In-place fast polynomial modular remainder

Jean-Guillaume Dumas, Bruno Grenet

In-place fast polynomial modular remainder

Jean-Guillaume Dumas* Bruno Grenet*

October 24, 2023

Abstract

We consider the fast in-place computation of the Euclidean polynomial modular remainder $R(X) \equiv A(X) \mod B(X)$ with A and B of respective degrees n and $m \leq n$. If the multiplication of two polynomials of degree k can be performed with $\mathfrak{M}(k)$ operations and $\mathcal{O}(k)$ extra space, then standard algorithms for the remainder require $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ arithmetic operations and, apart from that of A and B, at least $\mathcal{O}(n-m)$ extra memory. This extra space is notably usually used to store the whole quotient $Q(X)$ such that $A = BQ + R$ with $\deg R < \deg B$.

We avoid the storage of the whole of this quotient, and propose an algorithm still using $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ arithmetic operations but only $\mathcal{O}(m)$ extra space.

When the divisor B is sparse with a constant number of non-zero terms, the arithmetic complexity bound reduces to $\mathcal{O}(n)$.

When it is allowed to use the input space of A or B for intermediate computations, but putting A and B back to their initial states after the completion of the remainder computation, we further propose an in-place algorithm (that is with its extra required space reduced to $\mathcal{O}(1)$ only) using at most $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\log(m)\big)$ arithmetic operations, if $\mathfrak{M}(m)$ is quasi-linear, or $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ otherwise. We also propose variants that compute – still in-place and with the same complexity bounds – the over-place remainder $A(X) \equiv A(X) \mod B(X)$ and the accumulated remainder $R(X) \mathrel{+}= A(X) \mod B(X)$.

To achieve this, we develop techniques for Toeplitz matrix operations which output is also part of the input. In-place accumulating versions are obtained for the latter and then used for polynomial remaindering. This is realized via further reductions to accumulated polynomial multiplication, for which in-place fast algorithms have recently been developed.

Contents

*Université Grenoble Alpes. Laboratoire Jean Kuntzmann, CNRS, UMR 5224. 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble, cedex 9 France. {firstname.lastname}@univ-grenoble-alpes.fr

1 Introduction

Modular methods with dense univariate polynomials over a finite ring are of central importance in computer algebra and symbolic computation. For instance, they are largely used with lifting or Chinese remaindering techniques as the bases to compute at a larger precision. There, the quotient of the Euclidean division is not needed, but is very often computed anyway along the algorithm.

In terms of arithmetic operations, from the work of [17, 15] to the more recent results of [3], many sub-quadratic algorithms were developed for this task, based on fast polynomial multiplication [5, 14, 20]. But these fast algorithms come at the expense of (potentially large) extra temporary space to perform the computation. On the contrary, classical, quadratic algorithms, when computed sequentially, quite often require very few (constant) extra registers. Further work then proposed simultaneously "fast" and "in-place" algorithms, for both matrix and polynomial operations [4, 10, 11].

We here propose algorithms to extend the latter line of work. In the polynomial setting, we compute in-place the remainder only of the Euclidean division. This means that, e.g., with respect to [11, Alg. 3], we obtain the remainder without needing any space for the quotient. As polynomials and Toeplitz matrices are indeed different representations of the same objects, see, e.g., [1, 2, 10], we also develop methods for Toeplitz matrix operations, in-place *with accumulation* or *over-place*: those are difficult cases where the output is also part of the input. The difficulty when the result overwrites (parts) of the input, is that in-place methods start with absolutely no margin for extra space. Thus, for instance, the generic recursive techniques of [4, 10] usually do not apply. Instead we propose reductions to accumulated polynomial multiplication, for which a recent fast in-place algorithm has been developed [9, § 4].

Next we first detail our model for in-place computations in Section 1.1, recall some quadratic in-place algorithms in Section 1.2 and some fast algorithms in Sections 1.3 and 1.4. Then, in Sections 2 and 3, we derive new in-place algorithms for circulant and Toeplitz matrices. In Section 4 we then propose a fast remaindering algorithm using only a multiple of the remainder space and finally we combine all the reductions to obtain fast and in-place algorithms computing just the polynomial remainder.

1.1 In-place model

There exist different models for in-place algorithms. We here choose to call *in-place* an algorithm using only **the space of its inputs, its outputs, and at most $\mathcal{O}(1)$ extra space**. But algorithms are only allowed to modify their inputs, **if their inputs are restored to their initial state** afterwards (obviously those that are not both input and output). This is a less powerful model than when the input is purely read-only, but it turns out to be crucial in our case, especially when we have accumulation operations.

The algorithms we describe are *in-place with accumulation*. The archetypical example is a multiply-accumulate operation $a \mathrel{+}= b \times c$. For such an algorithm, the condition is that b and c are restored to their initial states at the end of the computation, while a (which is also part of the input) is replaced by $a + bc$.

Also, as a variant, by *over-place*, we mean an algorithm where the output replaces (parts of) its input (e.g., like $\vec{a} = b \cdot \vec{a}$). Similarly, we allow all of the input to be modified, provided that the parts of the input that are not the output are restored afterwards. In the following we signal by a "**Read-only:**" tag the parts of the input that the algorithm is not allowed to modify (the other parts are modifiable as long as they are restored). Note that in-place algorithms with accumulation are a special case of over-place algorithms.

For recursive algorithms, some space may be required to store the recursive call stack. (This space is bounded by the recursion depth of the algorithms, and managed in practice by the compiler.) Nonetheless, we call *in-place* a recursive algorithm whose only extra space is the call stack.

The main limitations of this model are for black-box inputs, or for different inputs whose representations share some data. For more details on these models, we refer to [19, 10].

1.2 In place and over-place quadratic algorithms

We recall here that classical, quadratic, algorithms for polynomial remaindering and triangular matrix operations can be performed in-place. For any ring $\mathbb{D}$ we have for instance the following over-place algorithms for triangular matrix operations, given in Alg. 1.

Algorithm 1 Over-place quadratic triangular matrix operations
(left: matrix-vector multiplication; right: triangular system solve)

Input: $U \in \mathbb{D}^{m\times m}$ upper triangular and $\vec{v} \in \mathbb{D}^m$
Read-only: U

Output: $\vec{v} \leftarrow U \cdot \vec{v}$
1: **for** $i = 1$ **to** m **do**
2: **for** $j = 1$ **to** $i - 1$ **do**
3: $v_j \mathrel{+}= U_{j,i} v_i$
4: **end for**
5: $v_i \leftarrow U_{i,i} v_i$;
6: **end for**

Output: $\vec{v} \leftarrow U^{-1} \cdot \vec{v}$
1: **for** $i = m$ **down-to** 1 **do**
2: **for** $j = m$ **down-to** $i + 1$ **do**
3: $v_i \mathrel{-}= U_{i,j} v_j$
4: **end for**
5: $v_i \leftarrow U_{i,i}^{-1} v_i$; {If $U_{i,i} \in \mathbb{D}^*$}
6: **end for**

For any vector $\vec{v} = [v_1, \ldots, v_m]$, we associate the polynomial of degree $\leq m - 1$, with these coefficients: $V(X) = \sum_{i=0}^{m-1} v_{i+1} X^i$. We then give a quadratic in-place remaindering in Alg. 2, as the classical long division.

Algorithm 2 In-place quadratic polynomial remainder

Input: $\vec{a} \in \mathbb{D}^N$, $\vec{b} \in \mathbb{D}^M$ with $b_M \in \mathbb{D}^*$
Read-only: $\vec{a}, \vec{b}$
Output: $R(X) = A(X) \mod B(X)$
1: **if** $N < M$ **then return** $R(X) = A(X) = \sum_{i=0}^{N-1} a_{i+1} X^i$ **end if**
2: Let $n = N - M + 1$ and $\vec{r} = [a_{n+1}, \ldots, a_N]$;
3: **for** $i = n$ **down-to** 1 **do**
4: Let $\vec{r} = [a_i, r_1, \ldots, r_{M-1}]$;
5: $r_M \leftarrow r_M \cdot b_M^{-1}$; {Local quotient is $r_M \cdot b_M^{-1}$}
6: $\vec{r} \mathrel{-}= r_M \cdot \vec{b}$; {Except for last entry (of index M)}
7: **end for**
8: **return** $R(X) = \sum_{i=0}^{M-2} r_{i+1} X^i$.

By abuse of notation, we will often use vectors for polynomials and vice versa. For n coefficients, we will then speak of a polynomial or a vector of *size* n and *degree* $n - 1$.

1.3 Fast algorithms

More generally, if $\mathfrak{M}(M)$ is a sub-multiplicative complexity bound on an algorithm computing the multiplication of polynomials of degree M, we suppose in the following that there exists (not in-place) algorithms, see, e.g., [11, 14, 20], such that:

- $M \times M$ Toeplitz matrix-vector multiplication requires less than $\mathcal{O}(\mathfrak{M}(M))$ operations and $\mathcal{O}(M)$ extra space. More precisely, if $t \cdot M$ extra space is required, the matrix-vector multiplication requires less than $\lambda_t \mathfrak{M}(M)$ operations;
- $M \times M$ Triangular Toeplitz system solve requires also $\mathcal{O}(\mathfrak{M}(M))$ operations and $\mathcal{O}(M)$ extra space. More precisely, if $s \cdot M$ extra space is required, the system solving requires less than $\lambda_s \mathfrak{M}(M)$ operations.

