Kaltofen's division-free determinant algorithm differentiated for matrix adjoint computation

Gilles Villard

CNRS, Université de Lyon, INRIA

Laboratoire LIP, ENSL, 46, Allée d'Italie, 69364 Lyon Cedex 07, France

Abstract

Kaltofen has proposed a new approach in (Kaltofen, 1992) for computing matrix determinants without divisions. The algorithm is based on a baby steps/giant steps construction of Krylov subspaces, and computes the determinant as the constant term of a characteristic polynomial. For matrices over an abstract ring, by the results of Baur and Strassen (1983), the determinant algorithm, actually a straight-line program, leads to an algorithm with the same complexity for computing the adjoint of a matrix. However, the latter adjoint algorithm is obtained by the reverse mode of automatic differentiation, hence somehow is not "explicit". We present an alternative (still closely related) algorithm for the adjoint that can be implemented directly, we mean without resorting to an automatic transformation. The algorithm is deduced by applying program differentiation techniques "by hand" to Kaltofen's method, and is completely decribed. As subproblem, we study the differentiation of programs that compute minimum polynomials of lineraly generated sequences, and we use a lazy polynomial evaluation mechanism for reducing the cost of Strassen's avoidance of divisions in our case.

Key words: matrix determinant, matrix adjoint, matrix inverse, characteristic polynomial, exact algorithm, division-free complexity, Wiedemann algorithm, automatic differentiation.

1. Introduction

Kaltofen has proposed in (Kaltofen, 1992) a new approach for computing matrix determinants. This approach has brought breakthrough ideas for improving the complexity estimate for the problem of computing the determinant without divisions over an abstract ring (see (Kaltofen, 1992; Kaltofen and Villard, 2004)). With these foundations, the algorithm of Kaltofen and Villard (2004) computes the determinant in $O(n^{2.7})$ additions,

⋆ This research was partly supported by the French National Research Agency, ANR Gecko.
Email address: `Gilles.Villard@ens-lyon.fr` (Gilles Villard).
URL: `http://perso.ens-lyon.fr/gilles.villard` (Gilles Villard).

subtractions, and multiplications. The same ideas also lead to the currently best known bit complexity estimate of Kaltofen and Villard (2004) for the problem of computing the characteristic polynomial.

We consider the straigth-line programs of Kaltofen (1992) for computing the determinant over abstract fields or rings (with or without divisions). Using the reverse mode of automatic differentiation (see Linnainmaa (1970, 1976), and (Ostrowski et al., 1971)), a straight-line program for computing the determinant of a matrix A can be (automatically) transformed into a program for computing the adjoint matrix A^* of A. This principle, stated by Baur and Strassen (1983, Cor. 5), is also applied by Kaltofen (1992, Sec. 1.2) for computing A^*. Since the adjoint program is derived by an automatic process, few is known about the way it computes the adjoint. The only available information seems to be the determinant program itself, and the knowledge we have on the differentiation process. Neither the adjoint program can be described, or implemented, without resorting to an automatic differentiation tool.

In this paper, by studying the differentiation of Kaltofen's determinant algorithm step by step, we produce an "explicit" adjoint algorithm. The determinant algorithm, that we first recall in Section 2 over an abstract field K, uses a Krylov subspace construction, hence mainly reduces to vector times matrix, and matrix times matrix products. Another operation involved is computing the minimum polynomial of a linearly generated sequence. We apply the program differentiation mechanism, reviewed in Section 3, to the different steps of the determinant program in Section 4. This leads us to the description of a corresponding new adjoint program over a field, in Section 5. The algorithm we obtain somehow calls to mind the matrix factorization of Eberly (1997, (3.4)). We note that our objectives are similar to Eberly's ones, whose question was to give an explicit inversion algorithm from the parallel determinant algorithm of Kaltofen and Pan (1991).

Our motivation for studying the differentiation and resulting adjoint algorithm, is the importance of the determinant approach of Kaltofen (1992), and Kaltofen and Villard (2004), for various complexity estimates. Recent advances around the determinant of polynomial or integer matrices (see Eberly et al. (2000); Kaltofen and Villard (2004); Storjohann (2003, 2005)), and matrix inversion (see Jeannerod and Villard (2006), and Storjohann (2008)) also justify the study of the general adjoint problem.

For computing the determinant without divisions over a ring R, Kaltofen applies the avoidance of divisions of Strassen (1973) to his determinant algorithm over a field. We apply the same strategy for the adjoint. From the algorithm of Section 5 over a field, we deduce an adjoint algorithm over an arbitrary ring R in Section 6. The avoidance of divisions involves computations with truncated power series. A crucial point in Kaltofen's approach is a "baby steps/giant steps" scheme for reducing the corresponding power series arithmetic cost. However, since we use the reverse mode of differentiation, the flow of computation is modified, and the benefit of the baby steps/giant steps is partly lost for the adjoint. This asks us to introduce an early, and lazy polynomial evaluation strategy for not increasing the complexity estimate.

The division-free determinant algorithm of Kaltofen (1992) uses $O\tilde{~}(n^{3.5})$ operations in R. The adjoint algorithm we propose has essentially the same cost. Our study may be seen as a first step for the differentiation of the more efficient algorithm of Kaltofen and Villard (2004). The latter would require, in particular, to consider asymptotically fast matrix multiplication algorithms that are not discussed in what follows.

Especially in our matrix context, we note that interpreting programs obtained by automatic differentiation, may have connections with the interpretation of programs derived using the transposition principle. We refer for instance to the discussion of Kaltofen (2000, Sec. 6).

Cost functions. We let $\mathsf{M}(n)$ be such that two univariate polynomials of degree n over an arbitrary ring R can be multiplied using $\mathsf{M}(n)$ operations in R. The algorithm of Cantor and Kaltofen (1991) allows $\mathsf{M}(n) = O(n \log n \log\log n)$. The function $O(\mathsf{M}(n))$ also measures the cost of truncated power series arithmetic over R. For bounding the cost of polynomial gcd-type computations over a commutative field K we define the function G. Let $\mathsf{G}(n)$ be such that the extended gcd problem (see (von zur Gathen and Gerhard, 1999, Chap. 11)) can be solved with $\mathsf{G}(n)$ operations in K for polynomials of degree $2n$ in $\mathsf{K}[x]$. The recursive Knuth/Schönhage half-Gcd algorithm (see (Knuth, 1970; Schönhage, 1971; Moenck, 1973)) allows $\mathsf{G}(n) = O(\mathsf{M}(n) \log n)$. The minimum polynomial of degree n, of a linearly generated sequence given by its first $2n$ terms, can be computed in $\mathsf{G}(n) + O(n)$ operations (see (von zur Gathen and Gerhard, 1999, Algorithm 12.9)). We will often use the notation $O\tilde{}$ that indicates missing factors of the form $\alpha(\log n)^{\beta}$, for two positive real numbers α and β.

2. Kaltofen's determinant algorithm over a field

Kaltofen's determinant algorithm extends the Krylov-based method of Wiedemann (1986). The latter approach is successful in various situations. We refer especially to the algorithms of Kaltofen and Pan (1991) and Kaltofen and Saunders (1991) around exact linear system solution that has served as basis for subsequent works. We may also point out the various questions investigated by Chen et al. (2002), and references therein.

