In-place sub-quadratic polynomial modular remainder

Jean-Guillaume Dumas, Bruno Grenet

In-place sub-quadratic polynomial modular remainder

Jean-Guillaume Dumas* Bruno Grenet*

April 3, 2023

Abstract

We consider the 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 $\mathcal{O}\left(\frac{n}{m}\mathfrak{M}(m)\right)$ arithmetic operations and, apart from that of A and B, $\mathcal{O}(n)$ 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 $\mathcal{O}\left(\frac{n}{m}\mathfrak{M}(m)\right)$ 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 $\mathcal{O}(n)$.

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 $\mathcal{O}(n^{\log_2(3)})$ arithmetic operations over any field of zero or odd characteristic and over most of the characteristic two ones.

To achieve this, we develop techniques for Toeplitz matrix operations which output is also part of the input.

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 [21, 17] to more recent results of [5], many sub-quadratic algorithms were developed for this task, based on fast polynomial multiplication [7, 16, 27]. 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 [6, 12, 13].

We here propose algorithms to extend the latter line of work. In the polynomial setting, we compute 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. As polynomials and Toeplitz matrices are indeed different representations of the same objects, see, e.g., [3, 4, 12], 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

*Université Grenoble Alpes. Laboratoire Jean Kuntzmann, CNRS, UMR 5224. 700 avenue centrale, IMAG - CS 40700, 38058 Grenoble, cedex 9 France. {firstname.lastname}@univ-grenoble-alpes.fr

with absolutely no margin for extra space. Thus, for instance, the generic recursive techniques of [6, 12] usually do not apply.

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 Section 1.3. Then, in Section 2, we derive new in-place algorithms for circulant and Toeplitz matrices over fields of zero or odd characteristic. In Section 3 we then propose a fast remaindering algorithm using only a multiple of the remainder space. Further, in Section 4, we extend this to using as extra space only a small fraction of the remainder space. Finally in Section 5 we combine these techniques to obtain sub-quadratic in-place algorithms for the polynomial remainder only. We also propose in Appendix B some ways to adapt our algorithms to the characteristic two specificities.

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 $\mathcal{O}(1)$ extra space**. But algorithms are only allowed to modify their inputs, **provided that their inputs are restored to their initial state** afterwards. We do not take into account the potential space used by the recursive call stack. (This space is bounded by the recursion depth of the algorithms, and managed in practice by the compiler.) 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 (e.g., like $\vec{a} \mathrel{+}= \vec{b}$). If atomic operations are used to perform the accumulation, then usually a constant number of extra space is sufficient. Differently, fast algorithms usually work with (non-constant) large blocks of data and having only the latitude to use a constant extra-space is often not sufficient.

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).

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 [24, 12].

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 Algorithm 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 Algorithm 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: $\vec{r}$ -= $r_M \cdot \vec{b}$; {Except for last entry (of index M)}
7: **end for**
8: **return** $R(X) = \sum_{i=0}^{M-2} r_{i+1} X^i$.

1.3 Fast algorithms

More generally, if $\mathfrak{M}(M)$ is a submultiplicative 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., [13, 16, 27], 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.

2 Accumulated in-place and over-place Toeplitz operations

We thus now turn to in-place accumulated and over-place Toeplitz matrix-vector operations. 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 instead double the operations while keeping the same dimension. We start by the usual definitions, also extending circulant matrices to f-circulant matrices, following [25] (see also, e.g., [23, Theorem 2.6.4]).

Definition 1. *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 2. *For $f \in \mathbb{D}$ and $\vec{a} \in \mathbb{D}^m$, the* f-circulant matrix *represented by $\vec{a}$, $\mathscr{C}_{\mathrm{f}}(\vec{a})$, is the $m \times m$ matrix (Γ_{ij}), such that:*

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

Definition 3. *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.*

We see immediately that we 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 $\vec{v}$.

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 a first 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 [24, 15, 9]. This gives us an in-place algorithm to compute the *accumulation* $\vec{c}$ += $\mathscr{C}_1(\vec{a}) \cdot \vec{b}$ is:

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

We use this idea and generalize it to f-circulant matrices, using fast methods for f-circulant matrices of [10, 25].

More precisely, we extend [25, Alg. 1] to (1) dimensions that are not a power of two (without any extra space for virtual zeroes), and (2) to any odd characteristic (i.e. we do not need an n-th root of unity), using techniques adapted from [11, Alg. 19]. To preserve a small memory footprint, it is crucial to handle carefully both cases where dimensions are not powers of two, and fields without roots of unity. On the one hand, using the closest larger power of two could double the required space. On the other hand, using an extended field would also multiply the size of the representation.

Our new algorithm is given as Algorithm 3. We use peeling to handle odd dimensions, and sum of squares to handle square roots. For now, we have two cases where this algorithm does not work:

- We need a division by 2, and thus cannot handle the even characteristic case;
- We need at least two squares, and thus cannot handle the field with 3 elements (our algorithm works over an extension, the field with 9 elements for instance, but this would, in theory, double the required space).

The cases we can tackle thus include any field of characteristic zero (complexes, reals, rationals, ...), the characteristic 3 with more than 9 elements, any other odd characteristic.

Theorem 4. *Over any field with zero or odd characteristic (except the field of cardinality 3), Algorithm 3 is correct and requires less than $\mathcal{O}(m^{\log_2(3)})$ operations.*

Proof. When m is even and f is a square, the correctness is that of [25, Alg. 1] (see also an equivalent proof in polynomial form in Appendix A). When f is not a square, we decompose $f \neq 1$ as a sum of squares such that: $(\lambda+1)f = \lambda 1^2 + b^2$. That is $b^2 = g = (\lambda+1)(f-1)+1$. For any $b \neq 0$, $\lambda = (b^2-1)/(f-1)-1$ is a possibility if both λ and $\lambda+1$ are invertible. As soon as the field contains at least 3 distinct non-zero squares, there is always a solution. This includes fields of zero characteristic, and odd characteristic fields with at least 7 elements (odd finite fields for instance have $(q-1)/2$ nonzero squares in $\mathbb{F}_q$). In the field with 5 elements, it is sufficient that both 1 and -1 are squares: it remains to remark that $\mathscr{C}_2 = -\mathscr{C}_1 + 2\mathscr{C}_{\text{-}1}$ and $\mathscr{C}_3 = 2\mathscr{C}_1 - \mathscr{C}_{\text{-}1}$. Thus only the case of $\mathbb{F}_3$ does not have a solution. Then, $\left(\left(\vec{c}(1+\lambda^{-1}) + \mathscr{C}_1(A)\vec{b}\right)\lambda + \mathscr{C}_{\mathrm{g}}(A)\vec{b}\right)\frac{1}{1+\lambda} = \vec{c} + \left(\frac{\lambda}{1+\lambda}\mathscr{C}_1(A) + \frac{1}{1+\lambda}\mathscr{C}_{\mathrm{g}}(A)\right)\vec{b}$. Now, $\frac{\lambda}{1+\lambda} + \frac{1}{1+\lambda} = 1$ and $\frac{\lambda}{1+\lambda} + \frac{1}{1+\lambda}g = \frac{\lambda+1+(\lambda+1)(f-1)}{1+\lambda} = f$, so that $\left(\frac{\lambda}{1+\lambda}\mathscr{C}_1(A) + \frac{1}{1+\lambda}\mathscr{C}_{\mathrm{g}}(A)\right) = \mathscr{C}_{\mathrm{f}}(A)$ and the non square branch of the algorithm is also correct.