Also, for sparse inputs, we suppose:

- $M \times M$ sparse Toeplitz (constant number of non-zero terms) matrix-vector multiplication requires less than $\mathcal{O}(M)$ operations and $\mathcal{O}(M)$ extra space;
- $M \times M$ Triangular sparse Toeplitz (constant number of non-zero terms) system solve requires less than $\mathcal{O}(M)$ operations and $\mathcal{O}(M)$ extra space.

1.4 In-place algorithms with accumulation

In an in-place setting, another difficulty arises when performing *accumulating* operations: the result is not even available to store intermediate computations. Recently, in [9], a bilinear technique was developed for the fast accumulated in-place polynomial multiplication, preserving an $\mathcal{O}(\mathfrak{M}(M))$ complexity bound. Thus we suppose in the following that there exists fast routines computing $C(X) \mathrel{+}= A(X)B(X)$ in-place.

Therefore, in this paper, we reduce other in-place accumulating routines (Toeplitz, circulant, convolutions operations) to this fundamental building block. This allows us, in fine, to obtain an in-place fast polynomial modular remainder.

2 Fast in-place convolution with accumulation

From algorithms for polynomial multiplications at a lower level, one can devise an algorithm for the generalized accumulated convolution $C \mathrel{+}= AB \mod (X^n - f)$. To compute such convolutions, one can reduce the computations to full in-place products. Those accumulated in-place polynomial multiplications (and their cost), will also be denoted by $\mathfrak{M}$. Then, the idea is to unroll a first recursive iteration and then to call any in-place accumulated polynomial multiplication on halves.

Algorithm 3 shows this, first with a Karatsuba-like iteration, using any in-place accumulated polynomial multiplications (e.g. classical, Karatsuba, Toom-k, DFT, etc.). Apart from the full polynomial multiplications, only a linear number of operations will be needed. Therefore, the overall complexity bound will be that of the accumulated polynomial multiplications.

Suppose indeed first that n is even, let $t = n/2$, and $A(X) = a_0(X) + X^t a_1(X)$, $B(X) = b_0(X) + X^t b_1(X)$, $C(X) = c_0(X) + X^t c_1(X)$ are all of degree $n - 1 = 2t - 1$ (so that all of a_0, a_1, b_0, b_1, c_0 and c_1 are of degree at most $t - 1$).

Then let $\tau_0 + X^t \tau_1 = a_0 b_1 + a_1 b_0$. Now, since $X^{2t} = X^n \equiv f \mod X^n - f$, we have that $C \mathrel{+}= AB \mod (X^n - f) = C(X) + a_0 b_0 + X^t \tau_0 + f \cdot \tau_1 + f \cdot a_1 b_1$. This can be computed with 4 full accumulated sequential product, each of degree no more than $2t - 2 = n - 2$; and by exchanging the lower and upper parts, when accumulating $a_0 b_1 X^t$ and $a_0 b_1 X^t$: this comes from the fact that: $(\tau_0 + X^t \tau_1) X^t \equiv X^t \tau_0 + f \cdot \tau_1 \mod X^n - f$.

A la Karatsuba, a more efficient version could use only 3 full polynomial products: let instead $m_0 = m_{00} + m_{01} X^t = a_0 b_0$, $m_1 = m_{10} + m_{11} X^t = (a_0 + a_1)(b_0 + b_1)$ and $m_2 = m_{20} + m_{21} X^t = a_1 b_1$ be these 3 full products, to be computed and accumulated in-place.

Then, again as $X^{2t} = X^n \equiv f \mod X^n$, we have that:

$$C \mathrel{+}= AB \mod (X^n - f) \text{ is } \begin{cases} c_0 + m_{00} + f m_{20} + f(m_{11} - m_{01} - m_{21}) + \\ (c_1 + m_{01} + f m_{21} + (m_{10} - m_{00} - m_{20})) X^t \end{cases} \tag{1}$$

Equation (1) is an in-place accumulating linear computation, in the sense of [9]: for $\vec{a} \in \mathbb{D}^m$, $\vec{b} \in \mathbb{D}^n$, $\vec{c} \in \mathbb{D}^s$; $\alpha \in \mathbb{D}^{t \times m}$, $\beta \in \mathbb{D}^{t \times n}$, $\mu \in \mathbb{D}^{s \times t}$, with no zero-rows in α, β, μ.

One indeed needs to compute $\vec{c} \mathrel{+}= \mu \vec{m}$, some linear combinations of the same t intermediate results $\vec{m}$, that are themselves bilinear combinations of the inputs, $\vec{m} = (\alpha \vec{a}) \odot (\beta \vec{b})$: for Eq. (1), by considering $\vec{m} = \left[\begin{array}{cc|cc|cc} m_{00} & m_{01} & m_{10} & m_{11} & m_{20} & m_{22} \end{array}\right]$, we have that $\mu = \left[\begin{array}{cc|cc|cc} 1 & -f & 0 & f & f & -f \\ -1 & 1 & 1 & 0 & -1 & f \end{array}\right] = \left[\begin{array}{c|c|c} M_0 & M_1 & M_2 \end{array}\right]$, for which $M_0^{-1} = (1-f)^{-1} \left[\begin{smallmatrix} 1 & f \\ 1 & 1 \end{smallmatrix}\right]$, $M_1^{-1} = \left[\begin{smallmatrix} 0 & 1 \\ f^{-1} & 0 \end{smallmatrix}\right]$, and $M_2^{-1} = (f^2 - f)^{-1} \left[\begin{smallmatrix} f & f \\ 1 & f \end{smallmatrix}\right]$.

From this, [9, Alg. 3] derives an accumulating in-place algorithm using 2×2 invertible blocks computing linear combinations like $\left[\begin{smallmatrix} c_i \\ c_j \end{smallmatrix}\right] \mathrel{+}= M \left[\begin{smallmatrix} \rho_0 \\ \rho_1 \end{smallmatrix}\right]$, via the equivalent algorithm: $M \left(\left(M^{-1} \left[\begin{smallmatrix} c_i \\ c_j \end{smallmatrix}\right]\right) \mathrel{+}= \left[\begin{smallmatrix} \rho_0 \\ \rho_1 \end{smallmatrix}\right]\right)$.

Again for Eq. (1), some successive operations are then simplified as follows:

(i) As $1 + f(1-f)^{-1} = (1-f)^{-1}$, then $M_0^{-1} \left[\begin{smallmatrix} c_0 \\ c_1 \end{smallmatrix}\right] = (1-f)^{-1} \left[\begin{smallmatrix} 1 & f \\ 1 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} c_0 \\ c_1 \end{smallmatrix}\right]$ can be sequentially computed via $c_1 \mathrel{+}= c_0$; $c_1 \mathrel{/}= (1-f)$; $c_0 \mathrel{+}= f c_1$.

(ii) As $M_2^{-1} \cdot M_0 = \left[\begin{smallmatrix} 0 & -1 \\ -f^{-1} & 0 \end{smallmatrix}\right] = - \left[\begin{smallmatrix} 1 & 0 \\ 0 & f^{-1} \end{smallmatrix}\right] \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, then computing M_2 right after M_0 allows to simplify the intermediate in-place operations into a variable swap, negations and multiplication by f^{-1}.

(iii) That negation can be delayed, since $(-M \cdot c) \mathrel{+}= \rho$ is equivalent to $-((M \cdot c) \mathrel{-}= \rho)$.

(iv) Similarly, $-M_1^{-1} \cdot M_2 = \left[\begin{smallmatrix} 1 & -f \\ -1 & 1 \end{smallmatrix}\right]$, thus computing M_1 after M_2 again simplifies to the sequential computation: $c_0 \mathrel{-}= c_1$; $c_1 \mathrel{-}= (1-f)$; $c_1 \mathrel{-}= f c_0$.

(v) Finally, $M_1 = \left[\begin{smallmatrix} f & 0 \\ 0 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and thus this combination is again just a swap of variables and a multiplication by f.

Overall, we obtain Alg. 3, thereafter.

