New fast euclidean algorithms

Marie-Françoise Roy*
Sidi Mohamed Sedjelmaci †

May 18, 2012

Abstract

We give new simple algorithms for the fast computation of the quotient boot and the gcd of two polynomials, and obtain a complexity $O(d(\log_2 d)^2)$, where d is the degree of the polynomials, similarly to [5, 4]. More precisely, denoting by $M(d)$ the cost of a fast multiplication of polynomials of degree d, we reach the complexity $(9/2\, M(d) + O(d)) \log_2 d$ where d is the degree of the polynomials in the non-defective case (when degrees drop one by one), and $(21\, M(d) + O(d)) \log_2 d + O(M(d))$ in the general case, improving the complexity bounds (respectively $(10\, M(d) + O(d)) \log_2 d$ and $(24\, M(d) + O(d)) \log_2 d + O(M(d))$) previsouly known for these problems [2] (see Exercise 11.7).

We hope that the simple character of our algorithms will make it easier to use fast algorithms in practice for these problems.

1 Introduction

While the number of coefficients arising in the euclidean remainder sequence of two polynomials of degre d is of order $O(d^2)$, the quotient boot, consisting of the sequence of quotients and the gcd, contains only $O(d)$ coefficients. Note that the quotient boot provides the gcd as well as enough information to compute in linear time the Cauchy index of a rational fraction with numerator and denominator bounded by d, which is a basic computational task in real algebraic geometry, [3, 1]. Our aim is to design an algorithm performing the quotien boot in quasi-linear time.

Let us introduce some notation. Let $\mathrm{P}(0)$ be a polynomial of degree d_0 and $\mathrm{P}(1)$ a polynomial of degree $d_1 < d_0$. Define the remainder sequence of $\mathrm{P}(0), \mathrm{P}(1)$ by

$$\mathrm{P}(i+1) := \mathrm{Rem}(\mathrm{P}(i-1), \mathrm{P}(i)), \tag{1}$$

*IRMAR (URA CNRS 305), Université de Rennes, Campus de Beaulieu 35042 Rennes cedex FRANCE, marie-francoise.roy@univ-rennes1.fr.

†LIPN (UMR 7030, CNRS), Université de Paris Nord, Av. J.-B. Clément, 93430, Villetaneuse France, sms@lipn.univ-paris13.fr.

stopping at the first n such that $\mathrm{P}(n+1) = 0$. Thus, $G = \gcd(\mathrm{P}(0), \mathrm{P}(1)) = \mathrm{P}(n)$, and the remainder sequence is

$$\mathrm{P}(0), \mathrm{P}(1), \ldots, \mathrm{P}(n).$$

We denote $d_i = \deg(\mathrm{P}(i))$ (by convention $\deg(0) = -\infty$).

Let

$$\mathrm{C}(i) := \mathrm{Quo}(\mathrm{P}(i-1), \mathrm{P}(i)) \tag{2}$$

be the quotient of $\mathrm{P}(i-1)$ by $\mathrm{P}(i)$.

Denoting

$$\mathrm{M}(i) := \begin{bmatrix} 0 & 1 \\ 1 & -\mathrm{C}(i) \end{bmatrix}, \tag{3}$$

$$\mathrm{V}(i) := \begin{bmatrix} \mathrm{P}(i) \\ \mathrm{P}(i+1) \end{bmatrix}, \tag{4}$$

we have

$$\mathrm{V}(i) = \mathrm{M}(i) \cdot \mathrm{V}(i-1). \tag{5}$$

Let also

$$\mathrm{M}(i,j) := \prod_{k=i-2^j+1}^{i} \mathrm{M}(k), \tag{6}$$

(note that $\mathrm{M}(i,0) := \mathrm{M}(i)$).

Our aim is to compute the **quotient boot** of $\mathrm{P}(0)$ and $\mathrm{P}(1)$, i.e. the quotients $\mathrm{C}(i)$, for $i = 1, \ldots, n$, as well as the gcd $G = \mathrm{P}(n)$ in time $O(d(\log_2 d)^2)$.

This paper contains three different algorithms for computing fastly the quotient boot.

They are all based on two classical properties that are usually adopted to design fast gcd algorithms (see [5, 4]):

- the leading part of two polynomials is enough to compute their quotient,
- some products of the form $\mathrm{M}(i,j)$, computed using fast multiplication, can be used for intermediate computations.

The main difference between our approach and the existing litterature, based on a Divide and Conquer approach, is that our algorithms proceed straightforwardly, basing the computation on the dyadic valuation of the iteration index.

In Section 2, we give two different algorithms computing the quotient boot when the degrees of the polynomials in the remainder sequence is known in advance: the first A_1 is particularly simple, and the second A_2 avoids some redundant computations. In Section 3 we describe our maint result, Algorithm B giving the quotient boot when the degrees of the remainders are unknown.

The complexity analysis of Algorithm B is given in details in Section 4 both in the non defective case (when the degrees of the quotients are equal to 1) and in the general case. We obtain the complexity $(9/2\, M(d) + O(d)) \log_2 d$ in the non-defective case (when degrees drop one by one), and $(21\, M(d) + O(d)) \log_2 d + O(M(d))$ in the general case, improving the complexity bounds (respectively $(10\, M(d) + O(d)) \log_2 d$ and $(24\, M(d) + O(d)) \log_2 d + O(M(d))$) previsouly known for these problems [2] (see Exercise 11.7).

2 Computing the quotient boots when the degrees of the remainders are known.

We suppose in this Section that we know in advance the degree d_i of the polynomials $\mathrm{P}(i)$ in the remainder sequence.

We first introduce notation that will be useful throughout the paper.

Given a polynomial

$$P = \sum_{\ell=0}^{\deg(P)} p_\ell X^\ell, \tag{7}$$

we denote, if $m \geq n$,

$$P_{]m\cdots n]} := \sum_{\ell=n}^{m-1} p_\ell X^\ell. \tag{8}$$

(with the convention that a monomial above the degree, or with a negative exponent has coefficient 0),

$$P_{]m\cdots]} := \sum_{\ell=0}^{m-1} p_\ell X^\ell. \tag{9}$$

and

$$P_{]\cdots n]} := \sum_{\ell=n}^{\deg(P)} p_\ell X^\ell. \tag{10}$$

If

$$V = \begin{bmatrix} P \\ Q \end{bmatrix} \tag{11}$$

is a vector of two polynomials, we denote by

$$V_{]m\cdots n]} := \begin{bmatrix} P_{]m\cdots n]} \\ Q_{]m\cdots n]} \end{bmatrix}, \tag{12}$$

$$V_{]m\cdots]} := \begin{bmatrix} P_{]m\cdots]} \\ Q_{]m\cdots]} \end{bmatrix}, \tag{13}$$

and

$$V_{]\cdots n]} := \begin{bmatrix} P_{]\cdots n]} \\ Q_{]\cdots n]} \end{bmatrix}. \tag{14}$$

The following obvious lemma will be very useful

Lemma 2.1 *Let V be a vector of two polynomials and M a 2×2 matrix of polynomials of degree at most d*

$$(M \cdot V)_{]\cdots n+d]} = (M \cdot V_{]\cdots n]})_{]\cdots n+d]}\,.$$

In other words, $M \cdot V$ and $M \cdot V_{]\cdots n]}$ coincide up to degree $n + d$.

