In-place fast polynomial modular remainder

Jean-Guillaume Dumas, Bruno Grenet

In-place fast polynomial modular remainder

Jean-Guillaume Dumas* Bruno Grenet*

June 27, 2024

Abstract

We consider the simultaneously fast and 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 $m \leq n$. Fast algorithms for this usually come at the expense of a linear amount of extra temporary space. In particular, they require to first compute and store the whole quotient $Q(X)$ such that $A = BQ + R$.

We here propose an *in-place* algorithm (that is with only $\mathcal{O}(1)$ extra space) to compute the sole remainder, using the input space of A and B as intermediate storage but ultimately restoring them to their initial state. If $\mathfrak{M}(k)$ denotes the complexity of a (not-in-place) algorithm to multiply two degree-k polynomials, our algorithm uses at most $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\log(m)\big)$ arithmetic operations, or $\mathcal{O}\big(\frac{n}{m}\mathfrak{M}(m)\big)$ if $\mathfrak{M}(n) = \Theta(n^{1+\epsilon})$ for some $\epsilon > 0$, We also propose variants that compute – still in-place and with the same kind of complexity bounds – the over-place remainder $A(X) \equiv A(X) \mod B(X)$, the accumulated remainder $R(X) \mathrel{+}= A(X) \mod B(X)$ and the accumulated modular multiplication $R(X) \mathrel{+}= A(X)C(X) \mod B(X)$.

To achieve this, we develop techniques for Toeplitz matrix operations whose output is also part of the input, and for convolutions. We reduce these tasks to accumulated polynomial multiplication, for which fast in-place algorithms have recently been developed.

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 usually computed anyway along the algorithm.

In terms of arithmetic operations, from the work of [18, 17] to the more recent results of [3], many sub-quadratic algorithms were developed for this task, based on fast polynomial multiplication [6, 16, 11]. But these fast algorithms require a linear amount of extra temporary space to perform the computation [13]. On the contrary, classical, quadratic algorithms, when computed sequentially, very often require very few (constant) extra registers [19]. To get the best of both worlds and reduce the costs due to space allocation, further work then proposed simultaneously "fast" and "in-place" algorithms for computing both the quotient and the remainder [12, 13].

We here describe fast 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 [13, Alg. 3], we obtain the remainder without needing any space for the quotient. To this end, we use a relaxed in-place model in which the inputs can serve as intermediate storage, *as long as they are restored at the end of the computation.* In practice, the ability to modify the inputs is usually offered to the programmer. Since they are finally restored, these algorithms can be used as subroutines or recursively (in a non-concurrent manner).

As polynomials and Toeplitz matrices are indeed different representations of the same objects, see, e.g., [1, 2, 12], we develop as building blocks fast methods for Toeplitz matrix operations, in-place *with accumulation*

*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

or *over-place*, 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, 12] for fast algorithms, usually do not apply. Instead, we propose reductions to accumulated polynomial multiplication, for which recent fast in-place algorithms have been developed [10, § 4].

Next we first detail our model for in-place computations in Section 1.1 and recall some existing algorithms in Section 1.2. Then, in Sections 2 and 3, we derive novel in-place algorithms for circulant and Toeplitz matrices. Finally, in Sections 4 and 5 we present fast in-place algorithms computing just the polynomial remainder, and for an accumulated modular multiplication.

1.1 Computational model

Our computational model is an *algebraic RAM*. The inputs and outputs are arrays of elements from a field $\mathbb{F}$. The machine is made of *algebraic registers* that each contain one field element, and *pointer registers* that each contain one pointer, that is one integer. Atomic operations are field operations on the algebraic registers and basic pointer arithmetic. We assume that the pointer registers are large enough to store the length of the input/output arrays.

Both inputs and outputs have read/write permissions. But algorithms are only allowed to modify their inputs **if their inputs are restored to their initial state** afterwards. In this model, we call *in-place* an algorithm using only **the space of its inputs, its outputs, and at most $\mathcal{O}(1)$ extra space**. For recursive algorithms, some space may be required to store the recursive call stack. (This stack is only made of pointers and its size is bounded by the recursion depth of the algorithms. In practice, it is managed 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 inputs whose representations share some data. A model with read-only inputs would be more powerful, but mutable inputs turn out to be necessary in our case. In particular, some of 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, we describe *over-place* algorithms, where the output replaces (parts of) its input (e.g., $\vec{a} \leftarrow b \cdot \vec{a}$). 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.

Our model is somewhat similar to catalytic machines and transparent space [5]. But here we allow only the input and output as catalytic space, and we do preserve the (not in-place) time complexity bound, up to a (quasi)-linear overhead. We refer to [5, 21, 12] for more details on these models.

1.2 Existing algorithms

Classical, quadratic, algorithms for polynomial remaindering and triangular matrix operations can be performed in-place, as recalled in Appendix A.

Let $\mathfrak{M}(M)$ be a sub-multiplicative complexity bound on an algorithm computing (not-in-place) the multiplication of polynomials of degree M. Then there exists (not in-place) algorithms such that $M \times M$ Toeplitz matrix-vector multiplication as well as triangular Toeplitz system solve require less than $\mathcal{O}(\mathfrak{M}(M))$ operations and $\mathcal{O}(M)$ extra space [13, 16, 11]. Further, for sparse inputs, $M \times M$ sparse Toeplitz (constant number of non-zero bands) matrix-vector multiplication as well as triangular sparse Toeplitz system solve require less than $\mathcal{O}(M)$ operations and $\mathcal{O}(M)$ extra space.

In an in-place setting, another difficulty arises when performing *accumulating* operations: the result is not even available to store intermediate computations. Recently, in [10], a bilinear technique was developed for the fast accumulated in-place polynomial multiplication, preserving an $\mathcal{O}(\mathfrak{M}(M))$ complexity bound. In other word, there now exists fast routines computing $C(X) \mathrel{+}= A(X)B(X)$ in-place, for any degrees. In the

following, we thus 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 initial idea is to unroll a first recursive iteration and then to call any in-place accumulated polynomial multiplication on halves.