Algorithm 3 In-place even degree f-convolution with accumulation

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of odd degree $n-1$; $f \in \mathbb{D} \setminus \{0, 1\}$.
Output: $C \mathrel{+}= AB \mod (X^n - f)$

if $n \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
  return the quadratic in-place polynomial multiplication.
else
  Let $t = n/2$; {n is even},
  Let $A = a_0 + X^t a_1$; $B = b_0 + X^t b_1$; $C = c_0 + X^t c_1$;
  {Iteration for m_0}
  $c_1 \mathrel{+}= c_0$;
  $c_1 \mathrel{/}= (1-f)$; {Simplification (i)}
  $c_0 \mathrel{+}= f \cdot c_1$;
  $\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$ {Via $\mathfrak{M}$, see Section 1.4, and denote $a_0 b_0 = \begin{bmatrix} m_{00} \\ m_{01} \end{bmatrix}$}
  $c_0 \mathrel{/}= f$; {Simplifications (ii) and (iii)}
  {Iteration for m_2}
  $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{-}= a_1 \cdot b_1$ {Via $\mathfrak{M}$, see Section 1.4, and denote $a_1 b_1 = \begin{bmatrix} m_{20} \\ m_{21} \end{bmatrix}$}
  $c_0 \mathrel{-}= c_1$; {this is $c_0/f + m_{00}/f - m_{01} + m_{20} - m_{21}$}
  $c_1 \mathrel{\star}= (1-f)$; {Simplification (iv)}
  $c_1 \mathrel{-}= f \cdot c_0$; {this is $c_1 - m_{00} + m_{01} - m_{20} + f m_{21}$}
  {Iteration for m_1}
  $a_0 \mathrel{+}= a_1$;
  $b_0 \mathrel{+}= b_1$;
  $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$ {Via $\mathfrak{M}$, see Section 1.4, and denote $(a_0 + a_1)(b_0 + b_1) = \begin{bmatrix} m_{10} \\ m_{11} \end{bmatrix}$}
  $b_0 \mathrel{-}= b_1$;
  $a_0 \mathrel{-}= a_1$;
  $c_0 \mathrel{\star}= f$; {Simplification (v)}
end if

Algorithm 3 works for odd degrees (even size), with both f and $(1-f)$ invertible. We need to design variant algorithms for the other cases.

Algorithm 4 deals with the case where $f = 1$. We present here only the version with 4 full products (as M_0 and M_2 are not invertible when $f = 1$).

Algorithm 4 In-place even degree 1-convolution with accumulation

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of odd degree $n-1$;
Output: $C \mathrel{+}= AB \mod (X^n - 1)$

1: **if** $n \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
2: **return** the quadratic in-place polynomial multiplication.
3: **else**
4: Let $t = n/2$; {n is even},
5: Let $A = a_0 + X^t a_1$; $B = b_0 + X^t b_1$; $C = c_0 + c_1 X^t$;
6: $\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$ {Via $\mathfrak{M}$, since $u + vY \mod Y^2 - 1 \equiv u + vY$}
7: $\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_1 \cdot b_1$ {Via $\mathfrak{M}$, since $Y^2 \equiv 1$}
8: $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{+}= a_0 \cdot b_1$ {Via $\mathfrak{M}$, since $(u + vY)Y \equiv v + uY$}
9: $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{+}= a_1 \cdot b_0$ {Via $\mathfrak{M}$, since $(u + vY)Y \equiv v + uY$}
10: **end if**

Algorithm 5 deals with the even degree case where $f \neq 0$. We here show only the version with 4 full products, but just for the sake of simplicity. The additional difficulty here is that the degrees of the lower and upper parts are different. Therefore, Alg. 5 ensures that there is always the correct space to accumulate.

Algorithm 5 In-place odd degree f-convolution with accumulation

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of even degree $n-1$; $f \in \mathbb{D}^*$.
Output: $C \mathrel{+}= AB \mod (X^n - f)$

1: **if** $n \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
2: **return** the quadratic in-place polynomial multiplication.
3: **end if**
4: Let $t = (n+1)/2$; {n is odd},
5: Let $A = a_0 + X^t a_1$; $B = b_0 + X^t b_1$; $C = c_0 + c_1 X^t$; {a_1,b_1,c_1 of degree $n-1-t = t-2$}
6: $\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$ {Via $\mathfrak{M}$, since degree $2t-2 < 2t-1 = n$}
7: $\begin{bmatrix} c_{1..(t-1)} \\ c_1 \end{bmatrix} \mathrel{+}= a_1 \cdot b_1$ {Via $\mathfrak{M}$, since $X^{2t} \equiv fX$, thus degree $(2t-4)+1 = n-2$, still $< n$}
8: $c_{0..(t-2)} \mathrel{/}= f$;
9: $\begin{bmatrix} c_{t..(2t-2)} \\ c_{0..(t-2)} \end{bmatrix} \mathrel{+}= a_0 \cdot b_1$ {Via $\mathfrak{M}$, since degree $2t-3 < 2t-1 = n$, with $(u + vX^{t-1})X^t \equiv (fv + uX^t)$}
10: $\begin{bmatrix} c_{t..(2t-2)} \\ c_{0..(t-2)} \end{bmatrix} \mathrel{+}= a_1 \cdot b_0$ {Via $\mathfrak{M}$, since degree $2t-3 < 2t-1 = n$, with $(u + vX^{t-1})X^t \equiv (fv + uX^t)$}
11: $c_{0..(t-2)} \mathrel{\star}= f$;

Finally, we also have to deal with the case $f = 0$, that is with the in-place short product $C \mathrel{+}= AB \mod X^n$. If the field cardinality is not 2, then a simple way is to compute two f-convolutions, with two different values for f, as shown in Remark 1.

Remark 1. *If $AB = QX^n + R$, of overall degree less than $2n-1$, then $AB = Q(X^n - 1) + (R + Q) = Q(X^n - g) + (R + gQ)$, for any g, and thus the respective remainders, by $X^n - 1$, and respectively by $X^n - g$, share the same quotient Q. Now for any $\lambda \notin \{0, 1\}$, taking $g = \lambda(\lambda - 1)^{-1}$, we obtain that $R = \lambda R + (1 - \lambda)R$*

together with $\lambda+g(1-\lambda)=0$. *Therefore* $C \mathrel{+}= AB \mod X^n$ *can be computed by* $C \mathrel{+}= \lambda AB \mod (X^n-1)$ *followed by* $C \mathrel{+}= (1-\lambda)AB \mod (X^n-g)$. *This is two calls among Algs. 3 to 5, with an overall complexity bound of* $\mathcal{O}(\mathfrak{M}(n))$.

There remains to compute a short product in the field with 2 elements. For this, we use a cutting in 3 blocks of size close to $n/3$ and [6, Eq. (24)] modulo 2, recalled in Eq. (2).

$$\begin{array}{l}
A(Y)=Y^2a_2+Ya_1+a_0; \quad B(Y)=Y^2b_2+Yb_1+b_0; \quad C(Y)=Y^2c_2+Yc_1+c_0;\\
\quad m_0=a_0\cdot b_0; \quad m_1=(a_0+a_1+a_2)\cdot(b_0+b_1+b_2); \quad m_2=a_2\cdot b_2;\\
\quad m_3=(a_0+a_2)\cdot(b_0+b_2); \quad m_4=(a_1+a_2)\cdot(b_1+b_2);\\
\quad m_5=(a_1+a_2)\cdot(b_0+b_1); \quad m_6=(a_0+a_2)\cdot(b_1+b_2);\\
\quad t_0=c_0+m_0; \quad t_1=c_1+m_1+m_2+m_3+m_4;\\
\quad t_2=m_1+m_3+m_4+m_5+m_6;\\
\textbf{then } C+AB \mod Y^3 \equiv Y^2t_2+Yt_1+t_0 \mod 2
\end{array} \tag{2}$$

From the 2×2 expansion of Eq. (2), again letting $m_i = m_{i0}+Ym_{i1}$, one obtains the 3×14 μ-matrix of Eq. (3):

$$\left[\begin{array}{cc|cc|cc|cc|cc|cc|cc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&0&1&0&1&0&0&0&0&0\\
0&0&1&1&0&1&1&1&1&1&1&hose0&1&0
\end{array}\right] \tag{3}$$

The five first pairs of columns can be dealt with the technique of [9]. The last two pairs are not full rank and thus cannot be handled like this (in other words, there are no places to put the high order terms of these two products, even though their are needed for the final result). They, however, just represent $(m_{50}+m_{60})$ mod $X^{n/3}$ and thus can be performed via 2 recursive calls of degree $n/3$. There remains to handle the cases where n is not a multiple of 3. For this, let $t=\lfloor n/3\rfloor$ and $Y=X^t$ and start by computing $C \mathrel{+}= AB$ mod X^{3t} via the algorithm sketched above. Now the remaining one or two coefficients, that of X^{3t} and X^{3t+1}, when applicable, are just accumulated directly, scalar multiplications by scalar multiplications. The whole process is shown in Alg. 6.

Overall, Alg. 6 uses 5 polynomial multiplications with polynomials of degree $n/3$, then 2 recursive calls of short products of degree $n/3$ and $\mathcal{O}(n)$ extra computations. The overall complexity is thus still $\mathcal{O}(\mathfrak{M}(n))$, as shown in Theorem 2.

Finally, Algs. 3 to 6 all together provide a complete solution for the fast in-place convolution with accumulation, as given in Alg. 7.