Let K be a commutative field. We consider $A \in \mathsf{K}^{n \times n}$, $u \in \mathsf{K}^{1 \times n}$, and $v \in \mathsf{K}^{n \times 1}$. We introduce the Hankel matrix $H = \left(uA^{i+j-2}v\right)_{1 \le i,j \le n} \in \mathsf{K}^{n \times n}$, and let $h_k = uA^k v$ for $0 \le k \le 2n-1$. We also assume that H is non-singular:

$$\det H = \det \begin{bmatrix} uv & uAv & \ldots & uA^{n-1}v \\ uAv & uA^2v & \ldots & uA^n v \\ \vdots & \ddots & \vdots & \vdots \\ uA^{n-1}v & \ldots & \ldots & uA^{2n-2}v \end{bmatrix} \neq 0. \tag{1}$$

In the applications, (1) is ensured either by construction of A, u, and v, as in (Kaltofen, 1992; Kaltofen and Villard, 2004), or by randomization (see the above cited references around Wiedemann's approach, and (Kaltofen, 1992; Kaltofen and Villard, 2004)).

One of the key ideas of Kaltofen (1992) for reducing the division-free complexity estimate for computing the determinant, is to introduce a "baby steps/giant steps" behaviour in the Krylov subspace construction. With baby steps/giant steps parameters $r = \lceil 2n/s \rceil$ and $s = \lceil \sqrt{n} \rceil$ $(rs \ge 2n)$ we consider the following algorithm.

Algorithm DET

Input: $A \in \mathsf{K}^{n\times n}, u \in \mathsf{K}^{1\times n}, v \in \mathsf{K}^{n\times 1}$

STEP I. $v_0 := v$; For $i = 1, \ldots, r-1$ do $v_i := Av_{i-1}$
STEP II. $B := A^r$
STEP III. $u_0 := u$; For $j = 1, \ldots, s-1$ do $u_j := u_{j-1}B$
STEP IV. For $i = 0, 1, \ldots, r-1$ do
    For $j = 0, 1, \ldots, s-1$ do $h_{i+jr} := u_j v_i$
STEP V. $f :=$ the minimum polynomial of $\{h_k\}_{0\le k\le 2n-1}$

Output: $\det A := (-1)^n f(0)$.

We ommit the proof of next theorem that establishes the correctness and the cost of Algorithm DET, and refer to Kaltofen (1992). We may simply note that the sequence $\{h_k\}_{0\le k\le 2n-1}$ is linearly generated. In addition, if (1) is true, then the minimum polynomial f of $\{h_k\}_{0\le k\le 2n-1}$, the minimum polynomial of A, and the characteristic (monic) polynomial of A coincide. Hence $(-1)^n f(0)$ is equal to the determinant of A. Via an algorithm that can multiply two matrices of $\mathsf{K}^{n\times n}$ in $O(n^\omega)$ we have:

Theorem 1. *If $A \in \mathsf{K}^{n\times n}$, $u \in \mathsf{K}^{1\times n}$, and $v \in \mathsf{K}^{n\times 1}$ satisfy (1), then Algorithm* DET *computes the determinant of A in $O(n^\omega \log n)$ operations in* K.

For the matrix product we may set $\omega = 3$, or $\omega = 2.376$ using the algorithm of Coppersmith and Winograd (1990). In the rest of the paper we work with a cubic matrix multiplication algorithm. Our study has to be generalized if fast matrix multiplication is introduced.

3. Backward automatic differentiation

The determinant of $A \in \mathsf{K}^{n\times n}$ is a polynomial Δ in $\mathsf{K}[a_{1,1}, \ldots, a_{i,j}, \ldots, a_{n,n}]$ of the entries of A. We denote the adjoint matrix by A^* such that $AA^* = A^*A = (\det A)I$. As noticed by Baur and Strassen (1983), the entries of A^* satisfy

$$a^*_{j,i} = \frac{\partial \Delta}{\partial a_{i,j}}, 1 \le i, j \le n. \tag{2}$$

The reverse mode of automatic differentiation allows to transform a program which computes Δ into a program which computes all the partial derivatives in (2). Among the rich literature about the reverse mode of automatic differentiation we may refer to the seminal works of Linnainmaa (1970, 1976) and Ostrowski et al. (1971). For deriving the adjoint program from the determinant program we follow the lines of Baur and Strassen (1983) and Morgenstern (1985).

Algorithm DET is a straight-line program over K. For a comprehensive study of straight-line programs for instance see (Bürgisser et al., 1997, Chapter 4). We assume that the entries of A are stored initially in n^2 variables δ_i, $-n^2 < i \le 0$. Then we assume that the algorithm is a sequence of arithmetic operations in K, or assignments to constants of K. Let L be the number of such operations. We assume that the result of each instruction is stored in a new variable δ_i, hence the algorithm is seen as a sequence of instructions

$$\delta_i := \delta_j \text{ op } \delta_k,\ \text{op} \in \{+,-,\times,\div\},\ \ -n^2 < j,k < i, \tag{3}$$

or

$$\delta_i := c,\ c \in \mathsf{K}, \tag{4}$$

for $1 \le i \le L$. Note that a binary arithmetic operation (3) where one of the operands is a constant of K can be implemented with the aid of (4). For any $0 \le i \le L$, the determinant maybe be seen as a rational function Δ_i of $\delta_{-n^2+1}, \ldots, \delta_i$, such that

$$\Delta_0(\delta_{-n^2+1}, \ldots, \delta_0) = \Delta(a_{1,1}, \ldots, a_{n,n}), \tag{5}$$

and such that the last instruction gives the result:

$$\det A = \delta_L = \Delta_L(\delta_{-n^2+1}, \ldots, \delta_L). \tag{6}$$

The reverse mode of automatic differentiation computes the derivatives (2) in a backward recursive way, from the derivatives of (6) to those of (5). Using (6) we start the recursion with

$$\frac{\partial \Delta_L}{\partial \delta_L} = 1,\ \frac{\partial \Delta_L}{\partial \delta_l} = 0,\ \ -n^2 < l \le L-1.$$

Then, writing

$$\Delta_{i-1}(\delta_{-n^2+1}, \ldots, \delta_{i-1}) = \Delta_i(\delta_{-n^2+1}, \ldots, \delta_i) = \Delta_i(\delta_{-n^2+1}, \ldots, g(\delta_j, \delta_k)), \tag{7}$$

where g is given by (3) or (4), we have

$$\frac{\partial \Delta_{i-1}}{\partial \delta_l} = \frac{\partial \Delta_i}{\partial \delta_l} + \frac{\partial \Delta_i}{\partial \delta_i}\frac{\partial g}{\partial \delta_l},\ \ -n^2 < l \le i-1, \tag{8}$$

for $1 \le i \le L$. Depending on g several cases may be examined. For instance, for an addition $\delta_i := g(\delta_k, \delta_j) = \delta_k + \delta_j$, (8) becomes

$$\frac{\partial \Delta_{i-1}}{\partial \delta_k} = \frac{\partial \Delta_i}{\partial \delta_k} + \frac{\partial \Delta_i}{\partial \delta_i}, \quad \frac{\partial \Delta_{i-1}}{\partial \delta_j} = \frac{\partial \Delta_i}{\partial \delta_j} + \frac{\partial \Delta_i}{\partial \delta_i}, \tag{9}$$

with the other derivatives ($l \ne k$ or j) remaining unchanged. In the case of a multiplication $\delta_i := g(\delta_k, \delta_j) = \delta_k \times \delta_j$, (8) gives that the only derivatives that are modified are

$$\frac{\partial \Delta_{i-1}}{\partial \delta_k} = \frac{\partial \Delta_i}{\partial \delta_k} + \frac{\partial \Delta_i}{\partial \delta_i}\,\delta_j, \quad \frac{\partial \Delta_{i-1}}{\partial \delta_j} = \frac{\partial \Delta_i}{\partial \delta_j} + \frac{\partial \Delta_i}{\partial \delta_i}\,\delta_k. \tag{10}$$

We see for instance in (10), where δ_j is used for updating the derivative with respect to δ_k, that the recursion uses intermediary results of the determinant algorithm. For the adjoint algorithm, we will assume that the determinant algorithm has been executed once, and that the δ_i's are stored in $n^2 + L$ memory locations.