Algorithm 1 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 $A(X) = a_0(X)+X^t a_1(X)$, $B(X) = b_0(X)+X^t b_1(X)$, $C(X) = c_0(X) + X^t c_1(X)$, 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 $\tau_0 + X^t\tau_1 = a_0b_1 + a_1b_0$. Since $X^{2t} = X^n \equiv f \mod X^n - f$, we have that: $C + AB \mod (X^n - f) = C + a_0b_0 + X^t\tau_0 + f\cdot\tau_1 + f\cdot a_1b_1$. This can be computed with 4 full accumulated sequential products, each of degree no more than $2t-2 = n-2$, and by exchanging the lower and upper parts when accumulating $a_0b_1X^t$ and $a_0b_1X^t$: this comes from the fact that $(\tau_0+X^t\tau_1)X^t \equiv X^t\tau_0 + f\cdot\tau_1 \mod X^n - f$.

À la Karatsuba, a more efficient version could use only 3 full polynomial products: let instead $m_0 = m_{00} + m_{01}X^t = a_0b_0$, $m_1 = m_{10} + m_{11}X^t = (a_0+a_1)(b_0+b_1)$ and $m_2 = m_{20} + m_{21}X^t = a_1b_1$ be these 3 full products, to be computed and accumulated in-place. As $X^{2t} = X^n \equiv f \mod X^n$, then $C + AB \mod (X^n - f)$ is also:

$$c_0 + m_{00} + fm_{20} + f(m_{11} - m_{01} - m_{21}) + \Big(c_1 + m_{01} + fm_{21} + (m_{10} - m_{00} - m_{20})\Big)X^t. \quad (1)$$

Equation (1) is an in-place accumulating linear computation, in the sense of [10]: it computes $\vec{c} \mathrel{+}= \mu\vec{m}$ where $\vec{m} = (\alpha\vec{a}) \odot (\beta\vec{b})$, for $\alpha = \beta = \begin{bmatrix}1&0\\1&1\\0&1\end{bmatrix} \in \mathbb{F}^{3\times 2}$ and $\mu = \left[\begin{array}{cc|cc|cc} 1&-f&0&f&f&-f\\-1&1&1&0&-1&f\end{array}\right] \in \mathbb{F}^{2\times 6}$. If $f \notin \{0,1\}$ and $\mu = [M_0|M_1|M_2]$, $M_0^{-1} = (1-f)^{-1}\begin{bmatrix}1&f\\1&1\end{bmatrix}$, $M_1^{-1} = \begin{bmatrix}0&1\\f^{-1}&0\end{bmatrix}$, and $M_2^{-1} = (f^2-f)^{-1}\begin{bmatrix}f&f\\1&f\end{bmatrix}$. From this, [10, Alg. 3] derives an accumulating in-place algorithm, using 2×2 invertible blocks, that computes $\begin{bmatrix}c_i\\c_j\end{bmatrix} \mathrel{+}= M\begin{bmatrix}\rho_0\\\rho_1\end{bmatrix}$, via the equivalent algorithm: $\begin{bmatrix}c_i\\c_j\end{bmatrix} \mathrel{*}= M^{-1}$; $\begin{bmatrix}c_i\\c_j\end{bmatrix} \mathrel{+}= \begin{bmatrix}\rho_0\\\rho_1\end{bmatrix}$; $\begin{bmatrix}c_i\\c_j\end{bmatrix} \mathrel{*}= M$. After some simplifications described below, we obtain Algorithm 1.

The algorithm is obtained from the output of [10, Alg. 3] after applying the following simplifications:

(i) As $1 + f(1-f)^{-1} = (1-f)^{-1}$, then $M_0^{-1}\begin{bmatrix}c_0\\c_1\end{bmatrix} = (1-f)^{-1}\begin{bmatrix}1&f\\1&1\end{bmatrix}\begin{bmatrix}c_0\\c_1\end{bmatrix}$ can be sequentially computed via $c_1 \mathrel{+}= c_0$; $c_1 \mathrel{/}= (1-f)$; $c_0 \mathrel{+}= fc_1$.

(ii) As $(M_2^{-1}\cdot M_0) = \begin{bmatrix}0&-1\\-f^{-1}&0\end{bmatrix} = -\begin{bmatrix}1&0\\0&f^{-1}\end{bmatrix}\begin{bmatrix}0&1\\1&0\end{bmatrix}$, 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 $-((M\cdot c) \mathrel{-}= \rho)$.

Algorithm 1 In-place even degree accumulating f-convolution.

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of odd degree $n-1$; $f \in \mathbb{F} \setminus \{0,1\}$.
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: **else**
4: Let $t = n/2$; {n is even},
5: Let $A = a_0 + X^t a_1$; $B = b_0 + X^t b_1$; $C = c_0 + X^t c_1$;
6: $c_1 \mathrel{+}= c_0$;
7: $c_1 \mathrel{/}= (1-f)$; {simplification (i)}
8: $c_0 \mathrel{+}= f \cdot c_1$;
9: $\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$; {$m_0$ via $\mathfrak{M}$ with $a_0 b_0 = \begin{bmatrix} m_{00} \\ m_{01} \end{bmatrix}$}
10: $c_0 \mathrel{/}= f$; {simplifications (ii) and (iii)}
11: $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{-}= a_1 \cdot b_1$; {$m_2$ via $\mathfrak{M}$ with $a_1 b_1 = \begin{bmatrix} m_{20} \\ m_{21} \end{bmatrix}$}
12: $c_0 \mathrel{-}= c_1$; {this is $c_0/f + m_{00}/f - m_{01} + m_{20} - m_{21}$}
13: $c_1 \mathrel{*}= (1-f)$; {simplification (iv)}
14: $c_1 \mathrel{-}= f \cdot c_0$; {this is $c_1 - m_{00} + m_{01} - m_{20} + f m_{21}$}
15: $a_0 \mathrel{+}= a_1$;
16: $b_0 \mathrel{+}= b_1$;
17: $\begin{bmatrix} c_1 \\ c_0 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$; {$m_1$ via $\mathfrak{M}$ with $(a_0 + a_1)(b_0 + b_1) = \begin{bmatrix} m_{10} \\ m_{11} \end{bmatrix}$}
18: $b_0 \mathrel{-}= b_1$;
19: $a_0 \mathrel{-}= a_1$;
20: $c_0 \mathrel{*}= f$; {simplification (v)}
21: **end if**

(iv) Similarly, $-(M_1^{-1} \cdot M_2) = \begin{bmatrix} 1 & -f \\ -1 & 1 \end{bmatrix}$. Up to swaps, this is $\begin{bmatrix} 1 & 0 \\ -f & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1-f \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$, which is equivalent to the sequential computation: $c_0 \mathrel{-}= c_1$; $c_1 \mathrel{*}= (1-f)$; $c_1 \mathrel{-}= f c_0$.