Theorem 2. *Using an in-place polynomial multiplication with complexity bounded by* $\mathfrak{M}(n)$, *Alg. 7 is correct, in-place and has complexity bounded by* $\mathcal{O}(\mathfrak{M}(n))$.

Proof. Correctness of Alg. 3 comes from that of Eq. (1). Correctness of Algs. 4 and 5 is direct from the degrees of the sub-polynomials, as given in their inside comments. Correctness of Alg. 6 comes from that of Eq. (2).

Also, all four algorithms are in-place as they use only in-place atomic operations or in-place polynomial multiplication.

Then, Algs. 3 to 5 use only a linear number of operations plus up to four calls to polynomial multiplications of half-degrees. Their complexity bound thus stems from the fact that $\mathfrak{M}(n)/n$ is increasing and that the number of operations is less than $\mathcal{O}(n)+4\mathfrak{M}(n/2)$.

Finally, Alg. 6 uses a linear number of atomic operations, 5 polynomial multiplications with polynomials of degree $n/3$ and only 2 recursive calls to third-degree polynomials. Thus its complexity bound satisfy $T(n)\leq 2T(n/3)+5\mathfrak{M}(n/3)+\mathcal{O}(n)$. Again, the non-decreasingness of $\mathfrak{M}(n)/n$, gives that $T(n)=\mathcal{O}(\mathfrak{M}(n))$. □

Algorithm 6 In-place short product (0-convolution) with accumulation

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of degree $< n$ in $\mathbb{D}[X]$;
Output: $C \mathrel{+}= AB \mod X^n$.

if $n \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
  return the quadratic in-place polynomial multiplication.
end if
if $\mathbb{D} \setminus \{0,1\} \neq \emptyset$ **then** {Use Remark 1}
  Choose $\lambda \notin \{0,1\}$ and let $g = \lambda(\lambda-1)^{-1}$;
  $C \mathrel{+}= \lambda AB \mod (X^n-1)$; {Algorithm 4 or Algorithm 5}
  return $C \mathrel{+}= (1-\lambda)AB \mod (X^n-g)$. {Algorithm 3 or Algorithm 5}
end if {The rest is thus just when $\mathbb{D} \simeq \mathbb{F}_2$}
Let $t = \lfloor n/3 \rfloor$;
Let $A \mod X^{3t} = a_0 + X^t a_1 + X^{2t} a_2$; $B \mod X^{3t} = b_0 + X^t b_1 + X^{2t} b_2$; $C \mod X^{3t} = c_0 + c_1 X^t + X^{2t} c_2$;
Let $m_i = m_{i0} + m_{i1} X^t$, $i = 0..6$, as in Eq. (2);

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \mathrel{+}= \left[\begin{array}{cc|cc|cc|cc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \cdot \left[\begin{array}{cc|cc|cc|cc|cc} m_{00} & m_{01} & m_{10} & m_{11} & m_{20} & m_{21} & m_{30} & m_{31} & m_{40} & m_{41} \end{array}\right]^{\intercal}$$ {[9, Alg. 3]}

$a_1 \mathrel{+}= a_2$; $b_0 \mathrel{+}= b_1$;
$c_2 \mathrel{+}= a_1 \cdot b_0 \mod X^t$ {Recursive call, for $c_2 \mathrel{+}= m_5 \mod X^t$}
$b_0 \mathrel{-}= b_1$; $a_1 \mathrel{-}= a_2$;
$a_0 \mathrel{+}= a_2$; $b_1 \mathrel{+}= b_2$;
$c_2 \mathrel{+}= a_0 \cdot b_1 \mod X^t$ {Recursive call, for $c_2 \mathrel{+}= m_6 \mod X^t$}
$b_1 \mathrel{-}= b_2$; $a_0 \mathrel{-}= a_2$;
if $n \geq 3t+1$ **then**
  for $i = 0$ **to** $3t$ **do**
    $c_{3t} \mathrel{+}= A_i \cdot B_{3t-i}$ {Scalar accumulations}
  end for
end if
if $n = 3t+2$ **then**
  for $i = 0$ **to** $3t+1$ **do**
    $c_{3t+1} \mathrel{+}= A_i \cdot B_{3t+1-i}$ {Scalar accumulations}
  end for
end if

Algorithm 7 In-place convolution with accumulation

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of degree $< n$; $f \in \mathbb{D}$.
Output: $C \mathrel{+}= AB \mod (X^n - f)$

if $f = 0$ **then**
  return Alg. 6.
else if n is odd **then**
  return Alg. 5.
else if $f = 1$ **then**
  return Alg. 4.
else
  return Alg. 3.
end if

3 Circulant and Toeplitz matrix operations with accumulation

Toeplitz matrix-vector multiplication can be reduced to circulant matrix-vector multiplication, via an embedding into a double-size circulant matrix. But this is not immediately in-place, since doubling the size requires a double space. We see in the following how we can instead double the operations while keeping the same dimension. We start by the usual definitions, also extending circulant matrices to f-circulant matrices, following, e.g., [18, Theorem 2.6.4].

3.1 Accumulating & in-place f-circulant matrices

Definition 3. *For $\vec{a} \in \mathbb{D}^m$, the* circulant matrix *represented by $\vec{a}$, $\mathscr{C}(\vec{a})$, is the $m \times m$ matrix (C_{ij}), such that $C_{1j} = a_j$ and the $(i+1)$-th row is the cyclic right shift by 1 of the i-th row.*

Definition 4. *For $f \in \mathbb{D}$ and $\vec{a} \in \mathbb{D}^m$, the (lower)* f-circulant matrix *represented by $\vec{a}$, $\mathscr{C}_{\mathrm{f}}(\vec{a})$, is the $m \times m$ matrix (Γ_{ij}), such that:*

$$\text{for } C = \mathscr{C}(\vec{a}), \begin{cases} \Gamma_{ij} = C_{ij} & \text{if } i \leq j, \\ \Gamma_{ij} = f \cdot C_{ij} & \text{otherwise.} \end{cases}$$

The algebra of f-Circulant matrices is in fact isomorphic to the algebra of polynomials modulo $X^n - f$ [18, Theorem 2.6.1]. This means that the product of an f-circulant matrix by a vector is obtained by the convolution of this vector by the vector representing the f-circulant matrix, and thus via Alg. 7.

In fact, it is well known that circulant matrices are diagonalized by a discrete Fourier transform, and hence can be manipulated via fast Fourier transforms as $\mathscr{C}_1(\vec{a}) = F_m^{-1}\mathrm{diag}(F_m\vec{a})F_m$, for F_m a DFT-matrix, see, e.g., [12, § 4.7.7]. This gives an alternative way to compute circulant matrix-vector multiplication in-place (using and restoring afterwards both the matrix and the vector). Indeed, the (even truncated) Fourier transform and its inverse can be computed in-place [19, 13, 8]. This gives us an in-place algorithm to compute the *accumulation* $\vec{c} \mathrel{+}= \mathscr{C}_1(\vec{a}) \cdot \vec{b}$ as:

1. $\vec{a} \leftarrow F_m\vec{a}$, $\vec{b} \leftarrow F_m\vec{b}$ and $\vec{c} \leftarrow F_m\vec{c}$;
2. $\vec{c} \mathrel{+}= \mathrm{diag}(\vec{a}) \cdot \vec{b}$;
3. $\vec{c} \leftarrow F_m^{-1}\vec{c}$; $\vec{b} \leftarrow F_m^{-1}\vec{b}$ and $\vec{a} \leftarrow F_m^{-1}\vec{a}$.

Now, this diagonalization allows to compute fast in-place accumulated circulant matrix-vector only when primitive roots exists, and only via this DFT algorithm. On the contrary, Alg. 7 has no restriction on f and is a reduction to any accumulated in-place polynomial multiplication (including DFT ones, as, e.g., [9, Alg. 7]).

3.2 Accumulating & in-place Toeplitz matrix operations

Definition 5. *For $\vec{a} \in \mathbb{D}^{2m-1}$, the (square)* Toeplitz matrix *represented by $\vec{a}$, $\mathcal{T}(\vec{a})$, is the $m \times m$ matrix (T_{ij}), such that $T_{ij} = a_{m+j-i}$. Similarly, for $\vec{a} \in \mathbb{D}^{m+n-1}$, we denote by $\mathcal{T}_{\mathrm{m,n}}(\vec{a})$ the $m \times n$ rectangular Toeplitz matrix defined by its first column, $\vec{a}_{1..m}$ bottom to top, and its first row, $\vec{a}_{m..(m+n-1)}$, left to right.*

Again, there is an isomorphism between rectangular Toeplitz matrices and the middle product of polynomials, see, e.g., [10, § 3.1]. We also immediately see that with these notations, we have for instance $\mathscr{C}_1(\vec{a}) = \mathscr{C}(\vec{a})$, $\mathscr{C}_0(\vec{a}) = \frac{1}{2}(\mathscr{C}_1(\vec{a}) + \mathscr{C}_{\text{-}1}(\vec{a}))$, or also $\mathscr{C}_{\mathrm{f}}(\vec{a}) = \mathcal{T}([f \cdot \vec{a}_{2..m}, \vec{a}])$, where $[\vec{u}, \vec{v}]$ denotes the vector obtained by concatenation of $\vec{u}$ and $\vec{v}$.