2.1 Algorithm A_1 (Quotient boot for known degrees)

We remark that the matrix $\mathrm{M}(i)$ depends only on the $d_{i-1}-d_i+1$ leading terms of $\mathrm{P}(i-1)$ and $\mathrm{P}(i)$, since the degree of $\mathrm{C}(i)$ is $d_{i-1}-d_i$. As a consequence,

$$\mathrm{C}(i) = \mathrm{Quo}(\mathrm{P}(i-1)_{]\cdots d_i]}, \mathrm{P}(i)_{]\cdots 2d_i-d_{i-1}]}) \tag{15}$$

We are going to define, by induction on i, a vector of two polynomials

$$\overline{\mathrm{V}}(i) = \begin{bmatrix} \overline{\mathrm{P}}(i) \\ \overline{\mathrm{Q}}(i) \end{bmatrix}, \tag{16}$$

which should be thought as an approximation of $\mathrm{V}(i)$: enough of the leading terms of $\overline{\mathrm{V}}(i)$ coincide with the leading terms of $\mathrm{V}(i)$ to allow the correct computation of $\mathrm{C}(i)$ as

$$\mathrm{Quo}(\overline{\mathrm{P}}(i-1)_{]\cdots d_i]}, \overline{\mathrm{Q}}(i-1)_{]\cdots 2d_i-d_{i-1}]}). \tag{17}$$

The first steps of our method. In order to compute $\mathrm{C}(1)$ (thus $\mathrm{M}(1,0)$) correctly, we divide $\mathrm{P}(0)_{]\cdots d_1]}$ by $\mathrm{P}(1)_{]\cdots 2d_1-d_0]}$ and notice that with less leading terms we would not get the right answer. Now rather than multiplying $\mathrm{V}(0)$ by $\mathrm{M}(1,0)$ to get $\mathrm{V}(1)$, we multiply enough leading terms of $\mathrm{V}(0)$ by $\mathrm{M}(1,0)$ in order to compute the correct $\mathrm{C}(2)$. Using our knowledge of d_2, we define

$$\overline{\mathrm{V}}(1) := \mathrm{M}(1,0) \cdot \mathrm{V}(0)_{]\cdots 2d_2-d_0]},$$

and notice that, since $\mathrm{C}(1)$ is of degree d_0-d_1, $\overline{\mathrm{V}}(1)$ coincides with $\mathrm{V}(1)$ up to degree $2d_2-d_1$, so that the quotient $\mathrm{Quo}(\overline{\mathrm{P}}(1)_{]\cdots d_2]}, \overline{\mathrm{Q}}(1)_{]\cdots 2d_2-d_1]})$ coincides with $\mathrm{C}(2)$. In other words $\mathrm{M}(1,0)$ and $\mathrm{M}(2,0)$ have been computed correctly from $\mathrm{V}(0)_{]\cdots 2d_2-d_0]}$.

So, in order to compute the correct $\mathrm{M}(3,0)$ and $\mathrm{M}(4,0)$, we need to compute correctly $\mathrm{V}(2)_{]\cdots 2d_4-d_2]}$, using our knowledge of d_4. This is done by defining:

$$\begin{aligned} \mathrm{M}(2,1) &:= \mathrm{M}(2,0) \cdot \mathrm{M}(1,0) \\ \overline{\mathrm{V}}(2) &:= \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]\cdots 2d_4-d_0]}. \end{aligned}$$

Since the entries of $\mathrm{M}(2,1)$ are polynomials of degree at most d_0-d_2, it is easy to check that $\overline{\mathrm{V}}(2)$ coincides with $\mathrm{V}(2)$ up to degree $2d_4-d_2$. So $\overline{\mathrm{V}}(2)$ can be used to compute successfully $\mathrm{M}(3,0)$, $\overline{\mathrm{V}}(3)$ and $\mathrm{M}(4,0)$. We define

$$\overline{\mathrm{V}}(3) := \mathrm{M}(3,0) \cdot \overline{\mathrm{V}}(2)_{]\cdots 2d_4-d_2]}.$$

Finally $\mathrm{M}(1,0)$, $\mathrm{M}(2,0)$, $\mathrm{M}(3,0)$ and $\mathrm{M}(4,0)$ have been computed correctly from $\mathrm{V}(0)_{]\cdots 2d_4-d_0]}$.

So in order to compute correctly $\mathrm{M}(5,0)$, $\mathrm{M}(6,0)$, $\mathrm{M}(7,0)$ and $\mathrm{M}(8,0)$, we need to compute correctly $\mathrm{V}(4)_{]\cdots 2d_8-d_4]}$, using our knowledge of d_8. We define

$$\begin{aligned} \mathrm{M}(4,1) &:= \mathrm{M}(4,0) \cdot \mathrm{M}(3,0) \\ \mathrm{M}(4,2) &:= \mathrm{M}(4,1) \cdot \mathrm{M}(2,1) \\ \overline{\mathrm{V}}(4) &:= \mathrm{M}(4,2) \cdot \mathrm{V}(0)_{]\cdots 2d_8-d_0]}. \end{aligned}$$

Since the entries of $\mathrm{M}(4,2)$ are polynomials of degree at most $d_0 - d_4$, $\overline{\mathrm{V}}(4)$ coincides with $\mathrm{V}(4)$ up to degree $2d_8 - d_4$.

So we can proceed safely to compute $\mathrm{M}(5,0)$, $\mathrm{M}(6,0)$, $\mathrm{M}(7,0)$ and $\mathrm{M}(8,0)$.

We observe that the number of coefficients up to which $\overline{\mathrm{V}}(i)$ has to coincide with $\mathrm{V}(i)$ depends on the dyadic valuation of i, i.e. on the maximum power of 2 dividing i.

Notation 2.2 *For every natural number $i \geq 0$, we denote by $\mathrm{v}(i)$ the dyadic valuation of i, i.e. the natural number such that $i = 2^{\mathrm{v}(i)}j$ with j odd, with the convention $\mathrm{v}(0) = +\infty$.*

We note $\mathrm{p}(i) = i - 2^{\mathrm{v}(i)}$, $\mathrm{f}(i) = i + 2^{\mathrm{v}(i)}$ ($\mathrm{p}(i)$ stands for past and $\mathrm{f}(i)$ for future), with the convention $\mathrm{f}(0) = +\infty, \mathrm{p}(0) = -\infty, d_\infty = -\infty$.

We define inductively $\mathrm{p}^{(\ell)}(i)$ by $\mathrm{p}^{(0)}(i) = i, \mathrm{p}^{(\ell)}(i) = \mathrm{p}(\mathrm{p}^{(\ell-1)}(i))$, until we reach 0.

For example if $i = 22, \mathrm{v}(i) = 1, \mathrm{f}(i) = 24, \mathrm{p}(i) = 20, \mathrm{p}^{(2)}(i) = 16, \mathrm{p}^{(3)}(i) = 0$.

The following lemma is immediate.

Lemma 2.3 *For ℓ from $\mathrm{v}(i) - 1$ to 0,*

$$\mathrm{p}(i - 2^\ell) = i - 2^{\ell+1},\ \mathrm{f}(i - 2^\ell) = i.$$

In particular

$$\mathrm{p}(i - 2^{\mathrm{v}(i)-1}) = \mathrm{p}(i),\ \mathrm{f}(i - 2^{\mathrm{v}(i)-1}) = i.$$

Description and correctness of Algorithm A_1. We are now ready for the description of Algorithm A_1.

We define, by induction on i,

$$\overline{\mathrm{V}}(i) := \begin{bmatrix} \overline{\mathrm{P}}(i) \\ \overline{\mathrm{Q}}(i) \end{bmatrix}, \tag{18}$$

as follows

$\overline{\mathrm{V}}(0) = \mathrm{V}(0)$,

$$\overline{\mathrm{C}}(i) := \mathrm{Quo}(\overline{\mathrm{P}}(i-1)_{]\cdots d_i]}, \overline{\mathrm{Q}}(i-1)_{]\cdots 2d_i - d_{i-1}]}), \tag{19}$$

$$\overline{\mathrm{M}}(i) := \begin{bmatrix} 0 & 1 \\ 1 & -\overline{\mathrm{C}}(i) \end{bmatrix}, \tag{20}$$

$$\overline{\mathrm{M}}(i,j) := \prod_{k=i-2^j+1}^{i} \overline{\mathrm{M}}(k), \tag{21}$$

(note that $\overline{\mathrm{M}}(i,0) := \overline{\mathrm{M}}(i)$) and

$$\overline{\mathrm{V}}(i) = \overline{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \mathrm{V}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}]}. \tag{22}$$

We now prove the following result, which is the key to the correctness of Algorithm A_1.

Proposition 2.4 *For every i,*

$$\overline{\mathrm{C}}(i) = \mathrm{C}(i).$$

Note that $\mathrm{f}(i-1) \geq i$, thus $d_{\mathrm{f}(i-1)} \leq d_i$, so that $2d_i - d_{i-1} \geq 2d_{\mathrm{f}(i-1)} - d_{i-1}$. So, Proposition 2.4 is an immediate corollary of the following Lemma, taking into account Equation (15).

Lemma 2.5 *For every i,*

$$\overline{\mathrm{V}}(i)_{]\cdots 2d_{\mathrm{f}(i)}-d_i]} = \mathrm{V}(i)_{]\cdots 2d_{\mathrm{f}(i)}-d_i]}.$$

In other words, $\overline{\mathrm{V}}(i)$ and $\mathrm{V}(i)$ coincide up to degree $2d_{\mathrm{f}(i)} - d_i$.