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

Overall, we obtain Alg. 1 that works for even n with $f \notin \{0,1\}$. We then need to design variant algorithms for the other cases. Algorithm 2 deals with the case where $f = 1$ (as M_0 and M_2 are not invertible when $f = 1$). We present here only the version with 4 full products, but a more efficient version with 3 products only like Alg. 1, could also be derived.

Algorithm 3 deals with the odd-n case where f is invertible. For the sake of simplicity, we also only present the version with 4 full products. The additional difficulty here is that the degrees of the lower and upper parts are different. Therefore, Alg. 3 ensures that there is always the correct space to accumulate.

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 $0 = \lambda \cdot 1 + (1-\lambda)g$. 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. 1 to 3, with an overall complexity bound still of $\mathcal{O}(\mathfrak{M}(n))$.*

Algorithm 2 In-place even degree accumulating 1-convolution.

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, let $Y = X^t$, so that $Y^2 \equiv 1$},
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 $\mathfrak{M}$, of degree $2t-2 \leq n-1$}
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**

There remains to compute a short product in the field with 2 elements. For this, we split the inputs and output in 3 blocks of size close to $n/3$ to obtain the 3×14 matrix μ of Eq. (2):

$$\left[\begin{array}{cc|cc|cc|cc|cc|cc|cc} 1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&1&0&1&0&1&0&0&0&0&0 \\ 0&0&1&1&0&1&1&1&1&1&1&0&1&0 \end{array}\right] \cdot \begin{bmatrix} m_{00} \\ m_{01} \\ \vdots \\ m_{61} \end{bmatrix}. \tag{2}$$

This is the 2×2 expansion of [7, Eq. (24)] modulo 2, recalled in Eq. (3), again letting $m_i = m_{i0} + Y m_{i1}$.

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

The five first pairs of columns of Eq. (2) can be dealt with the technique of [10, Alg. 3]. The last two pairs are not full rank and thus cannot be handled like this (in other words, there is no space to put the high order terms of these two products, even though they 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, of degree $3t$ and $3t+1$, when applicable, are just accumulated directly, scalar multiplications by scalar multiplications. The whole process is shown in Alg. 4.

Overall, Alg. 4 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. 1 to 4 all together provide a complete solution for the fast in-place convolution with accumulation, as given in Alg. 5.

Theorem 2. *Using an in-place polynomial multiplication with complexity bounded by $\mathfrak{M}(n)$, Alg. 5 is correct, in-place and has complexity bounded by $\mathcal{O}(\mathfrak{M}(n))$.*

Algorithm 3 In-place odd degree accumulating f-convolution.

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of even degree $n-1$; $f \in \mathbb{F}^*$.
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.
end if
Let $t = (n+1)/2$; {n is odd, so that $X^{2t} \equiv fX$},
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 $n-1-t = t-2$}
$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \mathrel{+}= a_0 \cdot b_0$ {via $\mathfrak{M}$, of degree $2t-2 \leq n-1$}
$\begin{bmatrix} c_{1..(t-1)} \\ c_1 \end{bmatrix} \mathrel{+}= a_1 \cdot b_1$ {via $\mathfrak{M}$, of degree $(2t-4)+1 \leq n-1$}
$c_{0..(t-2)} \mathrel{/}= f$;
$\begin{bmatrix} c_{t..(2t-2)} \\ c_{0..(t-2)} \end{bmatrix} \mathrel{+}= a_0 \cdot b_1$ {via $\mathfrak{M}$, since $2t-3 < n-1$ and $(u + vX^{t-1})X^t \equiv fv + uX^t$}
$\begin{bmatrix} c_{t..(2t-2)} \\ c_{0..(t-2)} \end{bmatrix} \mathrel{+}= a_1 \cdot b_0$ {via $\mathfrak{M}$, since $2t-3 < n-1$ and $(u + vX^{t-1})X^t \equiv fv + uX^t$}
$c_{0..(t-2)} \mathrel{*}= f$;

Algorithm 4 In-place accumulating short product (0-convolution).

Input: $A(X)$, $B(X)$, $C(X)$ polynomials of degree $< n$ in $\mathbb{F}[X]$;
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 $\mathbb{F} \setminus \{0,1\} \neq \emptyset$ **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 2 or Alg. 3}
  return $C \mathrel{+}= (1-\lambda)AB \mod (X^n - g)$. {Algorithm 1 or Alg. 3}
end if {the rest is thus just when $\mathbb{F} \simeq \mathbb{F}_2$}
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 $m_i = \begin{bmatrix} m_{i0} & m_{i1} \end{bmatrix} = m_{i0} + m_{i1} X^t$, $i = 0..6$, as in Eq. (3);
$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \mathrel{+}= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} m_0 & m_1 & m_2 & m_3 & m_4 \end{bmatrix}^\intercal$ {[10, 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: $c_2 \mathrel{+}= m_5 \mod X^t$}
$b_0 \mathrel{-}= b_1$; $a_1 \mathrel{-}= a_2$;
$a_0 \mathrel{+}= a_2$; $b_1 \mathrel{+}= b_2$;
$c_2 \mathrel{+}= a_0 \cdot b_1 \mod X^t$ {recursive call: $c_2 \mathrel{+}= m_6 \mod X^t$}
$b_1 \mathrel{-}= b_2$; $a_0 \mathrel{-}= a_2$;
if $n \geq 3t+1$ **then** {scalar accumulations}
  for $i = 0$ **to** $3t$ **do** $c_{3t} \mathrel{+}= A_i \cdot B_{3t-i}$ **end for**
end if
if $n = 3t+2$ **then** {scalar accumulations}
  for $i = 0$ **to** $3t+1$ **do** $c_{3t+1} \mathrel{+}= A_i \cdot B_{3t+1-i}$ **end for**
end if

Algorithm 5 In-place convolution with accumulation.