Fast algorithms for f-circulant matrices then allows us to build algorithms, by reduction, for accumulation with triangular and square Toeplitz matrices first, as sums of f-circulant in Algs. 8 and 9, and then for any Toeplitz matrix, again as sums of triangular Toeplitz matrices in Alg. 10.

Algorithm 8 In-place accumulating Upp. Triang. Toeplitz m-v. mult.

Input: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{D}^m$.
Output: $\vec{c} \mathrel{+}= \mathcal{T}([\vec{0}, \vec{a}]) \cdot \vec{b}$.
1: **return** $\vec{c} \mathrel{+}= \mathscr{C}_0(\vec{a}) \cdot \vec{b}$. {Algorithm 7}

Remark 6. *We here present f-circulant matrices where the coefficient acts on the lower left part of the matrix (excluding the diagonal). In the same manner, one can design an in-place fast algorithm for the other type of f-circulant matrices, where the coefficient would act on the upper right part of the matrix (excluding the diagonal). One could thus derive the transposed version of Alg. 8, with a single call to (the transposed version of) Alg. 7. Now it is also possible to use the same algorithm but on the transposed matrix and on reversed vectors: letting $\overleftarrow{b}$ be the reversed vector of $\vec{b}$, one can see by inspection that:*

$$\mathscr{C}_{\mathrm{f}}(\vec{a})^{\mathsf{T}} \cdot \overleftarrow{b} \text{ is the reverse of } \mathscr{C}_{\mathrm{f}}(\vec{a}) \cdot \vec{b} \tag{4}$$

Algorithm 9 In-place accumulating square Toeplitz m-v. mult.

Input: $\vec{a_1} \in \mathbb{D}^m$, $\vec{a_2}, \vec{b}, \vec{c} \in \mathbb{D}^{m+1}$,
Output: $\vec{c} \mathrel{+}= \mathcal{T}([\vec{a_1}, \vec{a_2}]) \cdot \vec{b}$.
1: $\vec{c} \mathrel{+}= \mathscr{C}_0(\vec{a_2}) \cdot \vec{b}$; {Algorithm 7}
2: Let $\vec{b_1} = \vec{b}_{1..m}$ and $\vec{c_2} = \vec{c}_{2..m+1}$;
3: In-place reverse $\vec{a_1}$ (resp. $\vec{b_1}$, $\vec{c_2}$) into $\overleftarrow{a_1}$ (resp. $\overleftarrow{b_1}$, $\overleftarrow{c_2}$)
4: $\overleftarrow{c_2} \mathrel{+}= \mathscr{C}_0(\overleftarrow{a_1}) \cdot \overleftarrow{b_1}$; {Equation (4) and Algorithm 7}
5: In-place reverse $\overleftarrow{a_1}$ (resp. $\overleftarrow{b_1}$, $\overleftarrow{c_2}$) into $\vec{a_1}$ (resp. $\vec{b_1}$, $\vec{c_2}$)
6: **return** $\vec{c}$.

Lemma 7. *Algorithms 8 and 9 are correct and have complexity bounded by $\mathcal{O}(\mathfrak{M}(n))$.*

Proof. The complexity bound comes from that of Alg. 7. Correctness is obtained directly looking at the values of the matrices:

- $\mathcal{T}([\vec{0}, \vec{a}]) = \mathscr{C}_0(\vec{a})$.
- $\mathcal{T}([\vec{a_1}, \vec{a_2}]) = \mathscr{C}_0(\vec{a_2}) + \begin{bmatrix} \vec{0}^{\mathsf{T}} & 0 \\ \mathscr{C}_0(\overleftarrow{a_1})^{\mathsf{T}} & \vec{0} \end{bmatrix}$.

Then for Alg. 9, Remark 6 and Eq. (4) shows that it is possible to compute $\mathscr{C}_0(\overleftarrow{a_1})^{\mathsf{T}} \cdot \vec{b_1}$ via the reverse of $\mathscr{C}_0(\overleftarrow{a_1}) \cdot \overleftarrow{b_1}$. □

From this, we give in Alg. 10 an in-place rectangular Toeplitz matrix-vector multiplication.

Proposition 8. *Algorithm 10 is correct and requires less than $\mathcal{O}(\mathfrak{M}(\max\{m, n\}))$ operations.*

Proof. If $m>n$, then there are first $\lfloor m/n \rfloor$ square $n \times n$ calls to Alg. 9. This is bounded by $\mathcal{O}((m/n)\mathfrak{M}(n)) \leq \mathcal{O}(\mathfrak{M}(m))$ operations. The remaining recursive call is then negligible. This is similar when $m<n$. □

These in-place accumulated Toeplitz matrix-vector multiplications, in turns, allows us to obtain an over-place triangular Toeplitz multiplication or system solve, given in Algs. 11 and 12.

Proposition 9. *Algorithm 11 is correct and requires less than $\mathcal{O}(\mathfrak{M}(m)\log(m))$ operations if $\mathfrak{M}(m)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(m))$ otherwise.*

Algorithm 10 In-place accumulating rectangular Toeplitz m-v. mult.

Input: $\vec{a} \in \mathbb{D}^{m+n-1}$, $\vec{b} \in \mathbb{D}^n$, $\vec{c} \in \mathbb{D}^m$,
Output: $\vec{c}$ += $\mathcal{T}_{\text{m,n}}(\vec{a}) \cdot \vec{b}$.

1: **if** $m = n$ **then**
2: **return** $\vec{c}$ += $\mathcal{T}(\vec{a}) \cdot \vec{b}$. {Algorithm 9}
3: **end if**
4: **if** $m > n$ **then**
5: Let $c_1 = \vec{c}_{1..n}$ and $c_2 = \vec{c}_{(n+1)..m}$;
6: c_1 += $\mathcal{T}\left(\vec{a}_{(m-n+1)..(m+n-1)}\right) \cdot \vec{b}$; {Algorithm 9}
7: c_2 += $\mathcal{T}_{\text{m-n,n}}\left(\vec{a}_{1..(m-1)}\right) \cdot \vec{b}$; {Recursive call}
8: **else**
9: Let $b_1 = \vec{b}_{1..m}$ and $b_2 = \vec{b}_{(m+1)..n}$;
10: c += $\mathcal{T}\left(\vec{a}_{1..(2m-1)}\right) \cdot b_1$; {Algorithm 9}
11: c += $\mathcal{T}_{\text{m,n-m}}\left(\vec{a}_{(m+1)..(m+n-1)}\right) \cdot b_2$; {Recursive call}
12: **end if**
13: **return** $\vec{c}$.

Algorithm 11 Over-place triang. Toeplitz m-v. mult.

Input: $\vec{a}, \vec{b} \in \mathbb{D}^m$, s.t. $a_1 \in \mathbb{D}^*$.
Output: $\vec{b} \leftarrow \mathcal{T}([\vec{a}, \vec{0}]) \cdot \vec{b}$.

1: **if** $m \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
2: **return** the quadratic in-place triangular m-v. mult. {Algorithm 1}
3: **end if**
4: Let $k = \lceil m/2 \rceil$, $b_1 = \vec{b}_{1..k}$ and $b_2 = \vec{b}_{(k+1)..m}$;
5: $b_2 \leftarrow \mathcal{T}([\vec{a}_{(k+1)..m}, \vec{0}]) \cdot b_2$; {Recursive call}
6: b_2 += $\mathcal{T}_{\text{m-k,k}}([a_1, \ldots, a_{m-1}]) \cdot b_1$; {Algorithm 10}
7: $b_1 \leftarrow \mathcal{T}([a_{(m-k+1)..m}, \vec{0}]) \cdot b_1$; {Recursive call}
8: **return** $\vec{b}$.