Algorithm 3 IPfCmv: In-place accumulated f-circulant matrix-vector multiplication

Input: $f \in \mathbb{D}$, $\vec{a}, \vec{b}, \vec{c} \in \mathbb{D}^m$.
Read-only: f.
Output: $\vec{c}$ += $\mathscr{C}_{\mathrm{f}}(\vec{a}) \cdot \vec{b}$.

if $m \leq$ Threshold **then** {Constant-time if Threshold $\in \mathcal{O}(1)$}
  return the quadratic in-place matrix-vector multiplication.
end if
if m is odd **then**
  repeat
    Let $\beta_1 \xleftarrow{\$} \mathbb{D}^*$, s.t. $\lambda_1 = (\beta_1 - 1) \in \mathbb{D}^*$; let $g = 1 - \beta_1^{-1}$;
  until g is a square
  Let $\beta_g = -f\beta_1$; $\lambda_g = f\beta_1$;
  Let $\check{a} = \vec{a}_{2..m}$, $\check{b} = \vec{b}_{2..m}$, $\hat{b} = \vec{b}_{1..m-1}$ and $\hat{c} = \vec{c}_{1..m-1}$;
  for $i = m$ **down-to** 2 **do** $b_i \star= \lambda_1$; b_i += $\lambda_g b_{i-1}$; **end for**
  $\hat{c}$ += $\mathscr{C}_1(\check{a}) \cdot \check{b}$; {IPfCmv with square and even dimension}
  for $i = 2$ **down-to** m **do** b_i -= $\lambda_g b_{i-1}$; b_i /= λ_1; **end for**
  for $i = m$ **down-to** 2 **do** $b_i \star= \beta_1$; b_i += $\beta_g b_{i-1}$; **end for**
  $\hat{c}$ += $\mathscr{C}_{\mathrm{g}}(\check{a}) \cdot \check{b}$; {IPfCmv with square and even dimension}
  for $i = 2$ **down-to** m **do** b_i -= $\beta_g b_{i-1}$; b_i /= β_1; **end for**
  $\vec{c}$ += $a_1 \cdot \vec{b}$; {Linear in-place accumulation}
  c_m += $f \cdot \check{a} \cdot \hat{b}$; {Linear in-place dotproduct}
else if f is not a square, or $f = 0$ **then**
  repeat
    Let $\lambda \xleftarrow{\$} \mathbb{D}^*$, s.t. $(\lambda + 1) \in \mathbb{D}^*$; Let $g = 1 + (\lambda + 1)(f - 1)$;
  until g is a square
  $\vec{c} \star= (1 + \lambda^{-1})$;
  $\vec{c}$ += $\mathscr{C}_1(a) \cdot \vec{b}$; {IPfCmv call}
  $\vec{c} \star= \lambda$;
  $\vec{c}$ += $\mathscr{C}_{\mathrm{g}}(A) \cdot \vec{b}$; {IPfCmv call with a square}
  $\vec{c} \star= (\lambda + 1)^{-1}$;
else
  Let $a_1 = \vec{a}_{1..m/2}$, $a_2 = \vec{a}_{m/2+1..m}$;
  Let $b_1 = \vec{b}_{1..m/2}$, $b_2 = \vec{b}_{m/2+1..m}$; {m is even}
  Let $c_1 = \vec{c}_{1..m/2}$, $c_2 = \vec{c}_{m/2+1..m}$;
  $\bar{c}_1 \leftarrow c_2 + c_1\sqrt{f}$ and $\bar{c}_2 \leftarrow c_2 - c_1\sqrt{f}$; {butterflies}
  $\bar{b}_1 \leftarrow b_2 + b_1\sqrt{f}$ and $\bar{b}_2 \leftarrow b_2 - b_1\sqrt{f}$; {butterflies}
  $\bar{a}_1 \leftarrow a_1 + a_2\sqrt{f}$ and $\bar{a}_2 \leftarrow a_1 - a_2\sqrt{f}$; {butterflies}
  $\bar{c}_1$ += $\mathscr{C}_{\sqrt{\mathrm{f}}}(\bar{a}_1) \cdot \bar{b}_1$; {Recursive IPfCmv call}
  $\bar{c}_2$ += $\mathscr{C}_{-\sqrt{\mathrm{f}}}(\bar{a}_2) \cdot \bar{b}_2$; {Recursive IPfCmv call}
  $a_1 \leftarrow (\bar{a}_1 + \bar{a}_2)/2$ and $a_2 \leftarrow (\bar{a}_1 - \bar{a}_2)/(2\sqrt{f})$;
  $b_1 \leftarrow (\bar{b}_1 - \bar{b}_2)/(2\sqrt{f})$ and $b_2 \leftarrow (\bar{b}_1 + \bar{b}_2)/2$;
  $c_1 \leftarrow (\bar{c}_1 - \bar{c}_2)/(2\sqrt{f})$ and $c_2 \leftarrow (\bar{c}_1 + \bar{c}_2)/2$;
end if
return $\vec{c}$.