Proof : By induction on i. The claim is obviously true for $i = 0$. We suppose by induction hypothesis that the claim is true for every $j < i$. In particular for $j = \mathrm{p}(i) < i$,

$$\overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(\mathrm{p}(i))}-d_{\mathrm{p}(i)}]} = \mathrm{V}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(\mathrm{p}(i))}-d_{\mathrm{p}(i)}]}.$$

By definition of $\overline{\mathrm{V}}(i)$, since the degree of the entries of $\mathrm{M}(i, \mathrm{v}(i))$ are at most $d_{\mathrm{p}(i)} - d_i$, and using Lemma 2.1,

$$\overline{\mathrm{V}}(i)_{]\cdots 2d_{\mathrm{f}(\mathrm{p}(i))}-d_i]} = \mathrm{V}(i)_{]\cdots 2d_{\mathrm{f}(\mathrm{p}(i))}-d_i]}.$$

It remains to notice that, since $\mathrm{v}(\mathrm{p}(i)) > \mathrm{v}(i)$,

$$\mathrm{f}(\mathrm{p}(i)) = \mathrm{p}(i) + 2^{\mathrm{v}(\mathrm{p}(i))} = i - 2^{\mathrm{v}(i)} + 2^{\mathrm{v}(\mathrm{p}(i))} \geq i + 2^{\mathrm{v}(i)} = \mathrm{f}(i),$$

and $d_{\mathrm{f}(\mathrm{p}(i))} \leq d_{\mathrm{f}(i)}$. *QED*

The input of Algorithm A_1 is $\mathrm{P}(0), \mathrm{P}(1)$ and the degrees d_i of $\mathrm{P}(i)$. Its output is the quotient boot.

Algorithm 2.6 (A_1, Quotient boot for known degrees)

- *Initialize* $\overline{\mathrm{V}}(0) := \mathrm{V}(0)$, $i := 1$.
- **While** $\overline{\mathrm{Q}}(i-1) \neq 0$, *with*

$$\overline{\mathrm{V}}(i-1) := \begin{bmatrix} \overline{\mathrm{P}}(i-1) \\ \overline{\mathrm{Q}}(i-1) \end{bmatrix},$$

  - *Define*

$$\begin{aligned} \mathrm{C}(i) &:= \mathrm{Quo}(\overline{\mathrm{P}}(i-1)_{]\cdot d_i]}, \overline{\mathrm{Q}}(i-1)_{]\cdots 2d_i - d_{i-1}]}), \\ \mathrm{M}(i,0) &:= \begin{bmatrix} 0 & 1 \\ 1 & -\mathrm{C}(i) \end{bmatrix}. \end{aligned}$$

– *If i is even, compute for $\ell = 0 \cdots \mathrm{v}(i) - 1$,*

$$\mathrm{M}(i, \ell + 1) \quad := \quad \mathrm{M}(i, \ell) \cdot \mathrm{M}(i - 2^\ell, \ell).$$

– *Compute*

$$\overline{\mathrm{V}}(i) \quad := \quad \mathrm{M}(i, \mathrm{v}(i)) \cdot \overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}]}.$$

– *Replace i by $i + 1$.*

- **EndWhile**
- *Define* $G := \overline{\mathrm{P}}(i - 1)$.

The correctness of Algorithm A_1 follows from Proposition 2.4.

2.2 Algorithm A_2: Improved quotien boot for known degrees.

It is possible to improve the preceding algorithm by taking into account the partial computation already performed in the computation of $\overline{\mathrm{V}}(i)$.

The first steps of our method. We explain how Algorihm A_2 works for the first values of i.

The first change in Algorithm A_2 with respect to Algorithm A_1 appears when we compute $\overline{\mathrm{V}}(2)$. Rather than computing $\overline{\mathrm{V}}(2)$ as $\mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]\cdots 2d_4 - d_0]}$, we compute it as

$$\mathrm{M}(2,0) \cdot \overline{\mathrm{V}}(1) + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]2d_2 - d_0 \cdots 2d_4 - d_0]}.$$

Indeed

$$\begin{aligned}
\overline{\mathrm{V}}(2) &= \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]\cdots 2d_4 - d_0]} \\
&= \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]\cdots 2d_2 - d_0]} + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]2d_2 - d_0 \cdots 2d_4 - d_0]} \\
&= \mathrm{M}(2,0) \cdot \mathrm{M}(1) \cdot \mathrm{V}(0)_{]\cdots 2d_2 - d_0]} + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]2d_2 - d_0 \cdots 2d_4 - d_0]} \\
&= \mathrm{M}(2,0) \cdot \overline{\mathrm{V}}(1) + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]2d_2 - d_0 \cdots 2d_4 - d_0]}.
\end{aligned}$$

The advantage of this new method for computing $\overline{\mathrm{V}}(2)$ is that we use more extensively the previously computed data, namely $\overline{\mathrm{V}}(1)$, rather than starting directly from $\mathrm{V}(0)$, and this saves some computation.

The next change with respect to Algorithm A_1 is the computation of $\overline{\mathrm{V}}(4)$. Rather than computing $\overline{\mathrm{V}}(4)$ as $\mathrm{M}(4,2) \cdot \mathrm{V}(0)_{]\cdots 2d_8 - d_0]}$, we compute it as

$$\mathrm{M}(4,0) \cdot \overline{\mathrm{V}}(3) + \mathrm{M}(4,1) \cdot \overline{\mathrm{V}}(2)_{]2d_4 - d_2 \cdots]} + \mathrm{M}(4,2) \cdot \mathrm{V}(0)_{]2d_4 - d_0 \cdots 2d_8 - d_0]}.$$

It is easy to check that

$$\begin{aligned}
\overline{\mathrm{V}}(4) &= \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]\cdots 2d_4-d_0]} + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,1)\cdot \mathrm{M}(2,1)\cdot \mathrm{V}(0)_{]\cdots 2d_4-d_0]} + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,1)\cdot \overline{\mathrm{V}}(2) + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,1)\cdot \overline{\mathrm{V}}(2)_{]\cdots 2d_4-d_2]} + \mathrm{M}(4,1)\cdot \overline{\mathrm{V}}(2)_{]2d_4-d_2\cdots]} \\
&\quad + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,0)\cdot \mathrm{M}(3)\cdot \overline{\mathrm{V}}(2)_{]\cdots 2d_4-d_2]} + \mathrm{M}(4,1)\cdot \overline{\mathrm{V}}(2)_{]2d_4-d_2\cdots]} \\
&\quad + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]} \\
&= \mathrm{M}(4,0)\cdot \overline{\mathrm{V}}(3) + \mathrm{M}(4,1)\cdot \overline{\mathrm{V}}(2)_{]2d_4-d_2\cdots]} + \mathrm{M}(4,2)\cdot \mathrm{V}(0)_{]2d_4-d_0\cdots 2d_8-d_0]}.
\end{aligned}$$

The advantage of this new method for computing $\overline{\mathrm{V}}(4)$ is that we use more extensively the previously computed data, namely $\overline{\mathrm{V}}(2)$ and $\overline{\mathrm{V}}(3)$, rather than starting directly from $\mathrm{V}(0)$, and this saves some computation.

Description and correctness of Algorithm "Improved quotien boot for known degrees"

Algorithm 2.7 (A_2: Improved quotient boot for known degrees)

- *Initialize* $\overline{\mathrm{V}}(0) := \mathrm{V}(0)$, $i := 1$.
- **While** $\overline{\mathrm{Q}}(i-1) \neq 0$, *with*

$$\overline{\mathrm{V}}(i-1) := \begin{bmatrix} \overline{\mathrm{P}}(i-1) \\ \overline{\mathrm{Q}}(i-1) \end{bmatrix},$$

  - *Define*

$$\begin{aligned}
\mathrm{C}(i) &:= \mathrm{Quo}(\overline{\mathrm{P}}(i-1)_{]\cdots d_i]}, \overline{\mathrm{Q}}(i-1)_{]\cdots 2d_i-d_{i-1}]}), \\
\mathrm{M}(i,0) &:= \begin{bmatrix} 0 & 1 \\ 1 & -\mathrm{C}(i) \end{bmatrix}.
\end{aligned}$$

  - *If i is even, compute for $k = 0 \cdots \mathrm{v}(i) - 1$,*

$$\mathrm{M}(i,k+1) := \mathrm{M}(i,k)\cdot \mathrm{M}(i-2^k,k),$$

  - *If i is odd, compute*

$$\overline{\mathrm{V}}(i) := \mathrm{M}(i,0)\cdot \overline{\mathrm{V}}(i-1)_{]\cdots 2d_{i+1}-d_{i-1}]}.$$

  - *If i is even, compute*

$$\begin{aligned}
\overline{\mathrm{V}}(i) &= \mathrm{M}(i,0)\cdot \overline{\mathrm{V}}(i-1) \\
&\quad + \textstyle\sum_{k=1}^{\mathrm{v}(i)-1} \mathrm{M}(i,k)\cdot \overline{\mathrm{V}}(i-2^k)_{]2d_i-d_{i-2^k}\cdots]} \\
&\quad + \mathrm{M}(i,\mathrm{v}(i))\cdot \overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}.
\end{aligned}$$

– *Replace* i *by* $i+1$.