Recursion (8) gives a practical mean, and a program, for computing the $N = n^2$ derivatives of Δ with respect to the $a_{i,j}$'s. For any rational function Q in N variables $\delta_{-N+1}, \ldots, \delta_0$ the corresponding general statement is:

Theorem 2. *[Baur and Strassen (1983)] Let $\mathcal{P}$ be a straight-line program computing Q in L operations in K. One can derive an algorithm $\partial\mathcal{P}$ that computes Q and the N partial derivatives $\partial Q/\partial \delta_l$ in less than $5L$ operations in K.*

Combining Theorem 2 with Theorem 1 gives the construction of an algorithm ∂DET for computing the adjoint matrix A^* (see (Baur and Strassen, 1983, Corollary 5)). The algorithm can be generated automatically via an automatic differentiation tool [1]. However, it seems unclear how it could be programmed directly, and, to our knowledge, it has no interpretation of its own.

4. Differentiating the determinant algorithm over a field

We apply the backward recursion (8) to Algorithm DET of Section 2 for deriving the algorithm ∂DET. We assume that A is non-singular, hence A^* is non-trivial. By construction, the flow of computation for the adjoint is reversed compared to the flow of Algorithm DET, therefore we start with the differentiation of STEP V.

4.1. Differentiation of the minimum polynomial constant term computation

At STEP V, Algorithm DET computes the minimum polynomial f of the linearly generated sequence $\{h_k\}_{0\le k\le 2n-1}$. Let λ be the first instruction index at which all the h_k's are known. We apply the recursion until step λ, globally, we mean that we compute the derivatives of Δ_λ. After the instruction λ, the determinant is viewed as a function Δ_{V} of the h_k's only. Following (7) we have

$$\det(A) = \Delta_\lambda(\delta_{-n^2+1}, \ldots, \delta_\lambda) = \Delta_{\text{V}}(h_1, \ldots, h_{2n-1}).$$

Hence we may focus on the derivatives $\partial\Delta_{\text{V}}/\partial h_k$, $0 \le k \le 2n-1$, the remaining ones are zero.

Using assumption (1) we know that the minimum polynomial f of $\{h_k\}_{0\le k\le 2n-1}$ has degree n, and if $f(x) = f_0 + f_1x + \ldots + f_{n-1}x^{n-1} + x^n$, then f satisfies

$$H \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & \ldots & h_{n-1} \\ h_1 & h_2 & \ldots & h_n \\ \vdots & \ddots & \vdots & \vdots \\ h_{n-1} & \ldots & \ldots & h_{2n-2} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} = - \begin{bmatrix} h_n \\ h_{n+1} \\ \vdots \\ h_{2n-1} \end{bmatrix} \tag{11}$$

see, e.g., (Kaltofen, 1992), or (von zur Gathen and Gerhard, 1999, Algorithm 12.9) together with (Brent et al., 1980). Applying Cramer's rule we see that

$$f_0 = (-1)^n \det \begin{bmatrix} h_1 & h_2 & \ldots & h_n \\ h_2 & h_3 & \ldots & h_{n+1} \\ \vdots & \ddots & \vdots & \vdots \\ h_n & \ldots & \ldots & h_{2n-1} \end{bmatrix} / \det H,$$

hence, defining $H_A = \left(uA^{i+j-1}v\right)_{1\le i,j\le n} = \left(h_{i+j-1}\right)_{1\le i,j\le n} \in \mathsf{K}^{n\times n}$, we obtain

$$\Delta_{\text{V}} = \frac{\det H_A}{\det H}. \tag{12}$$

[1] We refer for instance to `http://www.autodiff.org`

Let $\tilde{\mathcal{K}}_u$ and $\mathcal{K}_v$ be the Krylov matrices

$$\tilde{\mathcal{K}}_u = [u^T, A^T u^T, \ldots, (A^T)^{n-1} u^T]^T \in \mathsf{K}^{n\times n}, \tag{13}$$

and

$$\mathcal{K}_v = [v, Av, \ldots, A^{n-1}v] \in \mathsf{K}^{n\times n}. \tag{14}$$

Since $H = \tilde{\mathcal{K}}_u \mathcal{K}_v$, assumption (1) implies that both $\tilde{\mathcal{K}}_u$ and $\mathcal{K}_v$ are non-singular. Hence, using that A is non-singular, we note that $H_A = \tilde{\mathcal{K}}_u A \mathcal{K}_v$ also is non-singular.

For differentiating (12), let us first specialize (2) to Hankel matrices. We denote by $(\partial\Delta/\partial a_{i,j})(H)$ the substitution of the $a_{i,j}$'s for the entries of H in $\partial\Delta/\partial a_{i,j}$, for $1 \le i,j \le n$. From (2) we have

$$h^*_{j,i} = \frac{\partial \Delta}{\partial a_{i,j}}(H), 1 \le i,j \le n.$$

Since the entries of H are constant along the anti-diagonals, we deduce that

$$\frac{\partial \det H}{\partial h_k} = \sum_{i+j-2=k} \frac{\partial \Delta}{\partial a_{i,j}}(H) = \sum_{i+j-2=k} h^*_{j,i} = \sum_{i+j-2=k} h^*_{i,j},\ 0 \le k \le 2n-2.$$

In other words, we may write

$$\frac{\partial \det H}{\partial h_k} = \sigma_k(H^*),\ 0 \le k \le 2n-1, \tag{15}$$

where, for a matrix $M = (m_{ij})$, we define

$$\sigma_k(M) = 0 + \sum_{i+j-2=k} m_{ij},\ 1 \le i,j \le n.$$

The function $\sigma_k(M)$ is the sum of the entries in the anti-diagonal of M starting with $m_{1,k+1}$ if $0 \le k \le n-1$, and $m_{k-n+2,n}$ if $n \le k \le 2n-2$. Shifting the entries of H for obtaining H_A we also have

$$\frac{\partial \det H_A}{\partial h_k} = \sigma_{k-1}(H^*_A),\ 0 \le k \le 2n-1. \tag{16}$$

Now, differentiating (12), together with (15) and (16), leads to

$$\frac{\partial \Delta_{\mathrm{V}}}{\partial h_k} = \frac{(\partial \det H_A/\partial h_k)}{\det H} - \frac{(\partial \det H/\partial h_k)}{\det H}\frac{\det H_A}{\det H} = \frac{(\partial \det H_A/\partial h_k)}{\det H_A}\frac{\det H_A}{\det H} - \sigma_k(H^{-1})\Delta_{\mathrm{V}}$$

and, consequently, to

$$\frac{\partial \Delta_{\mathrm{V}}}{\partial h_k} = \left(\sigma_{k-1}(H_A^{-1}) - \sigma_k(H^{-1})\right)\Delta_{\mathrm{V}},\ 0 \le k \le 2n-1. \tag{17}$$

With (17) we identify the problem solved by the first step of the ∂Det algorithm, and provide first informations for interpreting or implementing the adjoint program. Various algorithms may be used for computing the minimum polynomial (for instance see (von zur Gathen and Gerhard, 1999, Algorithm 12.9)), that will lead to corresponding algorithms for computing the left sides in (17). However, we will not discuss these aspects, since the associated costs are not dominant in the overall complexity.

