of scalar $n \times n$ matrix multiplication will be denoted $\mathsf{MM}(n)$. This function is frequently written as $\mathsf{MM}(n) = O(n^\omega)$, where

$2 \leq \omega < 3$ is the so-called *exponent of the matrix multiplication*, see for instance **??**.

? described an algorithm for multiplying degree d polynomials with coefficients from an arbitrary (possibly non commutative) algebra using $O(\mathsf{M}(d))$ algebra operations. Viewing polynomial matrices as polynomials with scalar matrix coefficients, the result in **?** implies that $\mathsf{MM}(n,d) = O\left(\mathsf{M}(d)\,\mathsf{MM}(n)\right)$. Over base fields of cardinality larger than $2d-2$, the use of an evaluation / interpolation scheme allows to uncouple polynomial and matrix products and yields the better bound

$$\mathsf{MM}(n,d) = O\left(\mathsf{MM}(n)\,d + n^2\,\mathsf{M}(d)\log(d)\right). \tag{6}$$

An important remark (**??**) (see also **?**) is that if the base field supports FFT, then choosing the roots of unity as sample evaluation points improves the previous estimate to

$$\mathsf{MM}(n,d) = O\left(\mathsf{MM}(n)\,d + n^2\,d\log(d)\right). \tag{7}$$

However, the algorithm in (**??**) is dependent on the specific use of FFT, which might not be pertinent for polynomials of moderate degrees.

In contrast, using evaluation and interpolation at a geometric progression enables us to obtain the following result.

Theorem 4 *Let $n, d \geq 1$ and let k be a field of characteristic 0, or a finite field of cardinality at least $2d-1$. Then we have the estimate*

$$\mathsf{MM}(n,d) = (2d-1)\,\mathsf{MM}(n) + 6\,n^2\,\mathsf{M}(2d) + O(n^2 d).$$

Proof. In both cases, we use evaluation and interpolation on a geometric progression $1, q, \ldots, q^{2d-2}$ of size $2d-1$. In characteristic 0, we can take $q = 2$. If k is finite, we take for q a generator of the multiplicative group k^* (for practical purposes, we might as well choose q at random, if k has a large enough cardinality). □

Theorem 4 may be seen as an improvement by a log factor of the bound (6), generalizing the bound (7) to an arbitrary multiplication time M function that satisfies our hypotheses. Still, for polynomial matrices of high degrees, the method in (**??**) is better by a constant factor than ours, since the polynomial multiplication uses FFT, and thus itself requires evaluating and interpolating at the roots of unity.

To conclude this paper, Figures 1 and 2 display the speed-up obtained using our polynomial matrix multiplication algorithm, versus a naive product (thus, a larger number means a more significant improvement). The matrix sizes vary

from 1 to 120, the polynomial degrees vary from 0 to 200, and the base field is $\mathbb{Z}/p\mathbb{Z}$, where p is a 32 bit prime. The time ratios are given in the table of Figure 1 and displayed graphically in Figure 2.

The implementation is made using Shoup's NTL C++ library **?**; we used a naive matrix multiplication of cubic complexity, and NTL's built-in polynomial arithmetic. The timings are obtained on an Intel Pentium 4 CPU at 2GHz.

	15	35	55	75	95	115	135	155	175	195
20	0.6	0.8	1.4	1.4	1.6	1.6	2.2	2.1	1.7	1.7
30	1.1	1.2	2.0	2.0	2.4	2.3	3.3	3.2	2.5	2.5
40	1.2	1.6	2.6	2.7	3.2	3.0	4.3	4.1	3.3	3.2
50	1.4	2.0	3.2	3.3	3.9	3.6	5.3	5.1	4.1	4.0
60	1.7	2.3	3.8	3.9	4.6	4.4	6.3	6.1	4.8	4.7
70	1.9	2.6	4.3	4.5	5.3	5.0	7.2	6.9	5.6	5.4
80	2.1	2.9	4.8	4.9	6.0	5.6	8.1	7.7	6.2	5.9
90	2.3	3.3	5.5	5.7	6.6	6.2	9.0	8.6	6.9	6.7
100	2.5	3.5	6.0	6.2	7.3	6.8	9.8	9.3	7.5	7.3
110	2.6	3.9	6.3	6.6	7.8	7.3	10.6	10.1	8.1	7.9

Fig. 1. Speed-up between classical and improved polynomial matrix multiplication. Rows are indexed by the matrix size (20—110); columns are indexed by the matrix degree (15–195).