- **EndWhile**
- *Define* $G := \overline{\mathrm{P}}(i-1)$.

We now prove the correctness of Algorithm A_2 , which is a consequence of the following result and of the correctness of Algorithm A_1.

Lemma 2.8 *When* i *is even*

$$\begin{array}{rcl}\overline{\mathrm{V}}(i) & = & \mathrm{M}(i,0)\cdot\overline{\mathrm{V}}(i-1)\\ & & +\sum_{k=1}^{\mathrm{v}(i)-1}\mathrm{M}(i,k)\cdot\overline{\mathrm{V}}(i-2^k)_{]2d_i-d_{i-2^k}\cdots]}\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}.\end{array}$$

Proof : We prove by induction on ℓ from $\mathrm{v}(i)-1$ to 0 that

$$\begin{array}{rcl}\overline{\mathrm{V}}(i) & = & \mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^\ell)\\ & & +\sum_{k=\ell+1}^{\mathrm{v}(i)-1}\mathrm{M}(i,k)\cdot\overline{\mathrm{V}}(i-2^k)_{]2d_i-d_{i-2^k}\cdots]}\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}.\end{array}$$

Using Lemma 2.3,

$$\mathrm{p}(i-2^\ell)=i-2^{\ell+1},\ \mathrm{f}(i-2^\ell)=i.$$

In particular

$$\mathrm{p}(i-2^{\mathrm{v}(i)-1})=\mathrm{p}(i),\ \mathrm{f}(i-2^{\mathrm{v}(i)-1})=i.$$

If $\ell=\mathrm{v}(i)-1$,

$$\begin{array}{rcl}\overline{\mathrm{V}}(i) & = & \mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}\\ & = & \mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_i-d_{\mathrm{p}(i)}]}\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}\\ & = & \mathrm{M}(i,\mathrm{v}(i)-1)\cdot\mathrm{M}(i-2^{\mathrm{v}(i)-1},\mathrm{v}(i)-1)\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_i-d_{\mathrm{p}(i)}]}\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}\\ & = & \mathrm{M}(i,\mathrm{v}(i)-1)\cdot\overline{\mathrm{V}}(i-2^{\mathrm{v}(i)-1})\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}\end{array}$$

Suppose now by induction hypothesis that

$$\begin{array}{rcl}\overline{\mathrm{V}}(i) & = & \mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^\ell)\\ & & +\sum_{k=\ell+1}^{\mathrm{v}(i)-1}\mathrm{M}(i,k)\cdot\overline{\mathrm{V}}(i-2^k)_{]2d_i-d_{i-2^k}\cdots]}\\ & & +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}.\end{array}\quad .$$

The claim for $\ell - 1$ follows from

$$\begin{aligned}
\mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^{\ell}) &= \mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^{\ell})_{]\cdots 2d_i-d_{i-2^{\ell}}]} \\
&\quad +\mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^{\ell})_{]2d_i-d_{i-2^{\ell}}\cdots]} \\
&= \mathrm{M}(i,\ell-1)\cdot\mathrm{M}(i-2^{\ell-1},\ell-1)\cdot\overline{\mathrm{V}}(i-2^{\ell})_{]\cdots 2d_i-d_{i-2^{\ell}}]} \\
&\quad +\mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^{\ell})_{]2d_i-d_{i-2^{\ell}}\cdots]} \\
&= \mathrm{M}(i,\ell-1)\cdot\overline{\mathrm{V}}(i-2^{\ell-1}) + \mathrm{M}(i,\ell)\cdot\overline{\mathrm{V}}(i-2^{\ell})_{]2d_i-d_{i-2^{\ell}}\cdots]}.
\end{aligned}$$

QED

Remark 2.9 *Algorithm A_1 and Algorithm A_2 are half-gcd algorithm in the following sense. Let ν the number of bits of d, and $2^{\nu-1}$ the biggest power of 2 stricty smaller than d. At step $2^{\nu-1}$, Algorithm A_1 (resp. A_2) compute*

$$\overline{\mathrm{V}}(2^{\nu-1}) = \mathrm{V}(2^{\nu-1}) = \mathrm{M}(2^{\nu-1},\nu-1)\cdot\mathrm{V}(0)$$

and then calls itself with input $\overline{\mathrm{V}}(2^{\nu-1}) = \mathrm{V}(2^{\nu-1})$, without needing to record any of the preceeding computations.

3 An algorithm computing the quotient boot

The main difficulty we have to face now is that contrarily to what we supposed in the last section, we do not know the degree d_i of the successive remainders.

We are going to define for every i a vector of two polynomials $\widetilde{\mathrm{V}}(i) = \begin{bmatrix}\widetilde{\mathrm{P}}(i)\\ \widetilde{\mathrm{Q}}(i)\end{bmatrix}$, with the property that enough of the leading terms of $\widetilde{\mathrm{V}}(i)$ coincide with the leading terms of $\mathrm{V}(i)$ to allow the correct computation of $\mathrm{C}(i)$ as

$$\mathrm{Quo}(\widetilde{\mathrm{P}}(i-1)_{]\cdots d_i]}, \widetilde{\mathrm{Q}}(i-1)_{]\cdots 2d_i-d_{i-1}]}). \tag{23}$$

As before, the main idea is that since the leading part of $\mathrm{V}(i-1)$ only is necessary to compute $\mathrm{C}(i)$, we an postpone the multiplication of the part of lower degree of $\mathrm{V}(i-1)$ by $\mathrm{M}(i)$, which makes it possible to group the matrices $\mathrm{M}(i)$ before performing these postponed computations.

Since we do not know the d_i, we are going to compute non definitive $\widetilde{\mathrm{V}}(i)$ that will be updated during the computation. The idea is to start the computation as if the new degrees were decreasing as the old ones and to update the necessary $\widetilde{\mathrm{V}}(i)$ if it is not the case, i.e. if the drop of degree is bigger than expected.

The first steps of our method. We explain how the algorithm works for the first values of i.

$\star$ For $i = 1$, reading $\mathrm{V}(0)$ gives the correct value of d_1, and we can compute $\mathrm{C}(1)$ but in order to compute $\overline{\mathrm{V}}(1)$ the value of d_2 is necessary. Since we do

not know yet d_2 we start by acting as if $d_1 - d_2 \leq d_0 - d_1$. We initialize δ_1 to $2(d_0 - d_1)$ and define $\bar{\delta}_1 := \delta_1 + 2(d_0 - d_1)$. We define

$$\widetilde{\mathrm{V}}(1) := \mathrm{M}(1) \cdot \mathrm{V}(0)_{]\cdots d_0 - \bar{\delta}_1]}.$$

We note that $\widetilde{\mathrm{V}}(1)_{]\cdots d_1 - \delta_1]} = \mathrm{V}(1)_{]\cdots d_1 - \delta_1]}$, using Lemma 2.1 and check whether $\delta_1 \geq d_1$ or $\widetilde{\mathrm{Q}}(1)$ is a non zero polynomial of degree d_2 such that $\delta_1 \geq 2(d_1 - d_2)$. If not, we try increasing values of δ_1 (and $\bar{\delta}_1$), by steps of $2(d_0 - d_1)$, and update $\widetilde{\mathrm{V}}(1)$ by

$$\widetilde{\mathrm{V}}(1) := \widetilde{\mathrm{V}}(1) + \mathrm{M}(1) \cdot \mathrm{V}(0)_{]d_0 - \delta_1 \cdots d_0 - \bar{\delta}_1]},$$

until $\delta_1 \geq d_1$ or $\widetilde{\mathrm{Q}}(1)$ is a non zero polynomial of degree d_2 such that $\delta_1 \geq 2(d_1 - d_2)$. Since $\widetilde{\mathrm{V}}(1)$ coincides with $\mathrm{V}(1)$ up to degree $d_1 - \delta_1$, which is smaller than $2d_2 - d_1$, we can compute $\mathrm{C}(2)$ as

$$\mathrm{Quo}(\widetilde{\mathrm{P}}(1)_{]\cdots d_1]}, \widetilde{\mathrm{Q}}(1)_{]\cdots 2d_2 - d_1]}). \tag{24}$$