Now, for the odd dimension branch, we have that:

$$\mathscr{C}_{\mathrm{f}}(\vec{a}) = \left[\begin{array}{ccccc} a_1 & a_2 & \dots & \dots & a_m \\ f{\cdot}a_m & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ f{\cdot}a_3 & \dots & f{\cdot}a_m & a_1 & a_2 \\ \hline f{\cdot}a_2 & \dots & \dots & f{\cdot}a_m & a_1 \end{array}\right].$$

Thus apart from the diagonal, $a_1{\cdot}\vec{b}$, and the last row, $f{\cdot}\check{a}{\cdot}\hat{b}$, the matrix-vector multiplication can be performed by a lower triangular and an upper triangular Toeplitz multiplications, $f{\cdot}L{\cdot}\hat{b} + U{\cdot}\check{b}$. But $[fL\backslash U] = \mathscr{C}_{\mathrm{f}}(\check{a})$. Therefore both can be decomposed as $fL = \lambda_g\,\mathscr{C}_1(\check{a}) + \beta_g\,\mathscr{C}_{\mathrm{g}}(\check{a})$ and $U = \lambda_1\,\mathscr{C}_1(\check{a}) + \beta_1\,\mathscr{C}_{\mathrm{g}}(\check{a})$ so that finally $f{\cdot}L{\cdot}\hat{b} + U{\cdot}\check{b} = \mathscr{C}_1(\check{a})(\lambda_1\check{b} + \lambda_g\hat{b}) + \mathscr{C}_{\mathrm{g}}(\check{a})(\beta_1\check{b} + \beta_g\hat{b})$.

For being in-place, each of the 3 vectors $\vec{a}$, $\vec{b}$, $\vec{c}$ need initial and final linear combinations of halves. Those can be performed sequentially with, say, two temporaries: this is $\mathcal{O}(n)$ classical butterfly swaps of the generic form $t = a_i r; u = b_i s; \bar{a}_i = t + u; \bar{b}_i = t - u; \bar{a}_i \star\!= v; \bar{b}_i \star\!= w$.

Finally, for the complexity bound, when called with a square Algorithm 3 uses at most 3 recursive calls and therefore requires $T(m) \leq 3T(m/2) + 9m$ operations, that is $\mathcal{O}(m^{\log_2(3)})$. These bounds also bound the odd cases as the smallest power of two larger than x is $\mathcal{O}(x)$. □

Theorem 4 does not work in particular in characteristic 2, as it divides by 2. We show how to deal with most of the characteristic 2 cases in Appendix B.

Remark 5. *In finite cardinality q such that $p - 1 = t2^s$ with t odd and $s \geq 2$, it could be insteresting to instead randomly select a non-square a until $g \equiv a^{2^s} \neq 1 \mod p$. This way $\lambda \equiv (g - f)(f - 1)^{-1}$ and the successive square roots from g will be squares for s recursive calls.*

This generalization of fast algorithms for f-circulant matrices allows us to build algorithms for accumulation with triangular Toeplitz matrices first, as sums of f-circulant in Algorithms 4 and 5, and then for any Toeplitz matrix, again as sums of triangular Toeplitz matrices in Algorithm 6.

Algorithm 4 In-place accumulated Upp. Triang. Toeplitz m-v. mult.

Input: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{D}^m$.
Output: $\vec{c}$ += $\mathcal{T}([\vec{0}, \vec{a}]) \cdot \vec{b}$.
1: **return** $\vec{c}$ += $\mathscr{C}_0(\vec{a}) \cdot \vec{b}$. {Algorithm 3}

Algorithm 5 In-place accumulated Low. Triang. Toeplitz m-v. mult.

Input: $\vec{a} \in \mathbb{D}^{m-1}$, $\vec{b}, \vec{c} \in \mathbb{D}^m$.
Output: $\vec{c}$ += $\mathcal{T}([\vec{a}, \vec{0}]) \cdot \vec{b}$.
1: Let $t = 0$;
2: $\vec{c}$ += $\mathscr{C}_2([t, \vec{a}]) \cdot \vec{b}$; {Algorithm 3}
3: **return** $\vec{c}$ -= $\mathscr{C}_1([t, \vec{a}]) \cdot \vec{b}$. {Algorithm 3}

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 Algorithm 4: this would provide instead a better algorithm than Algorithm 5 for lower triangular Toeplitz matrices, with a single call to (the transposed version of) Algorithm 3.*

Algorithm 6 In-place accumulated square Toeplitz m-v. mult.

Input: $\vec{a_1} \in \mathbb{D}^{m-1}$, $\vec{a_2}, \vec{b}, \vec{c} \in \mathbb{D}^m$,
Output: $\vec{c}$ += $\mathcal{T}([\vec{a_1}, \vec{a_2}]) \cdot \vec{b}$.

1: $\vec{a_2}$ -= $[0, \vec{a_1}]$;
2: $\vec{c}$ += $\mathscr{C}_0(\vec{a_2}) \cdot \vec{b}$; {Algorithm 3}
3: $\vec{a_2}$ += $[0, \vec{a_1}]$;
4: Let $t = 0$;
5: **return** $\vec{c}$ += $\mathscr{C}_1([t, \vec{a_1}]) \cdot \vec{b}$. {Algorithm 3}

Lemma 7. *Algorithms 4 to 6 are correct.*

Proof. Directly looking at the matrices values, we see that:

- $\mathcal{T}([\vec{a_1}, \vec{a_2}]) = \mathscr{C}_0(\vec{a_2} - [0, \vec{a_1}]) + \mathscr{C}_1([0, \vec{a_1}])$ [23, Eq. (2.6.2)].
- $\mathcal{T}([\vec{a}, \vec{0}]) = \mathscr{C}_2([\vec{a}, 0]) - \mathscr{C}_1([\vec{a}, 0])$.
- $\mathcal{T}([\vec{0}, \vec{a}]) = \mathscr{C}_0(\vec{a})$. □

From this, we give in Algorithm 7 an in-place rectangular Toeplitz matrix-vector multiplication.

Algorithm 7 In-place accumulated 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 6}
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 6}
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 6}
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** $\vec{c}$.

Proposition 8. *Algorithm 7 is correct and requires less than* $\mathcal{O}\left(\max\{m,n\} \min\{m,n\}^{\log_2(3)-1}\right)$ *operations.*

Proof. If $m>n$, then there are first $\lfloor m/n \rfloor$ square $n \times n$ calls to Algorithm 6. This is bounded by $\mathcal{O}((m/n)n^{\log^2(3)})$ 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 Algorithms 8 and 9.