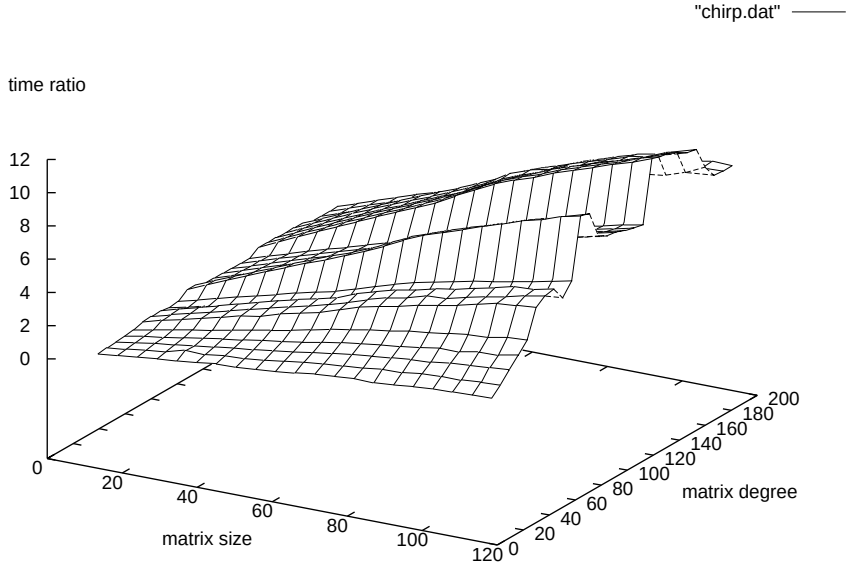

Fig. 2. Speed-up between classical and improved polynomial matrix multiplication.

Acknowledgments. We wish to thank Pierrick Gaudry, Bruno Salvy and Gilles Villard for useful comments on a first version of this article. Our thanks also go to the referees of this paper for their useful remarks.

References

Abramov, S. A., 1989. Rational solutions of linear differential and difference equations with polynomial coefficients. Zh. Vychisl. Mat. i Mat. Fiz. 29 (11), 1611–1620, 1757, english translation in U.S.S.R. Comp. Maths. Math. Phys., 7–12.

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

Beckermann, B., Labahn, G., 1992. A uniform approach for Hermite Padé and simultaneous Padé approximants and their matrix-type generalizations. Numer. Algorithms 3 (1-4), 45–54.

Beckermann, B., Labahn, G., 1994. A uniform approach for the fast computation of matrix-type Padé approximants. SIAM J. Matrix Analysis and Applic. 15 (3), 804–823.

Bini, D., Pan, V. Y., 1994. Polynomial and matrix computations. Vol. 1. Birkhäuser Boston Inc., Boston, MA, fundamental algorithms.

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

Borodin, A., Moenck, R. T., 1974. Fast modular transforms. J. Comput. Syst. Sci. 8 (3), 366–386.

Bostan, A., Gaudry, P., Schost, É., 2004. 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). Vol. 2948 of Lecture Notes in Computer Science. Springer–Verlag, pp. 40–58.

Bostan, A., Lecerf, G., Schost, É., 2003. Tellegen's principle into practice. In: ISSAC'03. ACM Press, pp. 37–44.

Bürgisser, P., Clausen, M., Shokrollahi, M. A., 1997. Algebraic complexity theory. Vol. 315 of Grundlehren Math. Wiss. Springer–Verlag.

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

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

von zur Gathen, J., Gerhard, J., 1999. Modern computer algebra. Cambridge University Press.

Gauss, C. F., 1863. Summatio quarundam serierum singularium. Opera, Vol. 2, Göttingen: Gess. d. Wiss., 9–45.