Proof. For the correctness, let $T = \mathcal{T}([\vec{a}, \vec{0}])$ and consider it as blocks $T_1 = \mathcal{T}([a_{(m-k+1)..m}, \vec{0}])$, $T_2 = \mathcal{T}([\vec{a}_{(k+1)..m}, \vec{0}])$ and $G = \mathcal{T}_{\text{m-k,k}}([a_1, \ldots, a_{m-1}])$. Then $T = \left[\begin{smallmatrix} T_1 & 0 \\ G & T_2 \end{smallmatrix}\right]$. Thus $T\vec{b} = \left[\begin{smallmatrix} T_1 b_1 \\ Gb_1 + T_2 b_2 \end{smallmatrix}\right]$. Let $\bar{b}_1 = T_1 b_1$, $\hat{b}_2 = T_2 b_2$ and $\bar{b}_2 = Gb_1 + T_2 b_2$. Then $\bar{b}_2 = \hat{b}_2 + Gb_1$ and this shows that the algorithm is correct. Now for the complexity bound, the cost function is $T(m) \leq 2T(m/2) + \mathcal{O}(\mathfrak{M}(m))$. This is $\leq \mathcal{O}(\mathfrak{M}(m)\log(m))$ if $\mathfrak{M}(m)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(m))$ otherwise. □

Proposition 10. *Algorithm 12 is correct and requires less than $\mathcal{O}(\mathfrak{M}(m)\log(m))$ operations if $\mathfrak{M}(m)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(m))$ otherwise.*

Proof. For the correctness, let $T = \mathcal{T}([\vec{0}, \vec{a}])$ and consider it as blocks $T_1 = \mathcal{T}([\vec{0}, \vec{a}_{1..k}])$, $T_2 = \mathcal{T}([\vec{0}, \vec{a}_{(k+1)..m}])$ and $G = \mathcal{T}_{\text{k,m-k}}(\vec{a}_{2..m})$. Then $T = \left[\begin{smallmatrix} T_1 & G \\ 0 & T_2 \end{smallmatrix}\right]$. Now define H, s.t. $T^{-1} = \left[\begin{smallmatrix} T_1^{-1} & H \\ 0 & T_2^{-1} \end{smallmatrix}\right]$. Then H satisfies $T_1^{-1}G + HT_2 = 0$. Also, we have $T^{-1}\vec{b} = \left[\begin{smallmatrix} T_1^{-1}b_1 + Hb_2 \\ T_2^{-1}b_2 \end{smallmatrix}\right]$. Let $\bar{b}_2 = T_2^{-1}b_2$ and $\bar{b}_1 = T_1^{-1}b_1 + Hb_2$. Then $\bar{b}_1 = T_1^{-1}b_1 + HT_2\bar{b}_2 = T_1^{-1}b_1 - T_1^{-1}G\bar{b}_2 = T_1^{-1}\left(b_1 - G\bar{b}_2\right)$ and this shows that the algorithm is correct. Now for the complexity bound, the cost function is $T(m) \leq 2T(m/2) + \mathcal{O}(\mathfrak{M}(m))$. Again, this is $\leq \mathcal{O}(\mathfrak{M}(m)\log(m))$ if $\mathfrak{M}(m)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(m))$ otherwise. □

We now have over-place Toeplitz methods. Next, we reduce the extra space for polynomial remaindering. Eventually, we combine these two techniques to obtain in-place polynomial remaindering.

Algorithm 12 Over-place triang. Toeplitz system solve

Input: $\vec{a}, \vec{b} \in \mathbb{D}^m$, s.t. $a_1 \in \mathbb{D}^*$.
Output: $\vec{b} \leftarrow \mathcal{T}([\vec{0}, \vec{a}])^{-1} \cdot \vec{b}$.

1: **if** $m \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
2: **return** the quadratic in-place triangular system solve.
3: **end if**
4: Let $k = \lceil m/2 \rceil$, $b_1 = \vec{b}_{1..k}$ and $b_2 = \vec{b}_{(k+1)..m}$;
5: $b_2 \leftarrow \mathcal{T}([\vec{0}, \vec{a}_{(k+1)..m}])^{-1} \cdot b_2$; {Recursive call}
6: $b_1 \mathrel{-}= \mathcal{T}_{\text{k,m-k}}(\vec{a}_{2..m}) \cdot b_2$; {Algorithm 10}
7: $b_1 \leftarrow \mathcal{T}([\vec{0}, \vec{a}_{1..k}])^{-1} \cdot b_1$; {Recursive call}
8: **return** $\vec{b}$.

4 In-place modular remainder

We consider the fast in-place (resp. over-place) computation of the Euclidean polynomial modular remainder $R(X) \equiv A(X) \mod B(X)$ (resp. $A(X) \equiv A(X) \mod B(X)$) with A and B of respective degrees n and $m \leq n$. Standard algorithms for the remainder require $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ arithmetic operations and, apart from that of A and B, at least $\mathcal{O}(n-m)$ extra memory [11]. This extra space is notably usually used to store the whole quotient $Q(X)$ such that $A = BQ + R$ with $\deg R < \deg B$.

We first show how to avoid the storage of the whole of this quotient, and propose an algorithm still using $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ arithmetic operations but only $\mathcal{O}(m)$ extra space (when the divisor B is sparse with a constant number of non-zero terms, the arithmetic complexity bound reduces to $\mathcal{O}(n)$).

Finally, we combine this with the techniques of Sections 2 and 3 and use the input space of A or B for intermediate computations in order to propose in-place and over-place algorithms for the modular remainder using at most $\mathcal{O}(\mathfrak{M}(m)\log(m))$ arithmetic operations if $\mathfrak{M}(m)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(m))$ otherwise.

4.1 Successively over-writing the quotients

With two polynomials A and B of respective degrees N and M, the computation of the Euclidean division remainder R of degree strictly less than M such that $A = BQ + R$ with quotient Q, can be rewritten as $R \equiv A - BQ \mod X^M$. This is therefore enough to compute the quotient only up to the degree $M-1$: let $A_M \equiv A \mod X^M$ and $Q_M \equiv Q \mod X^M$, then $R \equiv A_M - BQ_M \mod X^M$.

This observation is the ingredient that allows to compute the remainder using an extra space only of the order of the degree of the divisor B. One can also see this as the long division algorithm applied to blocks of dimension M.

Let us write the Euclidean equation $A = BQ + R$ in a Toeplitz matrix form. In Eqs. (5) and (6), we view the polynomials A, Q and R as vectors $[a_0, \ldots, a_N]$, $[q_0, \ldots, q_{N-M}]$, $[r_0, \ldots, r_{M-1}, 0, \ldots, 0]$ and then B as a Toeplitz matrix $B = \mathcal{T}([0, \ldots, 0, b_M, \ldots, b_0, 0, \ldots, 0])$. Then, focusing on the last $N - M + 1$ rows of Eq. (5) we obtain directly the upper triangular $(N - M + 1) \times (N - M + 1)$ Toeplitz system of equations whose solution is only the quotient, as shown in Eq. (6).

$$\begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} b_0 & & 0 \\ \vdots & \ddots & \\ b_M & \ddots & b_0 \\ & \ddots & \vdots \\ 0 & & b_M \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ \vdots \\ \\ \vdots \\ q_{N-M} \end{bmatrix} + \begin{bmatrix} r_0 \\ \vdots \\ r_{M-1} \\ \\ 0 \\ \\ \end{bmatrix}. \tag{5}$$

$$\begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} b_M & \dots & b_0 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_0 \\ & & & \ddots & \vdots \\ & 0 & & & b_M \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ \vdots \\ \vdots \\ q_{N-M} \end{bmatrix}. \tag{6}$$

We now let $n = N - M + 1$ and suppose that B is really of degree M, that is, its leading coefficients b_M is invertible in the coefficient domain. For the sake of simplicity we also assume that n is a multiple of M (otherwise, for now, just complete the polynomial A with virtual leading zero coefficients up to the next multiple of M) and let $\mu = \frac{n}{M}$. We then denote the $M \times M$ blocks of the Toeplitz matrix in Eq. (6) by: $T = \mathcal{T}([\vec{0}_{M-1}, b_{M..1}])$ and $G = \mathcal{T}([b_{M-1..0}, \vec{0}_{M-1}])$, that is:

$$T = \begin{bmatrix} b_M & \dots & b_1 \\ & \ddots & \vdots \\ & & b_M \end{bmatrix} \text{ and } G = \begin{bmatrix} b_0 & & \\ \vdots & \ddots & \\ b_{M-1} & \dots & b_0 \end{bmatrix}. \tag{7}$$

This in turns gives a way to access only the first coefficients of Q in an upper triangular Toeplitz system, with a 2-block band structure:

$$\begin{bmatrix} q_0 \\ \vdots \\ \vdots \\ q_{M-1} \end{bmatrix} = \left[\begin{array}{c:c} I_M & 0 \end{array}\right] \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} \tag{8}$$

where $\left[\begin{array}{c:c} I_M & 0 \end{array}\right]$ is the concatenation of the $M \times M$ identity matrix and the $(N-M) \times (N-M)$ zero matrix.

Finally, recovering the remainder from the first M rows of Eq. (5) equations is just like multiplying the quotient by G and thus $R = A - BQ \mod X^M$ can be written as:

$$R = \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_{M-1} \end{bmatrix} - \left[\begin{array}{c:c} G & 0 \end{array}\right] \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} \tag{9}$$

In fact, as we need only the first M coefficient of the Toeplitz system we just need the first block-row of the (upper triangular) inverse of the upper triangular 2-band Toeplitz matrix.

Now this first block-row, of the inverse, of an upper triangular Toeplitz matrix U, is given by a direct formula, obtained from either of the equations $U \cdot U^{-1} = I$ or $U^{-1} \cdot U = I$ (see, e.g., [7, Eq.(1)] for the scalar case). If we denote by H_i the blocks of that row, we have that:

$$\begin{cases} H_1 = T^{-1} \\ H_{i-1}G + H_i T = 0, & i = 2..\mu \\ TH_i + GH_{i-1} = 0, & i = 2..\mu \end{cases} \tag{10}$$

Solving Eq. (10), we just get that $H_i = T^{-1}(-GT^{-1})^i = (-GT^{-1})^i T^{-1}$.