$\star$ For $i = 2$, since we do not know yet d_4, we start by acting as if $d_2 - d_4 \leq d_0 - d_2$. We initialize δ_2 to $2(d_0 - d_2)$ and define $\bar{\delta}_1 := \delta_1 + 2(d_0 - d_1)$. We define

$$\widetilde{\mathrm{V}}(2) := \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]\cdots d_0 - \bar{\delta}_2]}.$$

We note that $\widetilde{\mathrm{V}}(2)_{]\cdots d_2 - \delta_2]} = \mathrm{V}(2)_{]\cdots d_2 - \delta_2]}$, using Lemma 2.1 and check whether $\delta_2 \geq d_2$ or $\widetilde{\mathrm{Q}}(2)$ is a non zero polynomial of degree d_3 such that $\delta_2 \geq 2(d_2 - d_3)$. If not, we try increasing values of δ_2 (and $\bar{\delta}_2$) by steps of $2(d_0 - d_2)$, and update

$$\widetilde{\mathrm{V}}(2) := \widetilde{\mathrm{V}}(2) + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]d_0 - \delta_2 \cdots d_0 - \bar{\delta}_2]},$$

until $\delta_2 \geq d_2$ or $\widetilde{\mathrm{Q}}(2)$ is a non zero polynomial of degree d_3 such that $\delta_2 \geq 2(d_2 - d_3)$. Since $\widetilde{\mathrm{V}}(2)$ coincides with $\mathrm{V}(2)$ up to degree $d_2 - \delta_2$, which is smaller than $2d_3 - d_2$, we can compute $\mathrm{C}(3)$ as

$$\mathrm{Quo}(\widetilde{\mathrm{P}}(2)_{]\cdots d_2]}, \widetilde{\mathrm{Q}}(2)_{]\cdots 2d_3 - d_2]}). \tag{25}$$

$\star$ For $i = 3$, a new phenomenon occurs, since $\mathrm{f}(3) = \mathrm{f}(2) = 4$. We initialize δ_3 to $2(d_2 - d_3)$ and define $\bar{\delta}_3 := \delta_3 + 2(d_2 - d_3)$. We wish to define

$$\widetilde{\mathrm{V}}(3) := \mathrm{M}(3) \cdot \widetilde{\mathrm{V}}(2)_{]\cdots d_2 - \bar{\delta}_3]}.$$

This is possible only if $d_2 - \bar{\delta}_3 \geq d_2 - \delta_2$ i.e. $\delta_2 - \delta_3 \geq 2(d_2 - d_3)$. If not, we store $\bar{\delta}_2$ in $\tilde{\delta}_2$ and define

$$\delta_2 := 2(d_0 - d_2) \, \lceil \frac{2(d_2 - d_3) + \delta_3}{2(d_0 - d_2)} \rceil$$

Before defining $\widetilde{\mathrm{V}}(3)$, we update $\widetilde{\mathrm{V}}(2)$ i.e. we compute

$$\widetilde{\mathrm{V}}(2) := \widetilde{\mathrm{V}}(2) + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]d_0 - \tilde{\delta}_2 \cdots d_0 - \bar{\delta}_2]}.$$

We then define

$$\widetilde{\mathrm{V}}(3) := \mathrm{M}(3) \cdot \widetilde{\mathrm{V}}(2)_{]\cdots d_2 - \bar{\delta}_3]},$$

and check whether $\delta_3 \geq d_3$ or $\widetilde{\mathrm{Q}}(3)$ is a non zero polynomial of degree d_4 such that $\delta_3 \geq 2(d_3 - d_4)$.

If not we increase δ_3 (and $\bar{\delta}_3$) by $2(d_2 - d_3)$ and wish to update

$$\widetilde{\mathrm{V}}(3) := \widetilde{\mathrm{V}}(3) + \mathrm{M}(3) \cdot \mathrm{V}(2)_{]d_2 - \delta_3 \cdots d_2 - \bar{\delta}_3]}.$$

This is possible only if $2d_2 - \bar{\delta}_3 \geq d_2 - \delta_2$ i.e. $\delta_2 - \delta_3 \geq 2(d_2 - d_3)$. If not, we store $\bar{\delta}_2$ in $\tilde{\delta}_2$ and define

$$\delta_2 := 2(d_0 - d_2) \, \lceil \frac{2(d_2 - d_3) + \delta_3}{2(d_0 - d_2)} \rceil$$

Before updating $\widetilde{\mathrm{V}}(3)$, we update $\widetilde{\mathrm{V}}(2)$ i.e. we compute

$$\widetilde{\mathrm{V}}(2) := \widetilde{\mathrm{V}}(2) + \mathrm{M}(2,1) \cdot \mathrm{V}(0)_{]d_0 - \tilde{\delta}_2 \cdots d_0 - \bar{\delta}_2]}.$$

We then update

$$\widetilde{\mathrm{V}}(3) := \widetilde{\mathrm{V}}(3) + \mathrm{M}(3) \cdot \mathrm{V}(2)_{]d_2 - \delta_3 \cdots d_2 - \bar{\delta}_3]},$$

and check whether $\delta_3 \geq d_3$ or $\widetilde{\mathrm{Q}}(3)$ is a non zero polynomial of degree d_4 such that $\delta_3 \geq 2(d_3 - d_4)$.

Note that at the end of this loop $\widetilde{\mathrm{V}}(3)$ coincides with $\mathrm{V}(3)$ up to degree $d_3 - \delta_3$, which is smaller than $2d_4 - d_3$. So, we can compute $\mathrm{C}(4)$.

Description and correctness of Algorithm B. In the general case we have the following algorithm.

Algorithm 3.1 (B, Quotient boot)

- *($\star_0$) Initialize* $\widetilde{\mathrm{V}}(0) := \mathrm{V}(0)$, $i := 1$.
- **While** $\widetilde{\mathrm{Q}}(i-1) \neq 0$, *with*

$$\widetilde{\mathrm{V}}(i-1) := \begin{bmatrix} \widetilde{\mathrm{P}}(i-1) \\ \widetilde{\mathrm{Q}}(i-1) \end{bmatrix},$$

  – *Define*

$$\begin{array}{rcl} d_i & := & \deg(\widetilde{\mathrm{Q}}(i-1)) \\ \widetilde{\mathrm{C}}(i) & := & \mathrm{Quo}(\widetilde{\mathrm{P}}(i-1)_{]\cdots d_i]}, \widetilde{\mathrm{Q}}(i-1)_{]\cdots 2d_i - d_{i-1}]}), \\ \widetilde{\mathrm{M}}(i,0) & := & \begin{bmatrix} 0 & 1 \\ 1 & -\widetilde{\mathrm{C}}(i) \end{bmatrix}. \end{array}$$

– *If i is even, compute for $\ell = 1 \cdots \mathrm{v}(i) - 1$,*

$$\widetilde{\mathrm{M}}(i, \ell+1) := \widetilde{\mathrm{M}}(i,\ell) \cdot \widetilde{\mathrm{M}}(i - 2^{\ell}, \ell),$$

– *Initialize* $\delta_i := 2(d_{\mathrm{p}(i)} - d_i)$

– **Repeat** *(Loop i)*

∗ $\delta_i := \bar{\delta}_i$, $\bar{\delta}_i := \delta_i + 2(d_{\mathrm{p}(i)} - d_i)$

∗ $I := \emptyset$, $j := i$, $\ell := \mathrm{p}(j)$, $m := \mathrm{p}(\ell)$,

∗ **While** $\ell > 0$ **and** $\delta_\ell - \delta_j < 2(d_\ell - d_j)$

· *Update*

$$\begin{array}{rcl} \tilde{\delta}_\ell & := & \bar{\delta}_\ell \\ \delta_\ell & := & 2(d_m - d_\ell) \left\lceil \frac{2(d_\ell - d_j) + \delta_j}{2(d_m - d_\ell)} \right\rceil, \\ \bar{\delta}_\ell & := & \delta_\ell + 2(d_m - d_\ell). \end{array}$$

· $I := \ell, I$; $j := \ell$, $\ell := m$, $m := \mathrm{p}(\ell)$.