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

1: **if** $f = 0$ **then return** Alg. 4 **end if**
2: **if** n is odd **then return** Alg. 3 **end if** $\{f \neq 0\}$
3: **if** $f = 1$ **then return** Alg. 2 **end if** $\{n \text{ is even}\}$
4: **return** Alg. 1. $\{f \notin \{0, 1\}\}$

Proof. Correctness of Alg. 1 comes from that of Eq. (1). Correctness of Algs. 2 and 3 is direct from the degrees of the sub-polynomials (as given in the comments, line by line). Correctness of Alg. 4 comes from that of Eq. (3). Also, all four algorithms are in-place as they use only in-place atomic operations or in-place polynomial multiplication.

The complexities of Algs. 1 to 3 satisfy $T(n) \leq 4\mathfrak{M}(n/2) + \mathcal{O}(n)$ since they use only a linear number of operations plus up to four calls to polynomial multiplications of half-degrees. Since $\mathfrak{M}(n)/n$ is non-decreasing, $T(n) = \mathcal{O}(\mathfrak{M}(n))$. Finally, Alg. 4 uses a linear number of atomic operations, five degree-$n/3$ polynomial multiplications and only two recursive calls with degree-$n/3$ polynomials. Thus, its complexity satisfies $T(n) \leq 2T(n/3) + 5\mathfrak{M}(n/3) + \mathcal{O}(n)$, whence $T(n) = \mathcal{O}(\mathfrak{M}(n))$ since $\mathfrak{M}(n)/n$ is non-decreasing. □

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., [20, Theorem 2.6.4].

3.1 Accumulating & in-place f-circulant

Definition 3. *For $\vec{a} \in \mathbb{F}^m$, $\mathscr{C}(\vec{a})$ is the* circulant matrix *represented by $\vec{a}$, that 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 $f \in \mathbb{F}$ and $\vec{a} \in \mathbb{F}^m$, 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 $X^n - f$ [20, 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. 5. Note that, as recalled in Appendix B, the case $f = 1$ can also be computed using a discrete Fourier transform, but only when primitive roots of sufficiently large order exist. By contrast, Alg. 5 has no restriction on f and is a reduction to any accumulated in-place polynomial multiplication (including DFT ones, as, e.g., [10, Alg. 7]).

3.2 Accumulating & in-place Toeplitz

Definition 5. *For $\vec{a} \in \mathbb{F}^{2m-1}$, $\mathcal{T}(\vec{a})$ is the (square)* Toeplitz matrix *represented by $\vec{a}$, that is, the $m \times m$ matrix (T_{ij}), such that $T_{ij} = a_{m+j-i}$. Similarly, for $\vec{a} \in \mathbb{F}^{m+n-1}$, 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.*

The matrix-vector product for rectangular Toeplitz matrices and the middle product of polynomials are the same task, see, e.g., [12, § 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. 6 and 7, and then for any Toeplitz matrix, again as sums of triangular Toeplitz matrices in Alg. 8.

Algorithm 6 In-place accumulating Upp. Triang. Toeplitz m-v. mult.

Input: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{F}^m$.
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 5}

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. 6, with a single call to (the transposed version of) Alg. 5. 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 7 In-place accumulating square Toeplitz m-v. mult.

Input: $\vec{a_1} \in \mathbb{F}^m$, $\vec{a_2}, \vec{b}, \vec{c} \in \mathbb{F}^{m+1}$,
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 5}
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: $\overleftarrow{c_2} \mathrel{+}= \mathscr{C}_0(\overleftarrow{a_1}) \cdot \overleftarrow{b_1}$; {Eq. (4) and Algorithm 5}
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 6 and 7 are correct and have complexity bounded by $\mathcal{O}(\mathfrak{M}(n))$.*

Proof. The complexity bound comes from that of Alg. 5. Correctness is obtained directly looking at the values of the matrices. First, for Alg. 6, we have that $\mathcal{T}([\vec{0}, \vec{a}]) = \mathscr{C}_0(\vec{a})$. Second, for Alg. 7, Remark 6 and Eq. (4) show that it is possible to compute $\mathscr{C}_0(\overleftarrow{a_1})^{\mathsf{T}} \cdot \vec{b_1}$ via the reverse of $\mathscr{C}_0(\overleftarrow{a_1}) \cdot \overleftarrow{b_1}$. $\mathcal{T}([\vec{a_1}, \vec{a_2}]) = \mathscr{C}_0(\vec{a_2}) + \begin{bmatrix} \vec{0}^{\mathsf{T}} & 0 \\ \mathscr{C}_0(\overleftarrow{a_1})^{\mathsf{T}} & \vec{0} \end{bmatrix}$. ∎

From this, we give in Alg. 8 an in-place rectangular Toeplitz matrix-vector multiplication.

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

Proof. If $m>n$, there are $\lfloor m/n \rfloor$ calls to Alg. 7 in size n, requiring $\mathcal{O}((m/n)\mathfrak{M}(n)) \leq \mathcal{O}(\mathfrak{M}(m))$ operations. The remaining recursive call is then negligible. This is similar when $m<n$. ∎

These in-place accumulated Toeplitz matrix-vector multiplications, in turns, allow us to obtain both over-place triangular Toeplitz multiplication or system solve, given in Algs. 9 and 10.

Proposition 9. *Algorithm 9 is correct and requires less than $\mathcal{O}(\mathfrak{M}(m)\log(m))$ operations, or $\mathcal{O}(\mathfrak{M}(m))$ if $\mathfrak{M}(m) = \Theta(m^{1+\epsilon})$ for some $\epsilon > 0$.*

Algorithm 8 In-place accumulating rectangular Toeplitz matrix-vector multiplication.

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

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

Algorithm 9 Over-place triang. Toeplitz m-v. mult.