Gerhard, J., 2000. Modular algorithms for polynomial basis conversion and

greatest factorial factorization. In: RWCA'00. pp. 125–141.

Giorgi, P., Jeannerod, C.-P., Villard, G., 2003. On the complexity of polynomial matrix computations. In: ISSAC'03. ACM Press, pp. 135–142.

Goldman, J., Rota, G.-C., 1970. On the foundations of combinatorial theory. IV. Finite vector spaces and Eulerian generating functions. Studies in Appl. Math. 49, 239–258.

Hanrot, G., Quercia, M., Zimmermann, P., 2004. The Middle Product Algorithm, I. Appl. Algebra Engrg. Comm. Comput. 14 (6), 415–438.

Heine, E., 1847. Untersuchungen über die Reihe $1 + \frac{(1-q^\alpha)(1-q^\beta)}{(1-q)(1-q^\gamma)} \cdot x + \frac{(1-q^\alpha)(1-q^{\alpha+1})(1-q^\beta)(1-q^{\beta+1})}{(1-q)(1-q^2)(1-q^\gamma)(1-q^{\gamma+1})} \cdot x^2 + \ldots$. J. reine angew. Math. 34, 285–328.

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

Kaltofen, E., 2000. Challenges of symbolic computation: my favorite open problems. With an additional open problem by Robert M. Corless and David J. Jeffrey. J. Symb. Comp. 29 (6), 891–919.

Karatsuba, A., Ofman, Y., 1963. Multiplication of multidigit numbers on automata. Soviet Math. Dokl. 7, 595–596.

Knuth, D. E., 1998. The Art of Computer Programming, vol. 2, Seminumerical Algorithms, 3rd Edition. Addison-Wesley, Reading MA.

Moenck, R. T., Borodin, A., 1972. Fast modular transforms via division. 13th Annual IEEE Symposium on Switching and Automata Theory, 90–96.

Montgomery, P. L., 1992. An FFT extension of the elliptic curve method of factorization. Ph.D. thesis, University of California, Los Angeles CA.

Paule, P., 1995. Greatest factorial factorization and symbolic summation. J. Symb. Comp. 20 (3), 235–268.

Petkovšek, M., 1992. Hypergeometric solutions of linear recurrences with polynomial coefficients. J. Symb. Comp. 14 (2-3), 243–264.

Rabiner, L. R., Schafer, R. W., Rader, C. M., 1969. The chirp z-transform algorithm and its application. Bell System Tech. J. 48, 1249–1292.

Roman, S., 1984. The umbral calculus. Vol. 111 of Pure and Applied Mathematics. Academic Press Inc., New York.

Rothe, H. A., 1793. Formulae de serierum reversione demonstratio universalis signis localibus combinatorico-analyticorum vicariis exhibita. Leipzig.

Schoenberg, I. J., 1981. On polynomial interpolation at the points of a geometric progression. Proc. Roy. Soc. Edinburgh Sect. A 90 (3-4), 195–207.

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

Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation großer Zahlen. Computing 7, 281–292.

Shoup, V., 1996–2004. NTL: A library for doing number theory. `http://www.shoup.net`.

Stirling, J., 1730. Methodus Differentialis: sive Tractatus de Summatione et Interpolatione Serierum Infinitarum. Gul. Bowyer, London, english translation by Holliday, J. The Differential Method: A Treatise of the Summation

and Interpolation of Infinite Series. 1749.
Storjohann, A., 2002. High-order lifting. In: ISSAC'02. ACM Press, pp. 246–254.
Strassen, V., 1973. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. Numer. Math. 20, 238–251.
Tellegen, B., 1952. A general network theorem, with applications. Tech. Rep. 7, Philips Research.
Thomé, É., 2001. Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm. In: ISSAC'01. ACM Press, pp. 323–331.
Thomé, É., 2002. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. J. Symb. Comp. 33 (5), 757–775.
Villard, G., 1996. Computing Popov and Hermite forms of polynomial matrices. In: ISSAC'96. ACM Press, pp. 251–258.