We have recalled, in the introduction, that the minimum polynomial f (its constant term $f(0)$) can be computed from the h_k's in $\mathsf{G}(n) + O(n)$ operations in K. Hence Theorem 2 gives an algorithm for computing the derivatives using $5\mathsf{G}(n) + O(n)$ operations.

Alternatively, in the Appendix we propose a direct approach that takes advantage of (17). Proposition 4 shows that if f, H, and H_A are given, then the $\partial\Delta_{\text{V}}/\partial h_k$'s can be computed in $\mathsf{G}(n) + O(\mathsf{M}(n))$ operations in K.

4.2. Differentiation of the dot products

For differentiating STEP IV, Δ is seen as a function Δ_{IV} of the u_j's and v_i's. The entries of u_j are used for computing the r scalars $h_{jr}, h_{1+jr}, \ldots, h_{(r-1)+jr}$ for $0 \le j \le s-1$. The entries of v_i are involved in the computation of the s scalars $h_i, h_{i+r}, \ldots, h_{i+(s-1)r}$ for $0 \le i \le r-1$.

In (8), the new derivative $\partial\Delta_{i-1}/\partial\delta_l$ is obtained by adding the current instruction contribution to the previously computed derivative $\partial\Delta_i/\partial\delta_l$. Since all the h_{i+jr}'s are computed independently according to

$$h_{i+jr} = \sum_{l=0}^{n} (u_j)_l (v_i)_l,$$

it follows that the derivative of Δ_{IV} with respect to an entry $(u_j)_l$ or $(v_i)_l$ is obtained by summing up the contributions of the multiplications $(u_j)_l(v_i)_l$. We obtain

$$\frac{\partial\Delta_{\text{IV}}}{\partial(u_j)_l} = \sum_{i=0}^{r-1} \frac{\partial\Delta_{\text{V}}}{\partial h_{i+jr}} (v_i)_l, \; 0 \le j \le s-1, \; 1 \le l \le n, \tag{18}$$

and

$$\frac{\partial\Delta_{\text{IV}}}{\partial(v_i)_l} = \sum_{i=0}^{s-1} \frac{\partial\Delta_{\text{V}}}{\partial h_{i+jr}} (u_j)_l, \; 0 \le i \le r-1, \; 1 \le l \le n. \tag{19}$$

By abuse of notations (of the sign ∂), we let ∂u_j be the $n \times 1$ vector, respectively ∂v_i be the $1 \times n$ vector, whose entries are the derivatives of Δ_{IV} with respect to the entries of u_j, respectively v_i. Note that because of the index transposition in (2), it is convenient, here and in the following, to take the transpose form (column versus row) for the derivative vectors. Defining also

$$\partial H = \left(\frac{\partial\Delta_{\text{V}}}{\partial h_{i+jr}}\right)_{0\le i\le r-1, \; 0\le j\le s-1} \in \mathsf{K}^{r\times s},$$

we deduce, from (18) and (19), that

$$[\partial u_0, \partial u_1, \ldots, \partial u_{s-1}] = [v_0, v_1, \ldots, v_{r-1}]\, \partial H \in \mathsf{K}^{n\times s}. \tag{20}$$

and

$$\begin{bmatrix} \partial v_0 \\ \partial v_1 \\ \vdots \\ \partial v_{r-1} \end{bmatrix} = \partial H \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{s-1} \end{bmatrix} \in \mathsf{K}^{r\times n}. \tag{21}$$

Identities (20) and (21) give the second step of the adjoint algorithm. In Algorithm DET, STEP IV costs essentially $2rsn$ additions and multiplications in K. Here we have essentially $4rsn$ additions and multiplications using basic loops (as in STEP IV) for calculating the matrix products, we mean without an asymptotically fast matrix multiplication algorithm.

4.3. Differentiation of the matrix times vector and matrix products

The recursive process for differentiating STEP III to STEP I may be written in terms of the differentiation of the basic operation (or its transposed operation)

$$q := p \cdot M \in \mathsf{K}^{1\times n}, \tag{22}$$

where p and q are row vectors of dimension n, and M is an $n \times n$ matrix. We assume at this point (by construction of the recursion) that column vectors ∂p and ∂q of derivatives of the determinant with respect to the entries of p and q, are available. For instance, for differentiating STEP III, we will consider the ∂u_j's. We also assume that an $n \times n$ matrix ∂M, whose transpose gives the derivatives with respect to the m_{ij}'s, has been computed. Initially, for STEP III, we will take $\partial B = 0$.

Following the lines of previous section for obtaining (20) and (21), we see that differentiating (22) amounts to updating ∂p and ∂M according to

$$\begin{cases} \partial p := \partial p + M \cdot \partial q \in \mathsf{K}^{n}, \\ \partial M := \partial M + \partial q \cdot p \in \mathsf{K}^{n\times n}. \end{cases} \tag{23}$$

Starting from the values of the ∂u_j's computed with (20), and from $\partial B = 0$, for the differentiation of STEP III, (23) gives

$$\begin{cases} \partial u_{j-1} := \partial u_{j-1} + B \cdot \partial u_j, \\ \partial B := \partial B + \partial u_j \cdot u_{j-1},\ j = s-1, \ldots, 1. \end{cases} \tag{24}$$

For STEP II, we mean $B := A^r$, we show that the backward recursion leads to

$$\partial A := \sum_{k=1}^{r} A^{r-k} \cdot \partial B \cdot A^{k-1}. \tag{25}$$

Here, the notation ∂A stands for the $n \times n$ matrix whose transpose gives the derivatives $\partial \Delta_{\mathrm{II}} / \partial a_{i,j}$. We may show (25) by induction on r. For $r = 1$, $\partial A = \partial B$ is true. If (25) is true for $r-1$, then let $C = A^{r-1}$ and $B = CA$. Using (23), and overloading the notation ∂A, we have

$$\begin{cases} \partial C = A \cdot \partial B \in \mathsf{K}^{n\times n}, \\ \partial A = \partial B \cdot C \in \mathsf{K}^{n\times n}. \end{cases}$$

Hence, using (25) for $r-1$, we establish that

$$\begin{aligned} \partial A &= \partial A + \textstyle\sum_{k=1}^{r-1} A^{r-k-1} \cdot \partial C \cdot A^{k-1}, \\ &= \partial B \cdot C + \textstyle\sum_{k=1}^{r-1} A^{r-k-1} \cdot (A \cdot \partial B) \cdot A^{k-1} \\ &= \partial B \cdot A^{r-1} + \textstyle\sum_{k=1}^{r-1} A^{r-k} \cdot \partial B \cdot A^{k-1} = \textstyle\sum_{k=1}^{r} A^{r-k} \cdot \partial B \cdot A^{k-1}. \end{aligned}$$

Any specific approach for computing A^r will lead to an associated program for computing ∂A. Let us look, in particular, at the case where STEP II of Algorithm DET is implemented by repeated squaring, in essentially $\log_2 r$ matrix products. Consider the recursion

$$\begin{aligned}
&A_0 := A\\
&\text{For } k = 1, \ldots, \log_2 r \text{ do } A_{2^k} := A_{2^{k-1}} \cdot A_{2^{k-1}}\\
&B := A_r
\end{aligned}$$

that computes $B := A^r$. The associated program for computing the derivatives is

$$\begin{aligned}
&\partial A_r := \partial B\\
&\text{For } k = \log_2 r, \ldots, 1 \text{ do } \partial A_{2^{k-1}} := A_{2^{k-1}} \cdot \partial A_{2^k} + \partial A_{2^k} \cdot A_{2^{k-1}}\\
&\partial A := \partial A_0,
\end{aligned} \tag{26}$$

and costs essentially $2 \log_2 r$ matrix products.