Input: $\vec{a}, \vec{b} \in \mathbb{F}^m$, s.t. $a_1 \in \mathbb{F}^*$.
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 triang. m-v. mult. {Algorithm 17}
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 8}
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 $T = \begin{bmatrix} T_1 & 0 \\ G & T_2 \end{bmatrix}$. Thus $T\vec{b} = \begin{bmatrix} T_1 b_1 \\ Gb_1 + T_2 b_2 \end{bmatrix}$. 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 the algorithm is correct. Now for the complexity bound, the cost function is $T(m) \leq 2T(m/2) + \mathcal{O}(\mathfrak{M}(m))$, that is $\mathcal{O}(\mathfrak{M}(m)\log(m))$, or $\mathcal{O}(\mathfrak{M}(m))$ if $\mathfrak{M}(m) = \Theta(m^{1+\epsilon})$ for some $\epsilon > 0$. □

Proposition 10. *Algorithm 10 is correct and requires less than $\mathcal{O}(\mathfrak{M}(m)\log(m))$ operations, or $\mathcal{O}(\mathfrak{M}(m))$ if $\mathfrak{M}(m) = \Theta(m^{1+\epsilon})$ for some $\epsilon > 0$.*

Proof. First, let $T = \mathcal{T}([\vec{0}, \vec{a}])$ and consider it as blocks $T_1 = \mathcal{T}([\vec{0}, \vec{a}_{1..k}])$, $T_2 = \mathcal{T}([\vec{0}, \vec{a}_{1..(m-k)}])$ and $G = \mathcal{T}_{\text{k,m-k}}(\vec{a}_{2..m})$. Then $T = \begin{bmatrix} T_1 & G \\ 0 & T_2 \end{bmatrix}$. Now define H, s.t. $T^{-1} = \begin{bmatrix} T_1^{-1} & H \\ 0 & T_2^{-1} \end{bmatrix}$. Then H satisfies $T_1^{-1}G + HT_2 = 0$. Also, we have $T^{-1}\vec{b} = \begin{bmatrix} T_1^{-1}b_1 + Hb_2 & T_2^{-1}b_2 \end{bmatrix}^\intercal$. 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 $T(m) \leq 2T(m/2) + \mathcal{O}(\mathfrak{M}(m))$, that is $T(m) = \mathcal{O}(\mathfrak{M}(m)\log(m))$, or $\mathcal{O}(\mathfrak{M}(m))$ if $\mathfrak{M}(m) = \Theta(m^{1+\epsilon})$ for some $\epsilon > 0$. □

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 10 Over-place triang. Toeplitz system solve.

Input: $\vec{a}, \vec{b} \in \mathbb{F}^m$, s.t. $a_1 \in \mathbb{F}^*$.
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 triang. syst. solve. {Algorithm 17}
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: $b_2 \leftarrow \mathcal{T}([\vec{0}, \vec{a}_{1..(m-k)}])^{-1} \cdot b_2$; {recursive call}
6: $b_1 \mathrel{-}= \mathcal{T}_{\text{k,m-k}}(\vec{a}_{2..m}) \cdot b_2$; {Algorithm 8}
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 $R(X) \equiv A(X) \bmod B(X)$ (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 $\mathcal{O}\left(\frac{n}{m}\mathfrak{M}(m)\right)$ arithmetic operations and, apart from that of A and B, at least $\mathcal{O}(n-m)$ extra memory [13]. 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 $\mathcal{O}\left(\frac{n}{m}\mathfrak{M}(m)\right)$ 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 $\mathcal{O}(n)$).

Second, 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 derive in-place and over-place algorithms for the modular remainder using at most $\mathcal{O}(\mathfrak{M}(m)\log(m))$ arithmetic operations, or $\mathcal{O}(\mathfrak{M}(m))$ if $\mathfrak{M}(m) = \Theta(m^{1+\epsilon})$ for some $\epsilon > 0$.

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 Eq. (5), 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 $B = \mathcal{T}([0, \ldots, 0, b_M, \ldots, b_0, 0, \ldots, 0])$:

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

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):

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

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:

$$T = \begin{bmatrix} b_M & \cdots & \cdots & b_1 \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ 0 & & & b_M \end{bmatrix} \text{ and } G = \begin{bmatrix} b_0 & & 0 \\ \vdots & \ddots & \\ \vdots & & \ddots \\ b_{M-1} & \cdots & b_0 \end{bmatrix} . \tag{7}$$

This in turns gives a way to access only the first coefficients of Q in an upper triangular Toeplitz system:

$$\begin{bmatrix} q_0 \\ \vdots \\ \vdots \\ \vdots \\ q_{M-1} \end{bmatrix} = \begin{bmatrix} I_M & 0 \end{bmatrix} \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ \vdots \\ a_N \end{bmatrix} . \tag{8}$$

with a 2-block band structure where $\begin{bmatrix} I_M & 0 \end{bmatrix}$ 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:

$$R = \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ \vdots \\ a_{M-1} \end{bmatrix} - \begin{bmatrix} G & 0 \end{bmatrix} \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ \vdots \\ a_N \end{bmatrix} . \tag{9}$$

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., [8, Eq.(1)] for the scalar case). If we denote by H_i the blocks of that row, we have Eq. (10):

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

which solves as: $H_i = T^{-1}(-GT^{-1})^i = (-GT^{-1})^i T^{-1}$.

We have shown:

Lemma 11.

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

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

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

From Eq. (11) we thus can immediately deduce the following Alg. 11 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 11 Overwritten-quotient Euclidean remainder.