Proposition 9. *Algorithm 8 is correct and requires less than* $\mathcal{O}(m^{\log_2(3)})$ *operations.*

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 the algorithm is correct. Now for the complexity bound, the cost function is $T(m) \leq 2T(m/2) + \mathcal{O}(m^{\log_2(3)}) = \mathcal{O}(m^{\log_2(3)})$. □

Algorithm 8 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.
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 7}
7: $b_1 \leftarrow \mathcal{T}([a_{(m-k+1)..m}, \vec{0}]) \cdot b_1$; {Recursive call}
8: **return** $\vec{b}$.

Algorithm 9 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 -= $\mathcal{T}_{\text{k,m-k}}(\vec{a}_{2..m}) \cdot b_2$; {Algorithm 7}
7: $b_1 \leftarrow \mathcal{T}([\vec{0}, \vec{a}_{1..k}])^{-1} \cdot b_1$; {Recursive call}
8: **return** $\vec{b}$.

Proposition 10. *Algorithm 9 is correct and requires less than $\mathcal{O}(m^{\log_2(3)})$ operations.*

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 $\bar{b}_1 = T_1^{-1}b_1 + HT_2\bar{b}_2 = T_1^{-1}b_1 - T_1^{-1}G\bar{b}_2 = T_1^{-1}\left(b_1 - G\bar{b}_2\right)$. Now for the complexity bound, the cost function is $T(m) \leq 2T(m/2) + \mathcal{O}(m^{\log_2(3)}) = \mathcal{O}(m^{\log_2(3)})$. □

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.

3 Over-writing the quotient

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. (1) and (2), 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. (1)

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. (2).

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

$$\begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} b_M & \dots & b_0 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_0 \\ & & & \ddots & \vdots \\ 0 & & & & b_M \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ \vdots \\ \vdots \\ q_{N-M} \end{bmatrix}. \tag{2}$$

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 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. (2) 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:

$$T = \begin{bmatrix} b_M & \dots & b_1 \\ & \ddots & \vdots \\ & & b_M \end{bmatrix} \text{ and } G = \begin{bmatrix} b_0 & & \\ \vdots & \ddots & \\ b_{M-1} & \dots & b_0 \end{bmatrix}. \tag{3}$$

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:

$$\begin{bmatrix} q_0 \\ \vdots \\ \vdots \\ q_{M-1} \end{bmatrix} = \left[\begin{array}{c:c} I_M & 0 \end{array}\right] \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} \tag{4}$$

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. (1) equations is just like multiplying the quotient by G and thus $R = A - BQ \mod X^M$ can be written as:

$$R = \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_{M-1} \end{bmatrix} - \left[\begin{array}{c:c} G & 0 \end{array}\right] \begin{bmatrix} T & G & & 0 \\ & \ddots & \ddots & \\ & & \ddots & G \\ 0 & & & T \end{bmatrix}^{-1} \begin{bmatrix} a_M \\ \vdots \\ \vdots \\ a_N \end{bmatrix} \tag{5}$$

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 that:

$$\begin{cases} H_1 = T^{-1} \\ H_{i-1}G + H_iT = 0, \quad i = 2..\mu \\ TH_i + GH_{i-1} = 0, \quad i = 2..\mu \end{cases} \tag{6}$$

Solving Eq. (6), we just get that $H_i = T^{-1}(-GT^{-1})^i = (-GT^{-1})^iT^{-1}$.

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. (5) and Theorem 11, we obtain now that:

$$R = \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_{M-1} \end{bmatrix} - GT^{-1} \left(\sum_{i=1}^{\mu} \left(-GT^{-1}\right)^{i-1} \vec{a_i} \right) = \sum_{i=0}^{\mu} \left(-GT^{-1}\right)^{i} \vec{a_i} \tag{7}$$

From Eq. (7) we thus can immediately deduce the following Algorithm 10 that uses only $\mathcal{O}(M)$ extra memory space in a Horner-like fashion of the polynomial in $(-GT^{-1})$ of Eq. (7). Note that this algorithm does not modify its input along its course: both $A(X)$ and $B(X)$ are for now read-only.

Algorithm 10 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 $n = N - M + 1$, $\mu = \lceil \frac{n}{M} \rceil$;
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 10 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 Theorem 11 and Eq. (7). 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 $\lceil\frac{N-M+1}{M}\rceil((\lambda_s+\lambda_t)\mathfrak{M}(M)+M) = \mathcal{O}(\frac{N}{M}\mathfrak{M}(M))$ 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 10 is in fact just the long division polynomial algorithm applied to sub-blocks of the polynomial of size M:*

- $\vec{t} = T^{-1}\cdot\vec{r}$*, Line 7, corresponds to computing the quotient of the current leading coefficients, of the dividend, by B;*
- $\vec{r} = (-G)\cdot\vec{t}$*, Line 8, corresponds to recovering the lower part of the multiplication of that current quotient by B;*
- $\vec{r}$ *+=* $\vec{a_i}$*, Line 9, 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 [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 as Theorem 12, that is less than $2\lambda\frac{N}{M}\mathfrak{M}(M)$ operations and $(1+s)M$ extra space.

4 Bands and Time-memory trade-off

Here we present a variant that enables to balance speed with storage. The algorithm still uses blocks, but requires only a small fraction of extra space. If the output space is M, then one obtains an algorithm using the space of one of its input, M space for the output plus only a fraction $m = M/\mu$ of extra space. The drawback is that our complexity bounds become $\mathcal{O}\big(\frac{N}{m}\frac{M}{m}\mathfrak{M}(m)\big)$.

Therefore this opens a whole range of algorithms in practice, from $m = \mathcal{O}(1)$, that is a constant extra space but quadratic complexity $\mathcal{O}(NM)$; to $\mu = \mathcal{O}(1)$, with $M(1+o(1))$ total space required (including the output of size M, that is, the extra space can be a very small fraction of the output, but not quite a constant), and a complexity that remains $\mathcal{O}\big(\frac{N}{M}\mathfrak{M}(M)\big)$.