From the values of the ∂v_i's computed with (21), we finally differentiate STEP I, and update ∂A according to

$$\begin{cases} \partial v_{i-1} := \partial v_{i-1} + \partial v_i \cdot A, \\ \partial A := \partial A + v_{i-1} \cdot \partial v_i, \;\; i = r-1, \ldots, 1. \end{cases} \tag{27}$$

Now, ∂A is the $n \times n$ matrix whose transpose gives the derivatives $\partial \Delta_{\mathrm{I}} / \partial a_{i,j} = \partial \Delta / \partial a_{i,j}$, hence from (2) we know that $A^* = \partial A$.

STEP III and STEP I both cost essentially r ($\approx s$) matrix times vector products. From (24) and (27) the differentiated steps both require r matrix times vector products, and $2rn^2 + O(rn)$ additional operations in K.

5. The adjoint algorithm over a field

We call ADJOINT the algorithm obtained from the successive differentiations of Section 4. Algorithm ADJOINT is detailed below. We keep the notations of previous sections. We use in addition $U \in \mathsf{K}^{s \times n}$ and $V \in \mathsf{K}^{n \times r}$ (resp. $\partial U \in \mathsf{K}^{n \times s}$ and $\partial V \in \mathsf{K}^{r \times n}$) for the right sides (resp. the left sides) of (20) and (21).

The cost of ADJOINT is dominated by STEP IV*, which is the differentiation of the matrix power computation. As we have seen with (26), the number of operation is essentially twice as much as for Algorithm DET. The code we give allows an easy implementation.

We note that if the product by $\det A$ is avoided in STEP I*, then the algorithm computes the matrix inverse A^{-1}. We may put this into perspective with the algorithm given by Eberly (1997). With $\tilde{\mathcal{K}}_u$ and $\mathcal{K}_v$ the Krylov matrices of (13) and (14), Eberly has proposed a processor-efficient inversion algorithm based on

$$A^{-1} = \mathcal{K}_v H_A^{-1} \tilde{\mathcal{K}}_u. \tag{28}$$

To see whether a baby steps/giant steps version of (28) would lead to an algorithm similar to ADJOINT deserves further investigations.

Algorithm Adjoint (∂Det)

Input: $A \in \mathsf{K}^{n\times n}$ non-singular, and the intermediary data of Algorithm Det

All the derivatives are initialized to zero

step i*. /* *Requires the Hankel matrices H and H_A, see (17)* */
$\partial\Delta_{\mathrm{V}}/\partial h_k := \big(\sigma_{k-1}(H_A^{-1}) - \sigma_k(H^{-1})\big)\det A,\ 0 \le k \le 2n-1$

step ii*. /* *Requires the u_j's and v_i's, see (20) and (21)* */
$\partial U := V \cdot \partial H$
$\partial V := \partial H \cdot U$

step iii*. /* *Requires $B = A^r$, see (24)* */
For $j = s-1, \ldots, 1$ do
$\partial u_{j-1} := \partial u_{j-1} + B \cdot \partial u_j$
$\partial B := \partial B + \partial u_j \cdot u_{j-1}$

step iv*. /* *Requires the powers of A, see (25) or (26)* */
$A^* := \sum_{k=1}^{r} A^{r-k} \cdot \partial B \cdot A^{k-1}$

step v*. /* *See (27)* */
For $i = r-1, \ldots, 1$ do
$\partial v_{i-1} := \partial v_{i-1} + \partial v_i \cdot A$
$A^* := A^* + v_{i-1} \cdot \partial v_i$

Output: The adjoint matrix $A^* \in \mathsf{K}^{n\times n}$.

6. Application to computing the adjoint without divisions

Now let A be an $n \times n$ matrix over an abstract ring R. Kaltofen's algorithm for computing the determinant of A without divisions applies Algorithm Det on a well chosen univariate polynomial matrix $Z(z) = C + z(A - C)$ where $C \in \mathbb{Z}^{n\times n}$, with a dedicated choice of projections $u = \varphi \in \mathbb{Z}^{1\times n}$ and $v = \psi \in \mathbb{Z}^{n\times 1}$. The algorithm uses Strassen's avoidance of divisions (see (Strassen, 1973; Kaltofen, 1992)). Since the determinant of Z is a polynomial of degree n in z, the arithmetic operations over K in Det may be replaced by operations on power series in $\mathsf{R}[[z]]$ modulo z^{n+1}. Once the determinant of $Z(z)$ is computed, the evaluation $(\det Z)(1) = \det(C + 1 \times (A - C))$ gives the determinant of A. The choice of C, φ and ψ is such that, whenever a division by a truncated power series is performed the constant coefficients are ± 1. Therefore the algorithm necessitates no divisions. Note that, by construction of $Z(z)$, the constant terms of the power series involved when Det is called with inputs $Z(z), \varphi$ and ψ, are the intermediary values computed by Det with inputs C, φ and ψ.

The cost for computing the determinant of A without divisions is then deduced as follows. In step i and step ii of Algorithm Det applied to $Z(z)$, the vector and matrix entries are polynomials of degree $O(\sqrt{n})$. The cost of step ii dominates, and is

$O(n^3\mathsf{M}(\sqrt{n})\log n) = O\tilde{}(n^3\sqrt{n})$ operations in R. STEP III, IV, and V cost $O(n^2\sqrt{n})$ operations on power series modulo z^{n+1}, that is $O(n^2\mathsf{M}(n)\sqrt{n})$ operations in R. Hence $\det Z(z)$ is computed in $O\tilde{}(n^3\sqrt{n})$ operations in R, and $\det A$ is obtained with the same cost bound.

An main property of Kaltofen's approach (which also holds for the improved blocked version of Kaltofen and Villard (2004)), is that the scalar value $\det A$ is obtained via the computation of the polynomial value $\det Z(z)$. This property seems to be lost with the adjoint computation. We are going to see how Algorithm ADJOINT applied to $Z(z)$ allows to compute $A^* \in \mathsf{R}^{n\times n}$ in time $O\tilde{}(n^3\sqrt{n})$ operations in R, but does not seem to allow the computation of $Z^*(z) \in \mathsf{R}[z]^{n\times n}$ with the same complexity estimate. Indeed, a key point in Kaltofen's approach for reducing the overall complexity estimate, is to compute with small degree polynomials (degree $O(\sqrt{n})$) in STEP I and STEP II. However, since the adjoint algorithm has a reversed flow, this point does not seem to be relevant for ADJOINT, where polynomials of degree n are involved from the beginning.