∗ **EndWhile**

∗ **While** $I \neq \emptyset$

· *Remove from I its first element j.*

· *($\star_a$) For δ from $\tilde{\delta}_j$ to δ_j by steps of $2(d_{\mathrm{p}(j)} - d_j)$, and $\bar{\delta} = \delta + 2(d_{\mathrm{p}(j)} - d_j)$ update $\widetilde{\mathrm{V}}(j)$ by adding*

$$\widetilde{\mathrm{M}}(j, \mathrm{v}(j)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(j))_{]d_{\mathrm{p}(j)} - \delta \cdots d_{\mathrm{p}(j)} - \bar{\delta}]}$$

∗ **EndWhile**

∗ *($\star_b$) If $\delta_i = 2(d_{\mathrm{p}(i)} - d_i)$*

· *If $d_i = d_i$ and i is even, define $\widetilde{\mathrm{V}}(i)$ as*

$$\begin{array}{l} \widetilde{\mathrm{M}}(i,0) \cdot \widetilde{\mathrm{V}}(i-1) \\ + \sum_{k=1}^{\mathrm{v}(i)-1} \widetilde{\mathrm{M}}(i,k) \cdot \widetilde{\mathrm{V}}(i - 2^k)_{]d_{i-2^k} - \bar{\delta}_{i-2^k-1} \cdots]} \\ + \widetilde{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]d_{\mathrm{p}(i)} - \bar{\delta}_{i-2^{\mathrm{v}(i)-1}} \cdots d_{\mathrm{p}(i)} - \bar{\delta}_i]}. \end{array}$$

· *Otherwise, define $\widetilde{\mathrm{V}}(i)$ as*

$$\widetilde{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]\cdots d_{\mathrm{p}(i)} - \bar{\delta}_i]} \tag{26}$$

∗ *($\star_c$) Otherwise update $\widetilde{\mathrm{V}}(i)$ by adding*

$$\widetilde{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]d_{\mathrm{p}(i)} - \delta_i \cdots d_{\mathrm{p}(i)} - \bar{\delta}_i]}.$$

– **Until** $\delta_i \geq 2(d_i - \deg(\widetilde{\mathrm{Q}}(i)))$ *or* $\delta_i \geq d_i$

– $i := i + 1$

• **EndWhile**

- *Define* $G := \widetilde{\mathrm{P}}(i-1)$.

Remark 3.2 *Note that if for every i from 0 to n, $d_{\mathrm{p}(i)} - d_i = d_i - d_{\mathrm{f}(i)}$, which is always the case in the non defective case where $n = d$, $d_i = d - i$, Algorithm B coincides with Algorithm A_2.*

We now prove the correctness of Algorithm B which is the following result:

Proposition 3.3

$$\widetilde{\mathrm{C}}(i) = \mathrm{C}(i).$$

Let us denote by γ_i the value of δ_i at the end of Loop i. Since $\gamma_i \leq 2(d_{i+1} - d_{i)}$, we have

$$d_i - \gamma_i \leq 2d_{i+1} - d_i,$$

and taking into account Equation 15, the proposition is a consequence of the following technical lemma:

Lemma 3.4 *After step ($\star_0$), $\mathrm{V}(0) = \widetilde{\mathrm{V}}(0)$.*
At the end of each step ($\star_a$) in Algorithm B

$$\widetilde{\mathrm{V}}(j)_{]\cdots d_j - \delta_j]} = \mathrm{V}(j)_{]\cdots d_j - \delta_j]}.$$

At the end of each step ($\star_b$) and ($\star_c$) in Algorithm B

$$\widetilde{\mathrm{V}}(i)_{]\cdots d_i - \delta_i]} = \mathrm{V}(i)_{]\cdots d_i - \delta_i]}.$$

As a consequence at the end of Loop i, $\widetilde{\mathrm{V}}(i)$ and $\mathrm{V}(i)$ coincide up to degree $d_i - \gamma_i$ and Proposition 3.3 holds.

Lemma 3.5 *Throughout Algorithm B, for every $j > 0$,*

$$\widetilde{\mathrm{V}}(j) = \mathrm{M}(j, \mathrm{v}(j)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)} - \bar{\delta}_j]}.$$

Proof : We prove that the claim holds by induction on the successive definitions and updates of the various $\widetilde{\mathrm{V}}(j)$ throughout Algorithm B.

- ($\star_a$): After the update,

  $$\widetilde{\mathrm{V}}(j) = \mathrm{M}(j, \mathrm{v}(j)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)} - \bar{\delta}_j]},$$

  since before the update

  $$\widetilde{\mathrm{V}}(j) = \mathrm{M}(j, \mathrm{v}(j)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)} - \bar{\delta}_j]}.$$

- ($\star_b$) The claim is true by Equation (26), and if i is even and $d_i = d_i$, bt Lemma 2.5.

- ($\star_c$) The proof is similar to ($\star_a$).

QED

Proof of Lemma 3.3: We prove that the claim holds by induction on the successive definitions and updates of the various $\widetilde{\mathrm{V}}(j)$ throughout Algorithm B.

- ($\star_0$) The base case is clear since $\widetilde{\mathrm{V}}(0) = \mathrm{V}(0)$.
- ($\star_a$) By induction hypothesis

$$\widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)}-\delta_{\mathrm{p}(j)}]} = \mathrm{V}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)}-\delta_{\mathrm{p}(j)}]}.$$

Noticing that

$$\delta_{\mathrm{p}(j)} \geq \bar{\delta}_j,$$

since the condition

$$\delta_{\mathrm{p}(j)} - \delta_j \geq \bar{\delta}_j - \delta_j = 2(d_{\mathrm{p}(j)} - d_j)$$

is ensured, we obtain

$$\widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)}-\bar{\delta}_j]} = \mathrm{V}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)}-\bar{\delta}_j]}.$$

Note that

$$\mathrm{V}(j) = \mathrm{M}(j, \mathrm{v}(j)) \cdot \mathrm{V}(\mathrm{p}(j)),$$

and that, by Lemma 3.5,

$$\widetilde{\mathrm{V}}(j) = \mathrm{M}(j, \mathrm{v}(j)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(j))_{]\cdots d_{\mathrm{p}(j)}-\bar{\delta}_j]}.$$

Since, by induction hypothesis,

$$\mathrm{C}(k) = \widetilde{\mathrm{C}}(k) = \mathrm{Quo}(\widetilde{\mathrm{P}}(k-1)_{]\cdots d_k]}, \widetilde{\mathrm{Q}}(k-1)_{]\cdots 2d_k - d_{k-1}]})$$

for every $k \leq j$, the degree of the entries of $\mathrm{M}(j, \mathrm{v}(j))$ are $d_{\mathrm{p}(j)} - d_j$, and using Lemma 2.1,

$$\widetilde{\mathrm{V}}(j)_{]\cdots d_j-\delta_j]} = \mathrm{V}(j)_{]\cdots d_j-\delta_j]},$$

since $d_j - \delta_j = d_{\mathrm{p}(j)} - d_j + d_{\mathrm{p}(j)} - \bar{\delta}_j$.

QED

Remark 3.6 *a) In the non defective case (i.e. when $d_i = d - i$ for all i), Algorithm B coincides with Algorithm A_2.*

b) Algorithm B is a half-gcd algorithm in the following sense. Let ν the number of bits of n, and $2^{\nu-1}$ the biggest power of 2 stricty smaller than n. At step $2^{\nu-1}$, Algorithm B compute

$$\widetilde{\mathrm{V}}(2^{\nu-1}) = \mathrm{V}(2^{\nu-1}) = \mathrm{M}(2^{\nu-1}, \nu - 1) \cdot \mathrm{V}(0)$$

and then calls itself with input $\mathrm{V}(2^{\nu-1}) = \widetilde{\mathrm{V}}(2^{\nu-1})$, without needing to record any of the preceeding computations.

4 Complexity analysis

4.1 Notation and preliminary remarks

Let $M(n)$ be the cost of a multiplication of two polynomials of degree n i.e. the maximum number of arithmetic operations performed in the base field by a given multiplication algorithm over any input polynomials of degree n. Note that if $M(n,m)$ is the cost of a multiplication of a polynomials of degree n by a polynomial of degree m, then $M(n,m) \leq M(\max(n,m))$.