Following [8, 26, 19, 18, 1, 20], we start by generalizing Theorem 11 to more Toeplitz bands:

Lemma 14. *For $\alpha-1$ matrices $G_2,\ldots,G_\alpha$ of dimension $m\times m$ and for T_1 an $m\times m$ upper triangular matrix, let:*

$$\begin{cases} H_1 = T_1^{-1} \\ \sum_{j=2}^{\min\{\alpha;i\}} H_{i-j+1}G_j + H_iT_1 = 0, \quad i = 2..\eta, \\ T_1H_i + \sum_{j=2}^{\min\{\alpha;i\}} H_{i-j+1}G_j = 0, \quad i = 2..\eta. \end{cases} \tag{8}$$

Then, the inverse of the $(m\eta)\times(m\eta)$ upper triangular Toeplitz matrix with α non-zero block bands is given in Eq. (9):

$$\begin{bmatrix} H_1 & \ldots & H_\eta \\ & \ddots & \ddots \\ & & H_1 \end{bmatrix} = \begin{bmatrix} T & G_2 & \ldots & G_\alpha & & 0 \\ & \ddots & & \ddots & & \\ & & & & & G_2 \\ & & & & \ddots & \vdots \\ & & & & & G_\alpha \\ 0 & & & & & T \end{bmatrix}^{-1} \tag{9}$$

Now we consider m consecutive coefficients of R in Eq. (5) for $m < M$. Let $\eta = \lceil \frac{N-M+1}{m} \rceil$ and $\alpha = \lceil \frac{M}{m} \rceil + 1$. If the consecutive coefficients start at index i_0, let $k = \lceil \frac{i_0-1}{m} \rceil$. Then now let $\vec{\rho}_k = [a_{i_0}, \ldots, a_{i_0+m-1}]$ and $[\vec{a_1}, \ldots, \vec{a_\eta}] = [a_M, \ldots, a_N, 0, \ldots, 0]$ again completed with zeroes if $N - M + 1$ is not a multiple of m.

Let the first m rows of $\begin{bmatrix} T & G \end{bmatrix}$ be $\begin{bmatrix} T_1 & G_2 & \ldots & G_\alpha & 0 \end{bmatrix} = \left[\begin{array}{c|c} I_m & 0 \end{array}\right] \begin{bmatrix} T & G \end{bmatrix}$. Finally, consider the m consecutive rows of $\left[\begin{array}{c|c} G & 0 \end{array}\right]$ starting at row i_0 as $\left[\begin{array}{ccc|c} F_0 & \ldots & F_k & 0 \end{array}\right]$ Then combining Eq. (5) and Theorem 14, we have the following:

Lemma 15. *For A and B polynomials of respective degrees N and M we let $R \equiv A \mod B$. Let $\vec{\rho}_k = [a_{i_0}, \ldots, a_{i_0+m-1}]$ and $[\vec{a_1}, \ldots, \vec{a_\eta}] = [a_M, \ldots, a_N, 0, \ldots, 0]$ and consider the Toeplitz blocks T and G of Eq. (3) together with the subsets of m rows:*
$\begin{bmatrix} T_1 & G_2 & \ldots & G_\alpha & 0 \end{bmatrix} = \left[\begin{array}{c|c} I_m & 0 \end{array}\right] \begin{bmatrix} T & G \end{bmatrix}$ *and:*
$\left[\begin{array}{ccc|c} F_0 & \ldots & F_k & 0 \end{array}\right] = \left[\begin{array}{c|c|c} 0_{i_0} & I_m & 0_{M-i_0-m} \end{array}\right] \cdot \left[\begin{array}{c|c} G & 0 \end{array}\right]$.

Finally let $[H_1, \ldots, H_\eta]$ be the components of the Toeplitz inverse as in Theorem 14, then: $\begin{bmatrix} r_{i_0} \\ \vdots \\ r_{i_0+m-1} \end{bmatrix} = \vec{\rho}_k - \sum_{j=0}^{k} F_j \left(\sum_{i=1}^{\eta-j} H_i \vec{a}_{i+j} \right)$.

This Theorem 15 together with the characterization of the inverse components of Eq. (8), now give us a way to compute this subset of the remainder via a generalized Horner-like process.

To save space and time the idea is to avoid the computation of the inverse altogether. For this, we recursively replace any computation of the form $H_i\vec{a_\ell}$, using Eq. (8) as follows: $H_i\vec{a_\ell} = H_iT_1T_1^{-1}\vec{a_\ell} = (-\sum_{j=2}^{\min\{\alpha;i\}} H_{i-j+1}G_j)T_1^{-1}\vec{a_\ell} = \sum_{j=2}^{\min\{\alpha;i\}} H_{i-j+1}(-G_jT_1^{-1}\vec{a_\ell})$. The core computations are now again of the form $G_jT_1^{-1}\vec{a_\ell}$ as in Algorithm 10 but α of them have to be performed at each step. In order to save the space to store them (which would be $m\alpha$, that is not smaller than $mM/m = M$), the idea is to modify instead the appropriate $\vec{a}_{i-j+1}$ parts of the input. The trick is that it is possible to do this up to $i = 1$, then store the overall sum into the final result space, and finally revert the whole computation (thus doubling the cost) to put back the input in its initial state. This is shown in Algorithm 11.

Theorem 16. *Algorithm 11 is correct and if Triangular Toeplitz solve requires $\lambda_s\mathfrak{M}(m)$ operations with $s{\cdot}m$ extra space and Toeplitz matrix-vector multiplication requires $\lambda_t\mathfrak{M}(m)$ operations with $t{\cdot}m$ extra space, then with $(2 + \max\{s;t\})m$ extra space, its arithmetic complexity is bounded by $\mathcal{O}\left(\frac{N}{m}\frac{M}{m}\mathfrak{M}(m)\right)$.*

Proof. Correctness directly comes from Theorems 14 and 15. Then, the complexity bound is twice $\eta(\lambda_s\mathfrak{M}(m)+(\alpha-1)\lambda_t\mathfrak{M}(m)+m)$ plus the middle computations $\sum_{k=0}^{\alpha-1}(k+1)(\lambda_s\mathfrak{M}(m)+\lambda_t\mathfrak{M}(m)+m)$. Its arithmetic complexity is thus bounded by: $(2\eta(\lambda_s+(\alpha-1)\lambda_t)+\alpha\frac{\alpha+1}{2}(\lambda_s+\lambda_t))\mathfrak{M}(m)+\left(2\eta+\alpha\frac{\alpha+1}{2}\right)m$. With $\eta = \lceil \frac{N-M+1}{m} \rceil$ and $\alpha = \lceil \frac{M}{m} \rceil + 1$ we obtain the claimed bound.

Now, if the operations are performed sequentially, then apart from the used input, $\vec{a_i}$, two size m memory slots are required, $\vec{r}$ and $\vec{t}$, and either the extra space of the Toeplitz solving or that of the multiplication. This is $(2 + \max\{s;t\})m$ extra space. ∎

Remark 17. *Again, we see that $\vec{r} = T_1^{-1} \cdot \vec{a_i}$ is computing the current quotient $\vec{r} = Quo(\vec{a_i}(X)X^{m-1}, B_1(X))$ and that the loop $(\vec{t} = G_j \cdot \vec{r})_{j=2..\alpha}$ is the multiplication by the whole divisor, $B(X)\vec{r}(X)$.*

5 In-place modular remainder

We now derive algorithms that uses 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. The idea is to combine Sections 2 to 4.