Our approach for computing A^* over R keeps the idea of running Algorithm ADJOINT with input $Z(z) = C + z(A - C)$, such that $Z^*(z)$ has degree less than n, and gives $A^* = Z^*(1)$. In Section 6.1, we verify that the implementation using Proposition 4, needs no divisions. We then show in Section 6.2 how to establish the cost estimate $O\tilde{}(n^3\sqrt{n})$. The principle we follow is to start evaluating polynomials at $z = 1$ as soon as computing with the entire polynomials is prohibitive.

6.1. Division-free Hankel matrix inversion and anti-diagonal sums

In Algorithm ADJOINT, divisions may only occur during the anti-diagonal sums computation. We verify here that with the matrix $Z(z)$, and the special projections $\varphi \in \mathbb{Z}^{1\times n}, \psi \in \mathbb{Z}^{n\times 1}$, the approach described in the Appendix for computing the anti-diagonal sums requires no divisions. Equivalently, since we use Strassen's avoidance of divisions, we verify that with the matrix C and the projections φ, ψ, the approach necessitates no divisions. As we are going to see, this a direct consequence of the construction of Kaltofen (1992).

Here we let $h_k = \varphi C^k \psi$ for $0 \le k \le 2n-1$, $a(x) = x^{2n}$, and $b(x) = h_0 x^{2n-1} + h_1 x^{2n-2} + \ldots + h_{2n-1}$. The extended Euclidean scheme with inputs a and b leads to a normal sequence, and after $n-1$ and n steps of the scheme, we get (see (Kaltofen, 1992, Sec. 2)):

$$s(x)a(x) + t(x)b(x) = c(x), \text{with } \deg s = n-2, \deg t = n-1, \deg c = n, \qquad (29)$$

and

$$\bar{s}(x)a(x) + \bar{t}(x)b(x) = \bar{c}(x), \text{with } \deg \bar{s} = n-1, \deg \bar{t} = n, \deg \bar{c} = n-1. \qquad (30)$$

The polynomial $\bar{t}$ is such that

$$\bar{t} = \pm x^n + \text{intermediate monomials } + 1 = \pm f, \qquad (31)$$

with f the minimum polynomial of $\{h_k\}_{0\le k\le 2n-1}$. One may check, in particular, that the n equations obtained by identifying the coefficients of degree $2n-1 \ge k \ge n$ in (30) give the linear system (11), that defines f. The polynomial c also has leading coefficient ± 1. By identifying the coefficients of degree $2n-1 \geq k \geq n$ in (29), we obtain:

$$H \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{n-1} \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & \ldots & h_{n-1} \\ h_1 & h_2 & \ldots & h_n \\ \vdots & \ddots & \vdots & \vdots \\ h_{n-1} & \ldots & \ldots & h_{2n-2} \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{n-1} \end{bmatrix} = \pm \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{32}$$

Therefore $t = \pm g$ with g the polynomial needed for computing (44)-(47), in addition to f. Since C, φ, and ψ are such that the extended Euclidean scheme necessitates no divisions (see (Kaltofen, 1992, Sec. 2)), we see that both f and g may be computed with no divisions. The only remaining division in the algorithm for Proposition 4 is at (38). From (31), this division is by $f_0 = 1$.

6.2. Lazy polynomial evaluation and division-free adjoint computation

We run Algorithm ADJOINT with input $Z(z) \in \mathsf{R}[z]^{n \times n}$, and start with operations on truncated power series modulo z^{n+1}. We assume that Algorithm DET has been executed, and that its intermediary results have been stored.

Using Proposition 4 and previous section, STEP I* requires $O(\mathsf{G}(n)\mathsf{M}(n)) = O\tilde{~}(n^2)$ operations in R for computing $\partial H(z)$ of degree n in $\mathsf{R}[z]^{r \times s}$. STEP II*, STEP III*, and V* cost $O(n^2\sqrt{n})$ operations in K, hence, taking into account the power series operations, this gives $O(n^2\mathsf{M}(n)\sqrt{n}) = O\tilde{~}(n^3\sqrt{n})$ operations in R for the division-free version. The cost analysis of STEP IV*, using (26) over power series modulo z^n, leads to $\log_2 r$ matrix products, hence to the time bound $O\tilde{~}(n^4)$, greater than the target estimate $O\tilde{~}(n^3\sqrt{n})$. As noticed previously, STEP III of Algorithm DET only involves polynomials of degree $O(\sqrt{n})$, while the reversed program for STEP IV* of Algorithm ADJOINT, relies on $\partial B(z)$ whose degree is n.

Since only $Z^*(1) = A^*$ is needed, our solution, for restricting the cost to $O\tilde{~}(n^3\sqrt{n})$, is to start evaluating at $z = 1$ during STEP IV*. However, since power series multiplications are done modulo z^n, this evaluation must be lazy. The fact that matrices $Z^k(z)$, $1 \leq k \leq r-1$, of degree at most $r-1$ are involved, enables the following. Let a and c be two polynomials such that $\deg a + \deg c = r-1$ in $\mathsf{R}[z]$, and let b be of degree $n \geq r-2$ in $\mathsf{R}[z]$. Considering the highest degree part of b, and evaluating the lowest degree part at $z = 1$, we define $b_H(z) = b_n z^{r-2} + \ldots + b_{n-r+2} \in \mathsf{R}[z]$ and $b_L = b_{n-r+1} + \ldots + b_0 \in \mathsf{R}$. We then remark that

$$\begin{aligned} \left(a(z)b(z)c(z) \bmod z^{n+1}\right)(1) &= \left(a(z)(b_H(z)z^{n-r+2} + b_L)c(z) \bmod z^{n+1}\right)(1), \\ &= \left(a(z)b_H(z)c(z) \bmod z^{r-1}\right)(1) + \left(a(z)b_L c(z)\right)(1). \end{aligned} \tag{33}$$

For modifying STEP IV*, we follow the definition of b_H and b_L, and first compute $\partial B_H(z) \in \mathsf{R}[z]^{n \times n}$ of degree $r-2$, and $\partial B_L \in \mathsf{R}^{n \times n}$. Applying (33), the sum $\sum_{k=1}^{r} Z^{r-k}(z) \cdot \partial B(z) \cdot Z^{k-1}(z)$ may then be evaluated at $z = 1$ by the program

$$\begin{aligned} \text{Modified STEP IV}^*. \quad Z^* &:= \left(\textstyle\sum_{k=1}^{r} Z^{r-k}(z) \cdot \partial B_H(z) \cdot Z^{k-1}(z) \bmod z^{r-1}\right)(1) \\ Z^* &:= Z^* + \left(\textstyle\sum_{k=1}^{r} Z^{r-k}(z) \cdot \partial B_L \cdot Z^{k-1}(z)\right)(1), \end{aligned} \tag{34}$$

in $\tilde{O}(n^3\mathsf{M}(r)) = \tilde{O}(n^3\sqrt{n})$ operations in R. This leads to an intermediary value $Z^* \in \mathsf{R}^{n\times n}$ before STEP V*. The value is updated at STEP V* with power series operations, and a final evaluation at $z = 1$ in time $\tilde{O}(n^2 r\mathsf{M}(n)) = \tilde{O}(n^3\sqrt{n})$. Since only STEP IV* has been modified, we obtain the following result.