We suppose that the function $M(n)$ satisfies the following properties

i)	$\frac{M(2^k)}{2^k} \leq \frac{M(n)}{n}$	whenever $2^k \leq n$
ii)	$\sum_{k=1}^{\nu-1} M(2^k) \leq M(2^\nu)$	
iii)	$\sum_{k=1}^{\nu-1} \frac{M(2^k)}{2^k} \leq \frac{\nu-1}{2} \frac{M(2^\nu)}{2^\nu}$	
iv)	$\sum_{i=1}^{n} M(\max(a_i,b_i)) \leq M(d)$,	whenever $\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i \leq d$.
v)	$\sum_{i=1}^{n} M(\max(a_i,b_i)) \leq M(d)$,	whenever $\max(a_i,b_i) \leq c_i$, $\sum_{i=1}^{n} c_i \leq d$.

These properties are easy to check for the functions $n \log n$ and $n \log n \log\log n$ which correspond to the cost of FFT multiplication in a field supporting FFT and general FFT multiplication in any field [2].

The cost of the multiplication of two 2×2 matrices with entries polyniomials of degree n takes 7 multiplications and a fixed number of additions of polynomials of degree n, and costs $7M(n) + O(n)$ using Strassen fast multiplication of matrices [6, 2]. The cost of the multiplication of a 2×2 matrix by a vector of length two with entries polyniomials of respective degree n is $4M(n)+O(n)$. The cost of the multiplication of a 2×2 matrix by a vector of length two with entries polyniomials of respective degree n and $2n$ is $7M(n) + O(n)$, since the vector can be split in two parts of degree n and we can perform the multiplication of two 2×2 matrices and a few additions.

We denote by $n+1$ the number of elements in the remainder sequence of P(0) and P(1), by ν the number of bits of n, $\nu = \lfloor \log_2 n \rfloor + 1$, i.e. $2^{\nu-1} \leq n < 2^\nu$.

4.2 Complexity analysis in the non-defective case

This is the case where $d_i = d - i$ and $n = d$.

Proposition 4.1 *The complexity t_{A_1} of Algorithm A_1, in the non-defective case is*

$$t_{A_1}(d) = \left(\frac{21}{4} M(d) + O(d)\right) \log_2 d + O(M(d)).$$

Proof : Let k be an integer between 1 and $\nu - 1$.

The cost of the computation of all the C(i), $1 \leq i \leq d$ takes $O(d)$ multiplications and divisions in the base field.

For every multiple i of 2^k, $i < d$ the computation of M(i,k) as

$$\mathrm{M}(i, k-1) \cdot \mathrm{M}(i - 2^{k-1}, k-1)$$

takes 7 multiplications and a fixed number of additions of polynomials of degree 2^{k-1}.The number of multiples of 2^k which are less than d is

$$\lfloor \frac{d-1}{2^k} \rfloor < \frac{d}{2^k}.$$

So the complexity of the computation of all the $\mathrm{M}(i,k)$, $i < d, k \leq \mathrm{v}(i)$ is

$$7\,\frac{d}{2}\sum_{k=1}^{\nu-1}\frac{M(2^{k-1})}{2^{k-1}} = 7\,\frac{d}{2}\sum_{k=1}^{\nu-2}\frac{M(2^k)}{2^k} + O(d) \leq 7\,\frac{d}{2}\;\frac{\nu-2}{2}\;\frac{M(2^{\nu-1})}{2^{\nu-1}} + O(d)$$

which is smaller than

$$\frac{7}{4}(\nu-2)M(d) + O(d).$$

using i) and iii).

For every $i \leq n$ such that $\mathrm{v}(i) = k < \nu - 1$, the computation of

$$\overline{\mathrm{V}}(i) := \mathrm{M}(i, \mathrm{v}(i)) \cdot \overline{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}]}$$

takes 14 multiplications and a fixed number of additions of polynomials of degree 2^k, since $2(d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}) = 4 \times 2^k$. The number of natural numbers $i < d$ such that $\mathrm{v}(i) = k$ is at most

$$\frac{d + 2^k - 1}{2^{k+1}}.$$

So the complexity of the computation of all the $\overline{\mathrm{V}}(i)$, $i \leq d, \mathrm{v}(i) < \nu - 1$ is

$$14\left(\frac{d}{2}\sum_{k=1}^{\nu-2}\frac{M(2^k)}{2^k} + \frac{1}{2}\sum_{k=1}^{\nu-2}M(2^k)\right) \leq 7\,d\;\frac{\nu-2}{2}\;\frac{M(2^{\nu-1})}{2^{\nu-1}} + 7M(2^{\nu-1})$$

whihch is smaller than

$$\frac{7}{2}(\nu-2)M(d) + 7M(d),$$

using i), ii) and iii).

Finally, for $i = 2^{\nu-1}$, $\mathrm{p}(i) = 0$, and the computation of

$$\overline{\mathrm{V}}(i) := \mathrm{M}(i, \nu - 1) \cdot \mathrm{V}(0)$$

takes 7 multiplications and a fixed number of additions of polynomials of degree $2^{\nu-1}$, since $d \leq 2 \times 2^{\nu-1}$.

The cost of the polynomial multiplications performed evaluated in the base field is bounded by

$$(\nu-2)\left(\frac{21}{4}\;M(d) + O(d)\right) + 14M(d) \leq (\nu+1)\left(\frac{21}{4}\;M(d) + O(d)\right).$$

QED

Proposition 4.2 *The complexity t_{A_2} of Algorithm 2.7 in the non-defective case is*

$$t_{A_2}(d) = \left(\frac{9}{2}M(d) + O(d)\right)\log_2 d.$$

Proof : Let k be an integer between 1 and $\nu - 1$.

The complexity of the computation of all the $\mathrm{M}(i,k)$, $i \leq d, k \leq \mathrm{v}(i)$ is

$$(\nu - 2)\left(\frac{7}{4}M(d) + O(d)\right),$$

as in the proof of Proposition 4.1.
For every i such that $k < \mathrm{v}(i) \leq \nu - 1$, the computation of the matrix-vector multiplication

$$\mathrm{M}(i,k) \cdot \overline{\mathrm{V}}(i - 2^k)_{]2d_i - d_{i-2^k} \cdots]}$$

costs 4 multiplications and a fixed number of additions of polynomials of degree 2^k, since $d_{i-2^{k+1}} - d_{i-2^k} = 2^k$.

So the complexity of the computation of all the $\mathrm{M}(i,k)\overline{\mathrm{V}}(i - 2^k)_{]2d_i - d_{i-2^k} \cdots]}$ is

$$(\nu - 2)\left(M(d) + O(d)\right),$$

similarly to the proof of Proposition 4.1.

For every $i \leq d$ such that $\mathrm{v}(i) = k < \nu - 1$, the computation of

$$\mathrm{M}(i, \mathrm{v}(i)) \cdot \overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i - d_{\mathrm{p}(i)} \cdots 2d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}]}$$

takes 7 multiplications and a fixed number of additions of polynomials of degree 2^k, since $2(d_{\mathrm{f}(i)} - d_i) = 2 \times 2^k$.

So the complexity of the computation of all the

$$\mathrm{M}(i, \mathrm{v}(i)) \cdot \overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i - d_{\mathrm{p}(i)} \cdots 2d_{\mathrm{f}(i)} - d_{\mathrm{p}(i)}]},$$

$i \leq n, \mathrm{v}(i) < \nu - 1$ is

$$\frac{7}{4}(\nu - 2)M(d) + \frac{7}{2}M(d),$$

similarly to the proof of Proposition 4.1.

Finally, for $i = 2^{\nu-1}$, $\mathrm{p}(i) = 0$, and the computation of

$$\overline{\mathrm{V}}(i) := \mathrm{M}(i, \nu - 1) \cdot \overline{\mathrm{V}}(0)_{]2d_i - d_0 \cdots]}$$

takes 4 multiplications and a fixed number of additions of polynomials of degree $2^{\nu-1}$, since $n \leq 2 \times 2^{\nu-1}$, as in the proof of Proposition 4.1.