We can now present in Algorithm 12, a fully in-place remainder where only $B(X)$ is modified but restored: this variant replaces only Lines 7 to 9 of Algorithm 10. From this we obtain, Algorithm 13 which directly update the dividend over-place, while also remaining fully in-place.

Algorithm 11 BRem: block remainder using & restoring its LHS

Input: m, $\vec{a} \in \mathbb{D}^N$, $\vec{b} \in \mathbb{D}^M$, $\eta = \lceil \frac{N-M+1}{m} \rceil$, $\alpha = \lceil \frac{M}{m} \rceil + 1$;.
Read-only: $\vec{b}$.
Output: $R = [R_0, \ldots, R_{\alpha-1}]$, with $R_k = \vec{\rho}_k - \sum_{j=0}^{k} F_j \left(\sum_{i=1}^{\eta-j} H_i \vec{a}_{i+j} \right)$ and the H_i as defined in Theorem 14.

Let η, α and T, G, T_1, $G_2, \ldots, G_\alpha$ as in Theorem 15;
Let $[\vec{\rho}_0, .., \vec{\rho}_{\alpha-1}] = [\vec{a}_{0..(M-1)}, \vec{0}]$ and $[\vec{a_1}, .., \vec{a_\eta}] = [\vec{a}_{M..N}, \vec{0}]$
for $i = \eta$ **down-to** 1 **do** {Propagate η times Eq. (8) to $\vec{a}$}
  $\vec{r} = T_1^{-1} \cdot \vec{a_i}$; {triangular Toeplitz solve}
  for $j = 2$ **to** $\min\{\alpha; i\}$ **do**
    $\vec{t} = G_j \cdot \vec{r}$; {Toeplitz matrix-vector multiplication}
    $\vec{a}_{i-j+1}$ -= $\vec{t}$;
  end for
end for
for $k = 0$ **to** $\alpha - 1$ **do**
  Let $i_0 = k \cdot m$;
  Let $\left[F_0 \ldots F_k \, 0 \right] = \left[0_{i_0} \, I_m \, 0_{M-i_0-m} \right] \cdot \left[G \, 0 \right]$;
  $R_k = \vec{\rho}_k$;
  for $j = 0$ **to** k **do** {Now this is just $\vec{\rho}_k - \sum_{j=0}^{k} F_j T_1^{-1} \vec{a_j}$}
    $\vec{r} = T_1^{-1} \cdot \vec{a_j}$; {triangular Toeplitz solve}
    $\vec{t} = F_j \cdot \vec{r}$; {Toeplitz matrix-vector multiplication}
    R_k -= $\vec{t}$;
  end for
end for
for $i = 1$ **to** η **do** {Unroll η times Eq. (8) to $\vec{a}$}
  $\vec{r} = T_1^{-1} \cdot \vec{a_i}$; {triangular Toeplitz solve}
  for $j = 2$ **to** $\min\{\alpha; i\}$ **do**
    $\vec{t} = G_j \cdot \vec{r}$; {Toeplitz matrix-vector multiplication}
    $\vec{a}_{i-j+1}$ += $\vec{t}$;
  end for
end for
return $R = [R_0, \ldots, R_{\alpha-1}]$.

Algorithm 12 IPER (R,A,B): in-place 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 $M - 1$.

if $M > N$ **then return** A.
Let n, μ, $[\vec{a_0}, \ldots, \vec{a_\mu}]$, T, G as in Algorithm 10;
$\vec{r} = \vec{a_\mu}$; {$\vec{r}$ in-place of the result}
for $i = \mu - 1$ **down-to** 0 **do**
  $\vec{r} \leftarrow T^{-1} \cdot \vec{r}$; {Algorithm 9}
  $\vec{r} \leftarrow (-G) \cdot \vec{r}$; {Algorithm 8}
  $\vec{r}$ += $\vec{a_i}$;
end for
return $R = \sum_{i=0}^{M-1} r_i X^i$.

Algorithm 13 OPER (A,B): Over-place 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 Algorithm 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 9}
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: $\vec{a}_{\mu-1}$ += $(-G_1) \cdot \vec{a_\mu}$ {Algorithm 7}
9: **end if**
10: **for** $i = \mu - 1$ **down-to** 1 **do**
11: $\vec{a_i} \leftarrow T^{-1} \cdot \vec{a_i}$; {Algorithm 9}
12: $\vec{a}_{i-1}$ += $(-G) \cdot \vec{a_i}$; {Algorithm 6}
13: **end for**
14: **return** $A = \vec{a_0}$.

Theorem 18. *Algorithms 12 and 13 are correct, in-place and require less than* $\mathcal{O}(NM^{\log_2(3)-1})$ *operations.*

Proof. Algorithm 12 calls $\mu{=}\mathcal{O}(N/M)$ times Algorithms 8 and 9, each call requiring less than $\mathcal{O}(M^{\log_2(3)})$ operations by Theorems 9 and 10. This dominates the cost of Algorithm 13: its Lines 5 to 8 are the over-place first iteration of Algorithm 12, for $s < M$. □

Similarly, we replace the blocks of Lines 4 to 7 and of Lines 21 to 24 as well as the block of Lines 15 to 17 of Algorithm 11, to obtain Algorithm 14. This again allows us to have some latitude in practice for different relative ratios between the degrees of the dividend and the divisor.

Theorem 19. *Algorithm 14 is correct, in-place and requires less than* $\mathcal{O}\big(N \frac{M}{m^{2-\log_2(3)}}\big)$ *operations.*

Proof. The algorithm calls $\eta\alpha + \alpha^2 = \mathcal{O}\big(\frac{N}{m}\frac{M}{m}\big)$ times Algorithm 7, each call requiring less than $\mathcal{O}(m^{\log_2 3})$ operations by Theorem 8. It also calls $\eta = \mathcal{O}(\frac{N}{m})$ times Algorithms 8 and 9, each call requiring also less than $\mathcal{O}(m^{\log_2(3)})$ operations by Theorems 9 and 10. □

6 Conclusion

We have presented novel algorithms computing f-circulant and Toeplitz matrix-vector multiplications in-place. This allows us to derive novel algorithms for accumulated or over-place Toeplitz multiplication or 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 and over-place algorithms computing only the remainder of the polynomial euclidean division.