Theorem 3. *Let $A \in \mathsf{R}^{n\times n}$. If Algorithm* ADJOINT*, modified according to (34), is executed with input $Z(z) = C + z(A - C)$, power series operations modulo z^{n+1}, and a final evaluation at $z = 1$, then the matrix adjoint A^* is computed in $\tilde{O}(n^3\sqrt{n})$ operations in* R.

7. Concluding remarks

We have developed an explicit algorithm for computing the matrix adjoint using only ring arithmetic operations. The algorithm has complexity estimate $\tilde{O}(n^{3.5})$. It represents a practical alternative to previously existing solutions for the problem, that rely on automatic differentiation of a determinant algorithm. Our description of the algorithm allows direct implementations. It should help understanding how the adjoint is computed using Kaltofen's baby steps/giant steps construction. Still, a full mathematical explanation deserves to be investigated. Our work has to be generalized to the block algorithm of Kaltofen and Villard (2004) (with the use of fast matrix multiplication algorithms) whose complexity estimate is currently the best known for computing the determinant, and the adjoint without divisions.

Acknowledgements. We thank Erich Kaltofen who has brought reference Ostrowski et al. (1971) to our attention.

Appendix: Hankel matrix inversion and anti-diagonal sums

For implementing (17), we study the computation of the anti-diagonal sums σ_k of H^{-1} and H_A^{-1}.

We first use the formula of Labahn et al. (1990) for Hankel matrices inversion. The minimum polynomial f of $\{h_k\}_{0\le k\le 2n-1}$ is $f(x) = f_0 + f_1 x + \ldots + f_{n-1}x^{n-1} + x^n$, and satisfies (11). Let the last column of H^{-1} be given by

$$H\,[g_0, g_1, \ldots, g_{n-1}]^T = [0, \ldots, 0, 1]^T \in \mathsf{K}^n. \tag{35}$$

Applying (Labahn et al., 1990, Theorem 3.1) with (11) and (35), we know that

$$H^{-1} = \begin{bmatrix} f_1 & \cdots & f_{n-1} & 1 \\ \vdots & \iddots & \iddots & \\ f_{n-1} & \iddots & 0 & \\ 1 & & & \end{bmatrix} \begin{bmatrix} g_0 & \cdots & g_{n-1} \\ & \ddots & \vdots \\ 0 & & g_0 \end{bmatrix} - \begin{bmatrix} g_1 & \cdots & g_{n-1} & 0 \\ \vdots & \iddots & \iddots & \\ g_{n-1} & \iddots & 0 & \\ 0 & & & \end{bmatrix} \begin{bmatrix} f_0 & \cdots & f_{n-1} \\ & \ddots & \vdots \\ 0 & & f_0 \end{bmatrix}. \tag{36}$$

For deriving an analogous formula for H_A^{-1}, using the notations of Section 4.1, we first recall that $H = \tilde{\mathcal{K}}_u\mathcal{K}_v$ and $H_A = \tilde{\mathcal{K}}_u A\mathcal{K}_v$. Multiplying (11) on the left by $\tilde{\mathcal{K}}_u A\tilde{\mathcal{K}}_u^{-1}$ gives

$$H_A\,[f_0, f_1, \ldots, f_{n-1}]^T = -[h_{n+1}, h_{n+2}, \ldots, h_{2n}]^T. \tag{37}$$

We also notice that

$$H_A H^{-1} = \left(\mathcal{K}_u^{-1} A^T \mathcal{K}_u\right)^T,$$

and, using the action of A^T on the vectors $u^T, \ldots, (A^T)^{n-2} u^T$, we check that $H_A H^{-1}$ is the companion matrix

$$H_A H^{-1} = \begin{bmatrix} 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & \ldots & 0 & 1 \\ -f_0 & -f_1 & \ldots & -f_{n-1} \end{bmatrix}.$$

Hence the last column $[g_0^*, g_1^*, \ldots, g_{n-1}^*]$ of H_A^{-1} is the first column of H^{-1} divided by $-f_0$. Using (36) for determining the first column of H^{-1}, we get

$$[g_0^*, g_1^*, \ldots, g_{n-1}^*]^T = -\frac{g_0}{f_0}[f_1, \ldots, f_{n-1}, 1]^T + [g_1, \ldots, g_{n-1}, 0]^T. \tag{38}$$

Applying (Labahn et al., 1990, Theorem 3.1), now with (37) and (38), we obtain

$$H_A^{-1} = \begin{bmatrix} f_1 & \ldots & f_{n-1} & 1 \\ \vdots & \iddots & \iddots & \\ f_{n-1} & \iddots & 0 & \\ 1 & & & \end{bmatrix} \begin{bmatrix} g_0^* & \cdots & g_{n-1}^* \\ & \ddots & \vdots \\ 0 & & g_0^* \end{bmatrix} - \begin{bmatrix} g_1^* & \cdots & g_{n-1}^* & 0 \\ \vdots & \iddots & \iddots & \\ g_{n-1}^* & \iddots & 0 & \\ 0 & & & \end{bmatrix} \begin{bmatrix} f_0 & \cdots & f_{n-1} \\ & \ddots & \vdots \\ 0 & & f_0 \end{bmatrix}. \tag{39}$$

From (36) and (39) we see that computing $\sigma_k(H^{-1})$ and $\sigma_{k-1}(H_A^{-1})$, for $0 \leq k \leq 2n-1$, reduces to computing the anti-diagonal sums for a product of triangular Hankel times triangular Toeplitz matrices. Let

$$M = LR = \begin{bmatrix} l_0 & l_1 & \ldots & l_{n-1} \\ l_1 & \iddots & \iddots & \\ \vdots & \iddots & 0 & \\ l_{n-1} & & & \end{bmatrix} \begin{bmatrix} r_0 & r_1 & \ldots & r_{n-1} \\ & \ddots & \ddots & r_{n-2} \\ & 0 & \ddots & \vdots \\ & & & r_0 \end{bmatrix}.$$

We have

$$m_{i,j} = \sum_{s=i-1}^{i+j-2} l_s r_{i+j-s-2}, \ 1 \leq i+j-1 \leq n, \tag{40}$$

and

$$m_{i,j} = \sum_{s=i-1}^{n-1} l_s r_{i+j-s-2}, \ n \leq i+j-1 \leq 2n-1. \tag{41}$$

For $0 \leq k \leq 2n-2$, $\sigma_k(M)$ is defined by summing the $m_{i,j}$'s such that $i+j-2=k$. Using (40) we obtain

$$\begin{aligned} \sigma_k(M) &= \textstyle\sum_{i=1}^{k+1} m_{i,k-i+2} = \sum_{i=1}^{k+1} \sum_{s=i-1}^{k} l_s r_{k-s}, \\ &= \textstyle\sum_{s=0}^{k} (s+1) l_s r_{k-s}, \ 0 \leq k \leq n-1, \end{aligned}$$

hence

$$(\sum_{s=0}^{n-1} l_s x^{s+1})'(\sum_{s=0}^{n-1} r_s x^s) \bmod x^n = \sum_{k=0}^{n-1} \sigma_k(M)\, x^k. \tag{42}$$