The cost of the polynomial multiplications performed evaluated in the base field is bounded by

$$\frac{9}{2}(\nu - 2)\left(M(d) + O(d)\right) + \frac{15}{2}M(d) \leq \frac{9}{2}\nu\left(M(d) + O(d)\right).$$

It is finally easy to see that the total cost of the additions for computing

$$\begin{aligned}\overline{\mathrm{V}}(i) &= \mathrm{M}(i,0)\cdot\overline{\mathrm{V}}(i-1)\\ &\quad +\textstyle\sum_{k=1}^{\mathrm{v}(i)-1}\mathrm{M}(i,k)\cdot\overline{\mathrm{V}}(i-2^k)_{]2d_i-d_{i-2^k}\cdots]}\\ &\quad +\mathrm{M}(i,\mathrm{v}(i))\cdot\overline{\mathrm{V}}(\mathrm{p}(i))_{]2d_i-d_{\mathrm{p}(i)}\cdots 2d_{\mathrm{f}(i)}-d_{\mathrm{p}(i)}]}.\end{aligned}$$

for every i such that $\mathrm{v}(i)=k$ is bounded by $O(d)$. *QED*

Since Algorithm 3.1 and Algorithm 2.7 coincide in the defective case, we have

Corollary 4.3 *The complexity t_B of Algorithm 3.1 in the non-defective case is*

$$\left(\frac{9}{2}M(d)+O(d)\right)\log_2 d.$$

Note that the constant $9/2$ that we obtain for Algorithm 3.1 improves significantly the previously known constant which was 10 [2].

4.3 Complexity analysis of Algorithm B (Algorithm 3.1).

Proposition 4.4 *The complexity of Algorithm 3.1 is* $21\,(M(d)+O(d))\,\log_2 d+O(M(d))$.

The proof of Proposition 4.4 uses the following Lemma

Lemma 4.5 *Denoting by Δ_i the end value of δ_i computed in Algorithm 3.1,*

$$\Delta_i \le 2(d_{\mathrm{p}(i)}-d_{\mathrm{f}(i)})$$

Proof : Let j be the biggest number such that δ_i is modified during the part Loop j of Algorithm 3.1. It is clear that $i\le j\le \mathrm{f}(i)-1$.

We first prove that denoting by γ_j the value of δ_j obtained at the end of Loop j of Algorithm 3.1

$$\gamma_j < 2(d_j-d_{j+1})+2(d_{\mathrm{p}(j)}-d_j) \tag{27}$$

Indeed if $\gamma_j\ge 2(d_j-d_{j+1})$ then $\deg(\widetilde{\mathrm{Q}}(j))=d_{j+1}$, because $\widetilde{\mathrm{V}}(i)$coincides with $\mathrm{V}(i)$ up to degree $d_j-\gamma_j\le 2d_{j+1}-d_j\le d_{j+1}$. But it was not the case that the previous value of δ_j, $\gamma_j-2(d_{\mathrm{p}(j)}-d_j)$ was strictly less than $2(d_j-\deg(\widetilde{\mathrm{Q}}(j))$, since the Loop j did not stop at $\delta_j=\gamma_j-2(d_{\mathrm{p}(j)}-d_j)$.

Let $\ell\le \mathrm{v}(i)$ be such that $\mathrm{p}^{(\ell)}(j)=i$ and let, for $n\le\ell$, $\bar\gamma_{\mathrm{p}^{(n)}(j)}$ be the value of $\delta_{\mathrm{p}^{(n)}(j)}$ obtained at the end of Loop j of Algorithm 3.1. It follows clearly from Algorithm 3.1 that

$$\begin{aligned}\bar\gamma_{\mathrm{p}(j)}-\gamma_j &\le 2(d_{\mathrm{p}(j)}-d_j)+2(d_{\mathrm{p}(\mathrm{p}(j))}-d_{\mathrm{p}(j)})\\ &\cdots\\ \Delta_i-\bar\gamma_{\mathrm{p}^{(\ell-1)}(j)} &\le 2(d_i-d_{\mathrm{p}^{(\ell-1)}(j)})+2(d_{\mathrm{p}(i)}-d_i)\end{aligned}$$

Summing, we get

$$\Delta_i - \gamma_j \quad \leq \quad 2(d_i - d_j) + 2(d_{\mathrm{p}(i)} - d_{\mathrm{p}(j)})$$

and taking into account Inequality 27 we get

$$\Delta_i \leq 2(d_i - d_{j+1}) + 2(d_{\mathrm{p}(i)} - d_j).$$

The claim follows immediately noticing that, since $i \leq j \leq \mathrm{f}(i) - 1$,

$$\Delta_i \leq 2(d_i - d_{\mathrm{f}(i)}) + 2(d_{\mathrm{p}(i)} - d_i)$$

QED

Proof of Proposition 4.4: • Using property *ii)* and fast division with remainder (see [2] page 247), the computation of $\mathrm{C}(i-1)$ for every $i \leq k$ costs $\sum_{i=1}^{k} 6M(d_{i-1} - d_i) \leq 6M(d) + O(d)$, since $\sum_{i=1}^{k}(d_{i-1} - d_i) \leq d$.

• For every ℓ less than $\nu - 1$, the computation of all the

$$\mathrm{M}(i, \ell) := \mathrm{M}(i, \ell - 1) \cdot \mathrm{M}(i - 2^{\ell-1}, \ell - 1),$$

for every multiple i of 2^ℓ less than n costs $7M(d) + O(d)$.

Indeed, for every multiple i of 2^ℓ the computation of

$$\mathrm{M}(i, \ell) := \mathrm{M}(i, \ell - 1) \cdot \mathrm{M}(i - 2^{\ell-1}, \ell - 1),$$

takes 7 multiplications and a fixed number of additions of polynomials of degree $d_{i-2^{k-1}} - d_i$ by polynomials of degree $d_{i-2^{k-1}} - d_{i-2^k}$. The claim follows, using *iv*).

• For every ℓ less than $\nu - 1$, the computation of all the

$$\widetilde{\mathrm{V}}(i) := \mathrm{M}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_i - d_{\mathrm{p}(i)} - \Gamma_i]}$$

for every i of such that $\mathrm{v}(i) = \ell$ costs $14M(d) + O(d)$. Indeed, for every i such that $\mathrm{v}(i) = \ell$, the computation of

$$\widetilde{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_i - d_{\mathrm{p}(i)}]}$$

in $(\star)_b$ costs 7 multiplications and a fixed number of additions of polynomials of degree $d_{\mathrm{p}(i)} - d_i$ if $d_i \neq d - i$. Moreover the computation of

$$\widetilde{\mathrm{M}}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]2d_i - d_{\mathrm{p}(i)} \cdots 2d_i - d_{\mathrm{p}(i)} - \Delta_i]}$$

costs 7 multiplications of polynomials and a fixed nomber of additions of polynomials of degree $d_{\mathrm{p}(i)} - d_{\mathrm{f}(i)}$ since $\Gamma_i \leq 2(d_{\mathrm{p}(i)} - d_{\mathrm{f}(i)})$. So the computation of

$$\widetilde{\mathrm{V}}(i) := \mathrm{M}(i, \mathrm{v}(i)) \cdot \widetilde{\mathrm{V}}(\mathrm{p}(i))_{]\cdots 2d_i - d_{\mathrm{p}(i)} - \Gamma_i]}$$

costs at most 14 multiplications of polynomials of degree $d_{\mathrm{p}(i)} - d_{\mathrm{f}(i)}$. If i is even and $d_i = d - i$, the result is also true by the complexity analysis of agorithm A_2.

The claim follows from iv) and v).

• Finally since $\nu \leq \log_2 d + 1$ the cost of the multiplications is bounded by

$$21\,(\,M(d) + O(d)\,)\,\log_2 d + O(M(d)),$$

while the number of additions is $O(d)\log_2 d$. QED

Note that the constant 21 that we obtain for Algorithm 3.1 improves slightly the previously known constant which was 24 [2].

References

[1] S. Basu, R. Pollack, M.-F. Roy, Algorithms in real algebraic geometry, Springer (2003), revised version http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html, (2005)

[2] J. von zur Gathen, J. Gerhard, Modern Computer Algebra, Cambridge University Press (1999)

[3] T. Lickteig, M.-F. Roy, Sylvester-Habicht sequences and fast Cauchy index computation, Journal of Symbolic Computation, 31 315-341 (2001)

[4] R. T. Moenck, Fast computation of GCDs, Proc. STOC '73, 142–151 (1973)

[5] A. Schönhage, Schnelle Berechnung von Kettenbruchentwicklugen, Acta Informatica, 1, 139-144 (1971)

[6] V. Strassen, Gaussian elimination is not optimal, Numerische mathematik, 13, 354-356 (1969)