Further work include finding ways to (1) further reduce the sub-quadratic complexity bounds for in-place versions (for instance Appendix B sketches the use of k-adic transforms instead of diadic transform that can further reduce the complexity exponent when the coefficient domain allows it); (2) deal with the field with 3 elements, there maybe some linearization of the extension with 9 elements could help; (3) avoid dividing by 2, in order to handle the even characteristic case. The latter can be achieved, in most of the characteristic 2 cases, via k-adic transforms. This is sketched in Appendix B.

Unfortunately there remains some cases where we still do not know sub-quadratic in-place algorithms. This is $\mathbb{F}_2$, $\mathbb{F}_3$, and the even characteristic finite fields with cyclic group cardinality a Mersenne prime (as of today, 51 are known).

Maybe some other fast transform that do not require extensions could be of help, provided that one can make them in-place [22, 2].

Algorithm 14 In-place euclidean block remainder

Input: m, $\vec{a} \in \mathbb{D}^N$, $\vec{b} \in \mathbb{D}^M$, $\eta = \left\lceil \frac{N-M+1}{m} \right\rceil$, $\alpha = \left\lceil \frac{M}{m} \right\rceil + 1$;.

Output: $R = [R_0, \ldots, R_{\alpha-1}]$, with $R_k = \vec{\rho}_k - \sum_{j=0}^{k} F_j \left(\sum_{i=1}^{\eta-j} H_i \vec{a}_{i+j} \right)$ and the H_i as defined in Theorem 14.

1: Let η, α $[\vec{\rho}_0, \ldots, \vec{\rho}_{\alpha-1}]$, T, G and $\begin{bmatrix} T_1 & G_2 & \ldots & G_\alpha & 0 \end{bmatrix} = \left[\begin{array}{c|c} I_m & 0 \end{array}\right] \begin{bmatrix} T & G \end{bmatrix}$ as in Algorithm 11;
2: **for** $i = \eta$ **down-to** 1 **do** {Propagate η times Eq. (8) to $\vec{a}$}
3: $\quad \vec{a_i} = T_1^{-1} \cdot \vec{a_i}$; {Algorithm 9}
4: $\quad$ **for** $j = 2$ **to** $\min\{\alpha; i\}$ **do**
5: $\quad\quad \vec{a}_{i-j+1}$ -= $G_j \cdot \vec{a_i}$; {Algorithm 7}
6: $\quad$ **end for**
7: **end for**
8: **for** $k = 0$ **to** $\alpha - 1$ **do**
9: $\quad$ Let $i_0 = k \cdot m$ and $\begin{bmatrix} F_0 \ldots F_k & 0 \end{bmatrix} = \begin{bmatrix} 0_{i_0} & I_m & 0_{M-i_0-m} \end{bmatrix} \cdot \begin{bmatrix} G & 0 \end{bmatrix}$;
10: $\quad R_k = \vec{\rho}_k$;
11: $\quad$ **for** $j = 0$ **to** k **do**
12: $\quad\quad R_k$ -= $F_j \cdot \vec{a_j}$; {Algorithm 7}
13: $\quad$ **end for**
14: **end for**
15: **for** $i = 1$ **to** η **do** {Unroll η times Eq. (8) to $\vec{a}$}
16: $\quad$ **for** $j = 2$ **to** $\min\{\alpha; i\}$ **do**
17: $\quad\quad \vec{a}_{i-j+1}$ += $G_j \cdot \vec{a_i}$; {Algorithm 7}
18: $\quad$ **end for**
19: $\quad \vec{a_i} = T_1 \cdot \vec{a_i}$; {Transpose of Algorithm 8}
20: **end for**
21: **return** $R = [R_0, \ldots, R_{\alpha-1}]$.

References

[1] Skander Belhaj, Marwa Dridi, and Ahmed Salam. A fast algorithm for solving banded Toeplitz systems. *Computers & Mathematics with Applications*, 70(12):2958–2967, 2015. URL: https://www.sciencedirect.com/science/article/pii/S0898122115004939, doi:https://doi.org/10.1016/j.camwa.2015.10.010.

[2] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ecfft) part i: Low-degree extension in time o(n log n) over all finite fields. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA'23, pages 700–737, 2023. doi:10.1137/1.9781611977554.ch30.

[3] 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.

[4] 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.

[5] A. Bostan, G. Lecerf, and É. Schost. Tellegen's Principle into Practice. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 37–44, New York, NY, USA, 2003. ACM. doi:10.1145/860854.860870.

[6] Brice Boyer, Jean-Guillaume Dumas, Clément Pernet, and Wei Zhou. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. In John P. May, editor, *ISSAC'2009, Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, Seoul, Korea*, pages 135–143, New York, July 2009. ACM Press. URL: http://hal.archives-ouvertes.fr/hal-00163141.

[7] 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.

[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. URL: https://www.sciencedirect.com/science/article/pii/S0747717121000705, doi:https://doi.org/10.1016/j.jsc.2021.10.002.

[10] Carmine Di Fiore, Francesco Tudisco, and Paolo Zellini. Lower triangular toeplitz–ramanujan systems whose solution yields the bernoulli numbers. *Linear Algebra and its Applications*, 496:510–526, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0024379516000987, doi:10.1016/j.laa.2016.02.007.

[11] Jean-Guillaume Dumas, Clément Pernet, and Alexandre Sedoglavic. Some fast algorithms multiplying a matrix by its adjoint. *Journal of Symbolic Computation*, 115:285–315, March 2023. URL: https://hal.archives-ouvertes.fr/hal-03095393, doi:10.1016/j.jsc.2022.08.009.

[12] 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.

[13] 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.

[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.

[15] 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.

[16] David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time O(nlog n). *J. 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] Fu-Rong Lin. An explicit formula for the inverse of band triangular Toeplitz matrix. *Linear Algebra and its Applications*, 428(2):520–534, 2008. Special Issue devoted to the Second International Conference on Structured Matrices. URL: https://www.sciencedirect.com/science/article/pii/S0024379507002145, doi:10.1016/j.laa.2007.05.007.

[19] Fu-Rong Lin, Wai-Ki Ching, and Michael K. Ng. Fast inversion of triangular Toeplitz matrices. *Theor. Comput. Sci.*, 315(2-3):511–523, 2004. doi:10.1016/j.tcs.2004.01.005.