We have shown:

Lemma 11.

$$\begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} = T^{-1}. \begin{bmatrix} I & -GT^{-1} & \dots & (-GT^{-1})^{\mu-1} \\ & \ddots & \ddots & \vdots \\ & & \ddots & -GT^{-1} \\ 0 & & & I \end{bmatrix}$$

Now denote by $[\vec{a_0}, \vec{a_1}, \dots, \vec{a_\mu}]$ the decomposition into blocks of dimension M of $[a_0, \dots, a_N]$. Combining Eq. (9) and Lemma 11, we obtain now that:

$$R = \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_{M-1} \end{bmatrix} - GT^{-1} \left(\sum_{i=1}^{\mu} \left(-GT^{-1}\right)^{i-1} \vec{a_i} \right) = \sum_{i=0}^{\mu} \left(-GT^{-1}\right)^{i} \vec{a_i} \tag{11}$$

From Eq. (11) we thus can immediately deduce the following Alg. 13 that uses only $\mathcal{O}(M)$ extra memory space in a Horner-like fashion of the polynomial in $(-GT^{-1})$ of Eq. (11). Note that this algorithm does not modify its input along its course: both $A(X)$ and $B(X)$ are for now read-only (in particular virtually padding $\vec{a}$ with virtual zeroes, Line 3, is therefore not an issue).

Algorithm 13 Overwritten-quotient Euclidean remainder

Input: $A(X)$, $B(X)$ in $\mathbb{D}[X]$ of respective degrees N and M.
Read-only: $A(X), B(X)$.
Output: $R(X) \equiv A(X) \mod B(X)$ of degree at most $M-1$.

1: **if** $M > N$ **then return** A.
2: Let $n = N - M + 1$, $\mu = \left\lceil \frac{n}{M} \right\rceil$;
3: Let $[\vec{a_0}, \dots, \vec{a_\mu}] = [a_0, \dots, a_N, \vec{0}]$; {Blocks of dimension M}
4: Let $T = \mathcal{T}([\vec{0}_{M-1}, b_M, \dots, b_1])$, $G = \mathcal{T}([b_{M-1}, \dots, b_0, \vec{0}_{M-1}])$;
5: $\vec{r} = \vec{a}_\mu$; {$\vec{r}$ in-place of the result}
6: **for** $i = \mu - 1$ **down-to** 0 **do**
7: $\quad \vec{t} = T^{-1} \cdot \vec{r}$; {Triang. Toeplitz solve}
8: $\quad \vec{r} = (-G) \cdot \vec{t}$; {Triang. Toeplitz m-v. mult.}
9: $\quad \vec{r}$ += $\vec{a_i}$;
10: **end for**
11: **return** $R = \sum_{i=0}^{M-1} r_i X^i$.

Theorem 12. *Algorithm 13 is correct and requires $\mathcal{O}\left(\frac{N}{M}\mathfrak{M}(M)\right)$ arithmetic operations and $\mathcal{O}(M)$ extra memory space. If the polynomial B is sparse with a constant number of non-zero coefficients, the arithmetic complexity is reduced to $\mathcal{O}(N)$.*

Proof. Correctness stems directly from Lemma 11 and Eq. (11). For the complexity bounds, we use the results of Section 1.3. For each block, the triangular Toeplitz system solve and the Toeplitz m-v. mult. require respectively $\lambda_s \mathfrak{M}(m)$ and $\lambda_t \mathfrak{M}(m)$ operations and, sequentially, $\max\{s;t\}M$ extra space. Apart from this space, we only need one extra vector, $\vec{t}$, to store intermediate results. Overall we thus perform $\mu((\lambda_s + \lambda_t)\mathfrak{M}(M) + M)$ operations. With $\mu = n/M$ and $n = N - M + 1$, this is $\left\lceil \frac{N-M+1}{M} \right\rceil ((\lambda_s + \lambda_t)\mathfrak{M}(M) + M) = \mathcal{O}(\frac{N}{M}\mathfrak{M}(M))$ operations, using $(1 + \max\{s;t\})M$ extra space.

Now if B is sparse with a constant number of non-zero elements, each triangular Toeplitz system solve and Toeplitz matrix-vector multiplication can be performed with only $\mathcal{O}(M)$ operations with the same extra memory space. Thus the overall arithmetic bound becomes $\mathcal{O}(\mu M) = \mathcal{O}(N)$. □

Remark 13. *Algorithm 13 is in fact just the long division polynomial algorithm applied to sub-blocks of the polynomial of size M:*

- *Line 7, $\vec{t} = T^{-1} \cdot \vec{r}$, corresponds to computing the quotient of the current leading coefficients, of the dividend, by B;*
- *Line 8, $\vec{r} = (-G) \cdot \vec{t}$, corresponds to recovering the lower part of the multiplication of that current quotient by B;*
- *Line 9, $\vec{r}$ += $\vec{a_i}$, updates the next M coefficients of the current dividend (the leading ones being zero by construction of the current quotient).*

This in-place long division by block is also sketched for instance in the proof of [11, Lemma 2.1]. The latter Lemma gives about $3\lambda\mathfrak{M}(N)$ operations and $(2+s)M$ extra space. In fact a refined analysis should also give the same (better) complexity as that of Theorem 12, that is less than $2\lambda\frac{N}{M}\mathfrak{M}(M)$ operations and $(1+s)M$ extra space.

4.2 In-place remainders via Toeplitz techniques

We now derive algorithms that use only $\mathcal{O}(1)$ extra memory space in the in-place model of Section 1.1: modifying the inputs is possible if and only if all inputs are restored to their initial state after the completion of the algorithm. This allows us to store some intermediate results, over-writing the input, provided that we can afterwards recompute the initial inputs in their entirety. Further, this enables recursive calls, as intermediate values are used but restored along the recursive descent. The general idea is then to combine Sections 3 and 4.1.

in fact, in the following, we present three in-place variants of polynomial remaindering:

- IPER: for $R(X) \equiv A(X) \mod B(X)$, creates a small remainder from a large polynomial using only the output space and that of the modulus B (i.e. A is read-only and B is restored to its initial state after completion);
- OPER: for $A(X) \equiv A(X) \mod B(X)$, updates a large polynomial into a small remainder (with B restored to its initial state after completion);
- APER: for $R(X)$ += $A(X) \mod B(X)$, accumulates the remainder of a large polynomial into a small one (with both A and B restored to their initial states after completion).

We can now present in Alg. 14, a fully in-place remainder where only $B(X)$ is modified but restored: this variant replaces only Lines 7 to 9 of Alg. 13 by their over-place variants, Algs. 11 and 12, that modify and restore some parts of B.

From Alg. 14 we also obtain, Alg. 15 which directly updates the dividend over-place, while also remaining fully in-place. In Alg. 14, A is read-only and can thus be virtually padded with zeroes, when $N - M + 1$ is not a multiple of M. This is not the case anymore in Alg. 15. This is the reason why we need to now take some special care of the last $s = (N+1) \mod M$ coefficients.

Finally, we also obtain the variant Alg. 16, where the remainder is accumulated in-place. As in Alg. 15 the dividend is updated over-place, but in fact enough information is preserved so as to put it back into its initial state after the accumulation of the actual remainder.

Theorem 14. *Algorithms 14 to 16 are correct, in-place and require less than $\mathcal{O}\left(\frac{N}{M}\mathfrak{M}(M)\log(M)\right)$ operations if $\mathfrak{M}(M)$ is quasi-linear, and $\mathcal{O}\left(\frac{N}{M}\mathfrak{M}(M)\right)$ otherwise.*

Proof. Algorithm 14 calls $\mu = \left\lceil \frac{N-M+1}{M} \right\rceil = \mathcal{O}(N/M)$ times Algs. 11 and 12, each call requiring less than $\mathcal{O}(\mathfrak{M}(M)\log(M))$ operations by Props. 9 and 10 if $\mathfrak{M}(M)$ is quasi-linear, and $\mathcal{O}(\mathfrak{M}(M))$ otherwise. This suffices to prove the Theorem's claims for Alg. 14. These costs also dominates the costs of Algs. 9 and 10 by Lemma 7 and Prop. 8, and therefore that of Alg. 15: its Lines 5 to 8 are the over-place first iteration of Alg. 14, for $s < M$. Finally, Alg. 16 is also similar, using the same sub-algorithms, and only roughly doubling the overall cost (by symmetry before and after the accumulation). □

Algorithm 14 IPER (R,A,B): In-place Polynomial Euclidean Remainder

Input: $A(X)$, $B(X)$ in $\mathbb{D}[X]$ of respective degrees N and M.
Read-only: $A(X)$.
Output: $R(X) \equiv A(X) \mod B(X)$ of degree at most $M-1$.