Input: $A(X)$, $B(X)$ in $\mathbb{F}[X]$ 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. **end if**
2: Let $n = N - M + 1$, $\mu = \lceil \frac{n}{M} \rceil$;
3: Let $[\vec{a_0}, \ldots, \vec{a_\mu}] = [a_0, \ldots, a_N, \vec{0}]$; {blocks of dimension M}
4: Let $T = \mathcal{T}([\vec{0}_{M-1}, b_M, \ldots, b_1])$, $G = \mathcal{T}([b_{M-1}, \ldots, b_0, \vec{0}_{M-1}])$;
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 11 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 is given by Lemma 11 and Eq. (11). For the complexity bounds, we use Section 1.2: for each block, the triangular Toeplitz system solve and the Toeplitz m-v. mult. require respectively $\lambda_\sigma \mathfrak{M}(m)$ and $\lambda_\tau \mathfrak{M}(m)$ operations and, sequentially, $\max\{\sigma;\tau\}M$ extra space. Apart from this space, we only need one extra vector, $\vec{t}$, to store intermediate results. Overall we thus perform $\mu((\lambda_\sigma + \lambda_\tau)\mathfrak{M}(M) + M)$ operations. With $\mu = n/M$ and $n = N - M + 1$, this is $\left\lceil \frac{N-M+1}{M} \right\rceil ((\lambda_\sigma + \lambda_\tau)\mathfrak{M}(M) + M) = \mathcal{O}(\frac{N}{M}\mathfrak{M}(M))$ operations, using $(1 + \max\{\sigma;\tau\})M$ 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 11 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).*

An in-place atomic (and thus quadratic) long division is given in [19]. In-place long division by blocks is also sketched for instance in the proof of [13, 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 bound 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 2, 3 and 4.1.

We present three variants of in-place polynomial remaindering:

- IPER: given A and B, it computes in-place $R = A \bmod B$ 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: given A and B, it computes both $Q = A \operatorname{div} B$ and $R = A \bmod B$ (such that $A = BQ + R$), replacing A by $\langle Q, R\rangle$ (with B restored after completion);
- APER: given A, B and R, it computes R += $A \bmod B$, accumulating the remainder into A (with both A and B restored after completion).

We present IPER in Alg. 12: this variant replaces only Lines 7 to 9 of Alg. 11 by their over-place variants, Algs. 9 and 10, that modify and restore some parts of B.

Algorithm 12 IPER (R, A, B): In-place Polynomial Euclidean Remainder.

Input: $A(X)$, $B(X)$ in $\mathbb{F}[X]$ 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. 11;
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 10}
6: $\quad \vec{r} \leftarrow (-G) \cdot \vec{r}$; {Algorithm 9}
7: $\quad \vec{r}$ += $\vec{a_i}$;
8: **end for**
9: **return** $R = \sum_{i=0}^{M-1} r_i X^i$.

We here give the variants of Alg. 12 where, in Alg. 13, the polynomial is overwritten by its quotient and remainder, see Remark 13; and where, in Alg. 14, the remainder is accumulated.

Algorithm 13 OPER(A, B): Over-place Polynomial Euclidean Quotient and Remainder.

Input: $A(X)$, $B(X)$ in $\mathbb{F}[X]$ of respective degrees N and M.
Output: A is replaced by both $A(X)/B(X)$ and $A(X) \mod B(X)$ of degrees $N-M$ and at most $M-1$.

1: **if** $M > N$ **then return** A.
2: Let n, μ, T, G as in Alg. 12 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: Let $T_1 = \mathcal{T}([\vec{0}_{s-1}, b_M, \ldots, b_{M-s+1}])$; {$s \times s$ upper left of T}
6: $\vec{a_\mu} \leftarrow T_1^{-1} \cdot \vec{a_\mu}$; {Algorithm 10}
7: Let $G_1 = \mathcal{T}_{\mathrm{M,s}}([b_{M-1}, \ldots, b_0, \vec{0}_{s-1}])$; {left s columns of G}
8: $\vec{a}_{\mu-1} \mathrel{-}= G_1 \cdot \vec{a_\mu}$; {Algorithm 8}
9: **end if**
10: **for** $i = \mu-1$ **down-to** 1 **do**
11: $\vec{a_i} \leftarrow T^{-1} \cdot \vec{a_i}$; {Algorithm 10}
12: $\vec{a}_{i-1} \mathrel{-}= G \cdot \vec{a_i}$; {Algorithm 7}
13: **end for**
14: **return** the quotient $[\vec{a_1}, \ldots, \vec{a}_\mu]$ and the remainder $\vec{a_0}$.

A nice property of OPER is that it is reversible: by computing in the reverse order $\vec{a}_{j-1} \mathrel{+}= G \cdot \vec{a_j}$; $\vec{a_j} \leftarrow T \cdot \vec{a_j}$ (and similarly with G_1 and T_1) one reverses the algorithm and recovers A. We denote by OPER^{-1} this recovery.

With this, APER is now just the application of OPER and OPER^{-1} interleaved with an update of the remainder. This is shown in Alg. 14.

Algorithm 14 APER(R, A, B): Accumulated in-place Polynomial Euclidean Remainder.

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

1: **if** $M > N$ **then return** $R \mathrel{+}= A$.
2: Let n, μ, $[\vec{a_0}, \ldots, \vec{a_\mu}]$ as in Alg. 13, and $\vec{r} = [r_0, \ldots, r_{M-1}]$;
3: OPER(A, B); {quotient and remainder, $\vec{a_0}$, in-place of A, via Algorithm 13}
4: $\vec{r} \mathrel{+}= \vec{a_0}$;
5: OPER$^{-1}(A, B)$; {recover A from quotient and remainder}
6: **return** $\vec{r}$.

All together, we obtain the following result.

Theorem 14. *Algorithms 12 to 14 are correct, in-place and require less than* $\mathcal{O}\left(\frac{N}{M}\mathfrak{M}(M)\log(M)\right)$ *operations, or* $\mathcal{O}\left(\frac{N}{M}\mathfrak{M}(M)\right)$ *if* $\mathfrak{M}(M) = \Theta(M^{1+\epsilon})$ *for some* $\epsilon > 0$.