In the same way, using (41) with $\bar{k} = k - n + 2$, we have

$$\begin{aligned}\sigma_k(M) &= \textstyle\sum_{i=1}^{n-\bar{k}+1} m_{i+\bar{k}-1,n-i+1} = \sum_{i=1}^{n-\bar{k}+1} \sum_{s=i}^{n-\bar{k}+1} l_{s+\bar{k}-2} r_{n-s},\\ &= \textstyle\sum_{s=\bar{k}-1}^{n-1} (s+n-k)\, l_s r_{k-s},\ n-1 \le k \le 2n-2,\end{aligned}$$

and

$$(\sum_{s=1}^{n} r_{n-s} x^s)'(\sum_{s=0}^{n-1} l_{n-s-1} x^s) \bmod x^n = \sum_{k=0}^{n-1} \sigma_{2n-k-2}(M)\, x^k. \tag{43}$$

It remains to apply (42) and (43) to the structured matrix products in (36) and (39), for computing the $\sigma_k(H^{-1})$ and $\sigma_k(H_A^{-1})$'s. Together with the minimum polynomial $f = f_0 + \ldots + f_{n-1}x^{n-1} + x^n$, let $g = g_0 + \ldots + g_{n-1}x^{n-1}$ (see (35)), and $g^* = g_0^* \ldots + g_{n-1}^* x^{n-1}$ (see (38)). We may now combine, respectively (36) and (39), with (42), for obtaining

$$f'g - g'f \bmod x^n = \sum_{k=0}^{n-1} \sigma_k(H^{-1})\, x^k, \tag{44}$$

and

$$f'g^* - (g^*)'f \bmod x^n = \sum_{k=0}^{n-1} \sigma_k(H_A^{-1})\, x^k. \tag{45}$$

Defining also $\mathrm{rev}(f) = 1 + f_{n-1}x + \ldots + f_0 x^n$, $\mathrm{rev}(g) = g_{n-1}x + \ldots + g_0 x^n$, and $\mathrm{rev}(g^*) = g_{n-1}^* x + \ldots + g_0^* x^n$, the combination of, respectively, (36) and (39), with (43), leads to

$$\mathrm{rev}(g)'\mathrm{rev}(f) - \mathrm{rev}(f)'\mathrm{rev}(g) \bmod x^n = \sum_{k=0}^{n-1} \sigma_{2n-k-2}(H)\, x^k, \tag{46}$$

and

$$\mathrm{rev}(g^*)'\mathrm{rev}(f) - \mathrm{rev}(f)'\mathrm{rev}(g^*) \bmod x^n = \sum_{k=0}^{n-1} \sigma_{2n-k-2}(H_A)\, x^k. \tag{47}$$

Proposition 4. *Assume that the minimum polynomial f and the Hankel matrices H and H_A are given. The anti-diagonal sums $\sigma_k(H^{-1})$ and $\sigma_k(H_A^{-1})$, for $0 \le k \le 2n-1$, can be computed in $\mathsf{G}(n) + O(\mathsf{M}(n))$ operations in K.*

Using the approach of Brent et al. (1980) we know that computing the last column of H^{-1} reduces to an extended Euclidean problem of degree $2n$. Hence the polynomial g is computed in $\mathsf{G}(n) + O(n)$ operations. From there, g^* is computed using (38). Then, applying (44)-(47) leads to the cost $O(\mathsf{M}(n))$.

References

Baur, W., Strassen, V., 1983. The complexity of partial derivatives. Theor. Comp. Sc. 22, 317–330.

Brent, R., Gustavson, F., Yun, D., 1980. Fast solution of Toeplitz systems of equations and computation of Padé approximations. Journal of Algorithms 1, 259–295.

Bürgisser, P., Clausen, M., Shokrollahi, M., 1997. Algebraic Complexity Theory. Volume 315, Grundlehren der mathematischen Wissenschaften. Springer-Verlag.

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

Chen, L., Eberly, W., Kaltofen, E., Saunders, B., Turner, W., Villard, G., 2002. Efficient matrix preconditioners for black box linear algebra. Linear Algebra and its Applications 343-344, 119–146.

Coppersmith, D., Winograd, S., 1990. Matrix multiplication via arithmetic progressions. J. of Symbolic Computation 9 (3), 251–280.

Eberly, W., Jul 1997. Processor-efficient parallel matrix inversion over abstract fields: two extensions. In: Proc. Second International Symposium on Parallel Symbolic Computation, Maui, Hawaii, USA. ACM Press, pp. 38–45.

Eberly, W., Giesbrecht, M., Villard, G., Nov. 2000. Computing the determinant and Smith form of an integer matrix. In: The 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA. IEEE Computer Society Press, pp. 675–685.

von zur Gathen, J., Gerhard, J., 1999. Modern Computer Algebra. Cambridge University Press.

Jeannerod, C., Villard, G., 2006. Asymptotically fast polynomial matrix algorithms for multivariable systems. Int. J. Control 79 (11), 1359–1367.

Kaltofen, E., Jul. 1992. On computing determinants without divisions. In: International Symposium on Symbolic and Algebraic Computation, Berkeley, California USA. ACM Press, pp. 342–349.

Kaltofen, E., 2000. Challenges of symbolic computation: my favorite open problems. J. of Symbolic Computation 29 (6), 891–919.

Kaltofen, E., Pan, V., 1991. Processor efficient parallel solution of linear systems over an abstract field. In: Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architecture. ACM-Press, pp. 180–191.

Kaltofen, E., Saunders, B., 1991. On Wiedemann's method of solving sparse linear systems. In: Proc. AAECC-9. LNCS 539, Springer Verlag. pp. 29–38.

Kaltofen, E., Villard, G., 2004. On the complexity of computing determinants. Computational Complexity 13, 91–130.

Knuth, D., 1970. The analysis of algorithms. In: Proc. International Congress of Mathematicians, Nice, France. Vol. 3. pp. 269–274.

Labahn, G., Choi, D., Cabay, S., 1990. The inverses of block Hankel and block Toeplitz matrices. SIAM J. Comput. 19 (1), 98–123.

Linnainmaa, S., 1970. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (in Finnish). Master's thesis, University of Helsinki, Dpt of Computer Science.

Linnainmaa, S., 1976. Taylor expansion of the accumulated rounding errors. BIT 16, 146–160.

Moenck, R., 1973. Fast computation of Gcds. In: 5 th. ACM Symp. Theory Comp. pp. 142–151.

Morgenstern, J., 1985. How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen. ACM SIGACT News 16, 60–62.

Ostrowski, G. M., Wolin, J. M., Borisow, W. W., 1971. Über die Berechnung von Ableitungen (in German). Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg 13 (4), 382–384.

Schönhage, A., 1971. Schnelle Berechnung von Kettenbruchenwicklungen. Acta Informatica 1, 139–144.

Storjohann, A., 2003. High-order lifting and integrality certification. Journal of Symbolic Computation 36 (3-4), 613–648, special issue International Symposium on Symbolic and Algebraic Computation (ISSAC'2002). Guest editors: M. Giusti & L. M. Pardo.

Storjohann, A., 2005. The shifted number system for fast linear algebra on integer matrices. Journal of Complexity 21 (4), 609–650.

Storjohann, A., Jul. 2008. On the complexity of inverting integer and polynomial matrices. Preprint D.R. Cheriton School of Computer Science, U. Waterloo, Ontario, Canada.

Strassen, V., 1973. Vermeidung von Divisionen. J. Reine Angew. Math. 264, 182–202.

Wiedemann, D., 1986. Solving sparse linear equations over finite fields. IEEE Transf. Inform. Theory IT-32, 54–62.