1: **if** $M > N$ **then return** A.
2: Let n, μ, $[\vec{a_0}, \ldots, \vec{a_\mu}]$, T, G as in Alg. 13;
3: $\vec{r} = \vec{a_\mu}$; {$\vec{r}$ in-place of the result}
4: **for** $i = \mu - 1$ **down-to** 0 **do**
5: $\quad \vec{r} \leftarrow T^{-1} \cdot \vec{r}$; {Algorithm 12}
6: $\quad \vec{r} \leftarrow (-G) \cdot \vec{r}$; {Algorithm 11}
7: $\quad \vec{r}$ += $\vec{a_i}$;
8: **end for**
9: **return** $R = \sum_{i=0}^{M-1} r_i X^i$.

Algorithm 15 OPER (A,B): Over-place Polynomial Euclidean Remainder

Input: $A(X)$, $B(X)$ in $\mathbb{D}[X]$ of respective degrees N and M.
Output: $A(X) \equiv A(X) \mod B(X)$ of degree at most $M-1$.

1: **if** $M > N$ **then return** A.
2: Let n, μ, T, G as in Alg. 14 and $s = (N+1) \mod M$;
3: Let $[\vec{a_0}, \ldots, \vec{a}_{\mu-1}] = [a_0, \ldots, a_{N-s}]$ and $\vec{a_\mu} = \vec{a}_{(N-s+1)..N}$;
4: **if** $s \neq 0$ **then**
5: $\quad$ Let $T_1 = \mathcal{T}([\vec{0}_{s-1}, b_M, \ldots, b_{M-s+1}])$; {$s \times s$ upper left of T}
6: $\quad \vec{a_\mu} \leftarrow T_1^{-1} \cdot \vec{a_\mu}$; {Algorithm 12}
7: $\quad$ Let $G_1 = \mathcal{T}_{\mathrm{M,s}}([b_{M-1}, \ldots, b_0, \vec{0}_{s-1}])$; {Left s columns of G}
8: $\quad \vec{a}_{\mu-1}$ += $(-G_1) \cdot \vec{a_\mu}$; {Algorithm 10}
9: **end if**
10: **for** $i = \mu - 1$ **down-to** 1 **do**
11: $\quad \vec{a_i} \leftarrow T^{-1} \cdot \vec{a_i}$; {Algorithm 12}
12: $\quad \vec{a}_{i-1}$ += $(-G) \cdot \vec{a_i}$; {Algorithm 9}
13: **end for**
14: **return** $A = \vec{a_0}$.

5 Conclusion

We have presented novel reductions of accumulated in-place f-circulant and Toeplitz matrix-vector multiplications to in-place polynomial multiplication. This allows us to derive novel algorithms for accumulated or over-place Toeplitz multiplication or Toeplitz system solving. We also present algorithms that reduce the extra storage required to compute the remainder only when dividing polynomials. Eventually, we combine these techniques to propose the first in-place, over-place and accumulating algorithms computing only the remainder of the polynomial Euclidean division.

References

[1] Dario Bini and Victor Y. Pan. Fast parallel polynomial division via reduction to triangular Toeplitz matrix inversion and to polynomial inversion modulo a power. *Inf. Process. Lett.*, 21(2):79–81, 1985. doi:10.1016/0020-0190(85)90037-7.

[2] Dario Bini and Victor Y. Pan. *Polynomial and matrix computations, 1st Edition*, volume 12 of *Progress in theoretical computer science*. Birkhäuser, 1994. doi:10.1007/978-1-4612-0265-3.

Algorithm 16 APER (R,A,B): Accumulated in-place Polynomial Euclidean Remainder

Input: $R(X)$, $A(X)$, $B(X)$ in $\mathbb{D}[X]$ of respective degrees $M-1$, N and M.
Output: $R(X)$ += $A(X) \mod B(X)$ of degree at most $M-1$.

1: **if** $M > N$ **then return** R += A.
2: Let n, μ, s, T, T_1, G, G_1, $[\vec{a_0}, \ldots, \vec{a_\mu}]$ as in Alg. 15, and $\vec{r} = [r_0, \ldots, r_{M-1}]$;
3: **if** $s \neq 0$ **then**
4: $\quad \vec{a_\mu} \leftarrow T_1^{-1} \cdot \vec{a_\mu}$; {Algorithm 12}
5: $\quad \vec{a}_{\mu-1}$ −= $G_1 \cdot \vec{a_\mu}$; {Algorithm 10}
6: **end if**
7: **for** $i = \mu - 1$ **down-to** 1 **do**
8: $\quad \vec{a_i} \leftarrow T^{-1} \cdot \vec{a_i}$; {Algorithm 12}
9: $\quad \vec{a}_{i-1}$ −= $G \cdot \vec{a_i}$; {Algorithm 9}
10: **end for**
11: $\vec{r}$ += $\vec{a_0}$;
12: **for** $i = 1$ **to** $\mu - 1$ **do**
13: $\quad \vec{a}_{i-1}$ += $G \cdot \vec{a_i}$; {Undo Line 9 via Algorithm 9}
14: $\quad \vec{a_i} \leftarrow T \cdot \vec{a_i}$; {Undo Line 8 via Algorithm 11}
15: **end for**
16: **if** $s \neq 0$ **then**
17: $\quad \vec{a}_{\mu-1}$ += $G_1 \cdot \vec{a_\mu}$; {Undo Line 5 via Algorithm 10}
18: $\quad \vec{a_\mu} \leftarrow T_1 \cdot \vec{a_\mu}$; {Undo Line 4 via Algorithm 11}
19: **end if**
20: **return** $\vec{r}$.

[3] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In Rafael Sendra, editor, *ISSAC'2003, Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation, Philadelphia,Pennsylvania*, pages 37–44, New York, August 2003. ACM Press. doi:10.1145/860854.860870.

[4] Brice Boyer, Jean-Guillaume Dumas, Clément Pernet, and Wei Zhou. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. In May [16], pages 135–143. doi:10.1145/1576702.1576713.

[5] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991. doi:10.1007/BF01178683.

[6] Murat Cenk and M. Anwar Hasan. Some new results on binary polynomial multiplication. *Journal of Cryptographic Engineering*, 5(4):289–303, 2015. doi:10.1007/s13389-015-0101-6.

[7] Daniel Commenges and Michel Monsion. Fast inversion of triangular Toeplitz matrices. *IEEE Transactions on Automatic Control*, 29(3):250–251, 1984. doi:10.1109/TAC.1984.1103499.

[8] Nicholas Coxon. An in-place truncated Fourier transform. *Journal of Symbolic Computation*, 110:66–80, 2022. doi:https://doi.org/10.1016/j.jsc.2021.10.002.

[9] Jean-Guillaume Dumas and Bruno Grenet. In-place accumulation of fast multiplication formulae. Technical report, IMAG-hal-04167499, September 2023. URL: https://hal.science/hal-04167499.

[10] Pascal Giorgi, Bruno Grenet, and Daniel S. Roche. Generic reductions for in-place polynomial multiplication. In James H. Davenport, Dongming Wang, Manuel Kauers, and Russell J. Bradford, editors, *ISSAC'2019, Proceedings of the 2019 International Symposium on Symbolic and Algebraic Computation, Beijing, China*, pages 187–194, New York, July 2019. ACM Press. doi:10.1145/3326229.3326249.

[11] Pascal Giorgi, Bruno Grenet, and Daniel S. Roche. Fast in-place algorithms for polynomial operations: division, evaluation, interpolation. In Ioannis Z. Emiris, Lihong Zhi, and Anton Leykin, editors, *ISSAC'2020, Proceedings of the 2020 International Symposium on Symbolic and Algebraic Computation, Kalamata, Greece*, pages 210–217, New York, July 2020. ACM Press. doi:10.1145/3373207.3404061.

[12] Gene H. Golub and Charles F. van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996. doi:10.56021/9781421407944.

[13] David Harvey and Daniel S. Roche. An in-place truncated Fourier transform and applications to polynomial multiplication. In Wolfram Koepf, editor, *ISSAC'2010, Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, Munich, Germany*, page 325–329, New York, July 2010. ACM Press. doi:10.1145/1837934.1837996.

[14] David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time O(nlog n). *Journal of the ACM*, 69(2):12:1–12:40, 2022. doi:10.1145/3505584.

[15] Hsiang-Tsung Kung. On computing reciprocals of power series. *Numerische Mathematik*, 22(5):341–348, October 1974. doi:10.1007/BF01436917.

[16] John P. May, editor. *ISSAC'2009, Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, Seoul, Korea*, New York, July 2009. ACM Press.

[17] Robert Moenck and Allan Borodin. Fast modular transforms via division. In *13th Annual Symposium on Switching and Automata Theory (Swat 1972)*, pages 90–96, October 1972. doi:10.1109/SWAT.1972.5.

[18] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Birkhäuser, Boston, MA, USA, 2001. doi:10.1007/978-1-4612-0129-8.

[19] Daniel S. Roche. Space-and time-efficient polynomial multiplication. In May [16], pages 295–302. doi:10.1145/1576702.1576743.

[20] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013. doi:10.1017/CBO9781139856065.