Proof. Algorithm 12 calls $\mu = \left\lceil \frac{N-M+1}{M} \right\rceil = \mathcal{O}\left(\frac{N}{M}\right)$ times Algs. 9 and 10, each call requiring less than $\mathcal{O}(\mathfrak{M}(M)\log(M))$ operations, or $\mathcal{O}(\mathfrak{M}(M))$ is $\mathfrak{M}(M) = \Theta(M^{1+\epsilon})$ for some $\epsilon > 0$, by Propositions 9 and 10. This suffices to prove the Theorem's claims for Alg. 12. These costs also dominates the costs of Algs. 7 and 8 by Lemma 7 and Proposition 8, and therefore that of Alg. 13: its Lines 5 to 8 are the over-place first iteration of Alg. 12, for $s < M$. Finally, Alg. 14 is also similar, using the same sub-algorithms, and only roughly doubling the overall cost (by symmetry before and after the accumulation). □

5 In-place modular multiplication

As a direct application of this possibility to compute modular remainder in-place, we show that modular multiplications, of the form $R(X) \mathrel{+}= A(X)C(X) \bmod B(X)$, can be computed in-place. The idea is to use

a similar technique as for algorithm APER (Alg. 14), but computing some coefficients of AC only when they are needed. These latter computations can be performed in-place of C, provided that they can be reverted in the end. This is actually always possible if A is smaller than both B and C, as explained, thereafter, in Alg. 15 and Proposition 15. If C is smaller than both A and B, then of course we simply exchange the roles of A and C.

This shows that multiplications in a polynomial extension of a finite field can be computed in-place. Indeed, this is a special case where A and C are both smaller than B, as operations are performed modulo the irreducible polynomial of the extension and elements of the extension field are polynomials of degree upper bounded by that of this irreducible polynomial.

Finally, in the remaining cases, we show in Alg. 16 and Proposition 16 that one can always start by reducing the smaller of A or C by B, in-place using OPER (Alg. 13), as the latter reduction is reversible.

Algorithm 15 AXPYIN(R, A, C, B): Accumulated in-place modular multiplication.

Input: $R(X)$, $A(X)$, $C(X)$, $B(X)$ in $\mathbb{F}[X]$ of resp. degrees $M-1$, L, N and M, with $L \leq \min\{N, M\}$.
Output: $R(X) \mathrel{+}= A(X)C(X) \mod B(X)$ of degree at most $M-1$.

1: **if** $L + N < M$ **then return** $R \mathrel{+}= AC$. {via $\mathfrak{M}$}
2: Let $q = L + N - M$; $\overleftarrow{b_1} = \vec{b}_{(M-1)..0}$; $\overleftarrow{b_2} = \vec{b}_{M..\max\{0,M-q+1\}}$;
3: Let $\overleftarrow{a_1} = \vec{a}_{L..\max\{0,M-L+1\}}$, $\overleftarrow{a_2} = \vec{a}_{L..\max\{0,L-q+1\}}$;
4: Let $\vec{c_2} = \vec{c}_{(N-q)..N}$, $\vec{r_1} = \vec{r}_{0..q}$;
5: Let $G = \mathcal{T}_{\mathrm{M,q+1}}(\overleftarrow{b_1}, \vec{0}_q)$, $T = \mathcal{T}_{\mathrm{q+1,q+1}}(\vec{0}_q, \overleftarrow{b_2}, \vec{0}_{\max\{0,q-M\}})$;
6: Let $A_0 = \mathcal{T}_{\mathrm{M,N+1}}(\vec{0}_{\max\{0,M-L-1\}}, \overleftarrow{a_1}\, \vec{0}_N)$;
7: Let $A_\mu = \mathcal{T}_{\mathrm{q+1,q+1}}(\vec{0}_q, \overleftarrow{a_2}, \vec{0}_{\max\{0,q-L\}})$;
8: $\vec{c_2} = A_\mu \cdot \vec{c_2}$; {Algorithm 9}
9: $\vec{c_2} = T^{-1} \cdot \vec{c_2}$; {Algorithm 10, the quotient of AC by B}
10: $\vec{r_1} \mathrel{-}= G \cdot \vec{c_2}$; {Algorithm 8, upper triang. rect. Toeplitz}
11: $\vec{c_2} = T \cdot \vec{c_2}$; {Algorithm 9, undo Line 9}
12: $\vec{c_2} = A_\mu^{-1} \cdot \vec{c_2}$; {Algorithm 10, undo Line 8}
13: **return** $\vec{r} \mathrel{+}= A_0 \cdot \vec{c}$; {Algorithm 8, banded Toeplitz}

Proposition 15. *Algorithm 15 is correct and requires less than* $\mathcal{O}\big(\big(\log(M) + \lceil \frac{N}{M} \rceil\big)\, \mathfrak{M}(M)\big)$ *operations, or only* $\mathcal{O}\big(\lceil \frac{N}{M} \rceil\, \mathfrak{M}(M)\big)$ *if* $\mathfrak{M}(M) = \Theta(M^{1+\epsilon})$ *for some* $\epsilon > 0$.

Proof. Replacing the left-hand side of Eq. (5) by AC we see that $AC = BQ + R$ can be written as Eq. (12):

$$\begin{bmatrix} a_0 & & 0 \\ \vdots & \ddots & \\ a_L & & a_0 \\ & \ddots & \vdots \\ 0 & & a_L \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} b_0 & & 0 \\ \vdots & \ddots & \\ b_M & & b_0 \\ & \ddots & \vdots \\ 0 & & b_M \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ \vdots \\ q_{N+L-M} \end{bmatrix} + \begin{bmatrix} r_0 \\ \vdots \\ r_{M-1} \\ 0 \end{bmatrix}. \tag{12}$$

With the notations of Alg. 15, Eq. (12) becomes: $\begin{bmatrix} A_0 \\ A_\mu \end{bmatrix} \cdot \vec{c} = \begin{bmatrix} G \\ T \end{bmatrix} \cdot \vec{q} + \vec{r}$ which gives both $A_\mu \cdot \vec{c_2} = T \cdot \vec{q}$ and $A_0 \cdot \vec{c} = G \cdot \vec{q} + \vec{r}$. Then, as $M \geq L$ implies $N \geq q$, both A_μ and T are invertible upper triangular matrices (with the leading polynomial coefficient in the diagonal). The residue to be added is thus $A_0 \cdot \vec{c} - G \cdot T^{-1} \cdot A_\mu \cdot \vec{c_2}$, as in Alg. 15.

Now for the complexity, A_μ, G and T are matrices with dimensions smaller that $M \times M$. Toeplitz operations with those thus require less than $\mathcal{O}(\mathfrak{M}(M) \log(M))$ operations (or $\mathcal{O}(\mathfrak{M}(M))$ if $\mathfrak{M}(M) = \Theta(M^{1+\epsilon})$), by Propositions 8 to 10. Finally, A_0 is an $M \times N$ Toeplitz so that the cost of applying it to a vector is bounded by $\mathcal{O}(\lceil \frac{N}{M} \rceil\, \mathfrak{M}(M))$, by Proposition 8. ∎

Finally, Alg. 16 shows that one can compute $R \mathrel{+}= AC \bmod B$, without any constraint on the degrees. For this we combine Algs. 13 and 15: if A is larger than C, compute $R \mathrel{+}= CA \bmod B$ instead; if A is larger

than B, start by reducing it in-place via OPER (Alg. 13), then compute the multiplication by Alg. 15 and eventually restore A via OPER^{-1}.

Algorithm 16 Fully in-place accumulated modular multiplication.

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

1: **if** $L > N$ **then return** $R \mathrel{+}= CA \mod B$. {recursive call}
2: **if** $L \leq M$ **then return** $\text{AXPYIN}(R, A, C, B)$. {Algorithm 15}
{now $N \geq L > M$, i.e., $d^\circ C \geq d^\circ A > d^\circ B$}
3: $\text{OPER}(A, B)$; {quotient and remainder, $\vec{a_0}$, in-place of A, via Algorithm 13}
4: $\text{AXPYIN}(R, \vec{a_0}, C, B)$; {Algorithm 15}
5: $\text{OPER}^{-1}(A, B)$; {recover A from quotient and remainder}
6: **return** R.

Proposition 16. *Algorithm 16 is correct and requires less than* $\mathcal{O}\big(\big(\log(M) + \lceil \frac{L+N}{M} \rceil\big)\, \mathfrak{M}(M)\big)$ *operations, or only* $\mathcal{O}\big(\lceil \frac{L+N}{M} \rceil\, \mathfrak{M}(M)\big)$ *if* $\mathfrak{M}(M) = \Theta(M^{1+\epsilon})$ *for some* $\epsilon > 0$.

Proof. This is the combination of Theorem 14 and Proposition 15. □

6 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. Section 5 also shows a direct application of these techniques for the multiplication in a polynomial extension.

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.

[3] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03, Philadelphia, Pennsylvania*, pages 37–44, August 2003. 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 *ISSAC'09, Seoul, Korea*, pages 135–143, July 2009. doi:10.1145/1576702.1576713.

[5] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *STOC'14*, pages 857–866. ACM, 2014. doi:10.1145/2591796.2591874.

[6] 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.

This material is based on work supported in part by the Agence Nationale pour la Recherche under Grants ANR-21-CE39-0006, ANR-15-IDEX-0002 and ANR-22-PECY-0010.

[7] 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.

[8] 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.

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

[10] Jean-Guillaume Dumas and Bruno Grenet. In-place accumulation of fast multiplication formulae. In *ISSAC'24, Raleigh, NC, USA*, July 2024. URL: https://hal.science/hal-04167499.

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

[12] Pascal Giorgi, Bruno Grenet, and Daniel S. Roche. Generic reductions for in-place polynomial multiplication. In *ISSAC'19, Beijing, China*, pages 187–194, July 2019. doi:10.1145/3326229.3326249.

[13] Pascal Giorgi, Bruno Grenet, and Daniel S. Roche. Fast in-place algorithms for polynomial operations: division, evaluation, interpolation. In *ISSAC'20, Kalamata, Greece*, pages 210–217, July 2020. doi:10.1145/3373207.3404061.

[14] 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.

[15] David Harvey and Daniel S. Roche. An in-place truncated Fourier transform and applications to polynomial multiplication. In *ISSAC'10, Munich, Germany*, page 325–329, July 2010. doi:10.1145/1837934.1837996.

[16] 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.

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

[18] 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.

[19] Michael Monagan. In-place arithmetic for polynomials over zn. In *DISCO'92*, pages 22–34, 1993. doi:10.1007/3-540-57272-4_21.

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

[21] Daniel S. Roche. Space-and time-efficient polynomial multiplication. In *ISSAC'09, Seoul, Korea*, pages 295–302, July 2009. doi:10.1145/1576702.1576743.

A In-place quadratic algorithms

For any field $\mathbb{F}$ we give here for instance the following classical over-place algorithms for triangular matrix operations, given in Alg. 17.

Algorithm 17 Over-place quadratic triangular matrix operations
(left: matrix-vector multiplication; right: triangular system solve)

Input: $U \in \mathbb{F}^{m\times m}$ upper triangular and $\vec{v} \in \mathbb{F}^m$
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 $U_{i,i} \in \mathbb{F}^*$}
6: **end for**

The classical long-division algorithm provides a quadratic in-place algorithm for computing the remainder of two polynomials, see Alg. 18.

Algorithm 18 In-place quadratic polynomial remainder.

Input: $A(X)$, $B(X)$ in $\mathbb{F}[X]$, of respective degrees N and M.
Read-only: $A(X)$, $B(X)$.
Output: $R(X) = A(X) \mod B(X)$ of degree at most $M - 1$.
1: **if** $N < M$ **then return** A **end if**
2: Let $\bar{B} = B \bmod X^M$, $n = N - M$ and $R \leftarrow A \operatorname{div} X^{n+1}$;
3: **for** $i = n$ **down-to** 1 **do**
4: $q \leftarrow r_{M-1} \cdot b_M^{-1}$; {leading coefficients of R and B}
5: $R \leftarrow a_i + X \cdot R \bmod X^M$;
6: $R \mathrel{-}= q \cdot \bar{B}$;
7: **end for**
8: **return** R

Algorithm 18 can be made over-place of its input A by considering that R is just a "pointer" to some position in A (with q in r_M): this is then close to the in-place quadratic version given in [19].

B Accumulating circulant matrix vector product

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., [14, § 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 [21, 15, 9]. 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 enables to compute fast in-place accumulated circulant matrix-vector only when primitive roots of sufficiently large order exist.