[20] Alexander N. Malyshev and Miloud Sadkane. Fast solution of unsymmetric banded Toeplitz systems by means of spectral factorizations and woodbury's formula. *Numerical Linear Algebra with Applications*, 21(1):13–23, 2014. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.1853, doi:10.1002/nla.1853.

[21] 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.

[22] Henri J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, 1980. doi:10.1109/TASSP.1980.1163372.

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

[24] Daniel S. Roche. Space-and time-efficient polynomial multiplication. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, pages 295–302. ACM, 2009.

[25] Andreas Rosowski. On fast computation of a circulant matrix-vector product. 2021. arXiv:2103.02605.

[26] William F. Trench. Explicit inversion formulas for toeplitz band matrices. *SIAM Journal on Algebraic Discrete Methods*, 6(4):546–554, 1985. doi:10.1137/0606054.

[27] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013.

A f-circulant/polynomial isomorphism

It is well known that the algebra of f-circulant $n\times n$ matrices is isomorphic to the algebra of polynomials modulo $X^n - f$, see, e.g., [23, Theorem 2.6.1]. More precisely, define the reverse of a vector of dimension n as $\overleftarrow{a} = \mathrm{rev}\,(\vec{a}) = a_{n..1}$ and identify polynomials to their vector of coefficients. Then we have that

$$P(X)Q(X) \mod (X^n - f) = \mathrm{rev}\,(\mathscr{C}_{\mathrm{f}}(P)\cdot \mathrm{rev}\,(Q)) \tag{10}$$

Let $Y = X^{n/2}$, then, when f is a square, the recursive branch of Algorithm 3 is a recursive application of the Chinese Remaindering theorem with $X^n - f = (X^{n/2}+\sqrt{f})(X^{n/2}-\sqrt{f})$, or $Y^2 - f = (Y+\sqrt{f})(Y-\sqrt{f})$. Indeed both factors are coprime and satisfy

$$\frac{1}{2\sqrt{f}}(Y+\sqrt{f}) - \frac{1}{2\sqrt{f}}(Y-\sqrt{f}) = 1. \tag{11}$$

Therefore Let $C_+ = (C + PQ \mod (Y+\sqrt{f}))$ and $C_- = (C + PQ \mod (Y-\sqrt{f}))$, then $C + PQ \mod (Y^2 - f) = \frac{1}{2\sqrt{f}}(Y+\sqrt{f})C_- - \frac{1}{2\sqrt{f}}(Y-\sqrt{f})C_+ = \frac{1}{2}(C_+ + C_-) + Y\frac{1}{2\sqrt{f}}(C_- - C_+)$.

This shows for instance the correctness of Algorithm 3, by seeing that $C+PQ = \mathrm{rev}\,(\mathrm{rev}\,(C) + \mathscr{C}_{\mathrm{f}}(P)\cdot\mathrm{rev}\,(Q))$:

- Let $P = [\vec{a}_1, \vec{a}_2] = \vec{a}_1 + Y\vec{a}_2$, $Q = [\vec{b}_1, \vec{b}_2] = \vec{b}_1 + Y\vec{b}_2$, $C = [\vec{c}_1, \vec{c}_2] = \vec{c}_1 + Y\vec{c}_2$;
- Then $\mathrm{rev}\,(Q) = \overleftarrow{b} = [\overleftarrow{b}_2, \overleftarrow{b}_1] = \overleftarrow{b}_2 + Y\overleftarrow{b}_1$, and $\mathrm{rev}\,(C) = \overleftarrow{c} = [\overleftarrow{c}_2, \overleftarrow{c}_1] = \overleftarrow{c}_2 + Y\overleftarrow{c}_1$;
- Thus:

$$\begin{aligned}
\bar{c}_1 &= \overleftarrow{c}_2 + \sqrt{f}\overleftarrow{c}_1 \equiv \overleftarrow{c} \mod (Y-\sqrt{f})\\
\bar{c}_2 &= \overleftarrow{c}_2 - \sqrt{f}\overleftarrow{c}_1 \equiv \overleftarrow{c} \mod (Y+\sqrt{f})\\
\bar{b}_1 &= \overleftarrow{b}_2 + \sqrt{f}\overleftarrow{b}_1 \equiv \overleftarrow{b} \mod (Y-\sqrt{f})\\
\bar{b}_2 &= \overleftarrow{b}_2 - \sqrt{f}\overleftarrow{b}_1 \equiv \overleftarrow{b} \mod (Y+\sqrt{f})\\
\bar{a}_1 &= \vec{a}_1 + \sqrt{f}\vec{a}_2 \equiv \vec{a} \mod (Y-\sqrt{f})\\
\bar{a}_2 &= \vec{a}_1 - \sqrt{f}\vec{a}_2 \equiv \vec{a} \mod (Y+\sqrt{f})
\end{aligned}$$

- Then:

$$\overleftarrow{C}_- = (\overleftarrow{c} + \mathscr{C}_{\mathrm{f}}(P)\cdot\overleftarrow{b}) \mod (Y-\sqrt{f}) = \bar{c}_1 + \mathscr{C}_{\sqrt{\mathrm{f}}}(\bar{a}_1)\cdot\bar{b}_1$$
$$\overleftarrow{C}_+ = (\overleftarrow{c} + \mathscr{C}_{\mathrm{f}}(P)\cdot\overleftarrow{b}) \mod (Y+\sqrt{f}) = \bar{c}_2 + \mathscr{C}_{-\sqrt{\mathrm{f}}}(\bar{a}_2)\cdot\bar{b}_2$$

- So that finally, $C + PQ = \mathrm{rev}\left(\frac{1}{2}(\overleftarrow{C}_+ + \overleftarrow{C}_-) + Y\frac{1}{2\sqrt{f}}(\overleftarrow{C}_- - \overleftarrow{C}_+)\right)$, i.e., $C + PQ = \frac{1}{2\sqrt{f}}(\vec{C}_- - \vec{C}_+) + Y\frac{1}{2}(\vec{C}_+ + \vec{C}_-)$.

B Finite field of Characteristic two

With the isomorphism of Appendix A we can find an equivalent of Algorithm 3 for most of the finite fields of characteristic two.

Denote $[x]g \stackrel{def}{=} g^x$. First, for a finite field of cardinality $q = 2^m$, let $q - 1 = kt$, with $k \leq t$. Then, consider a k-th power, f, in $\mathbb{F}_q^*$: that is, for a generator g of this cyclic group, let $f = g^{ik} = [ik]g$ for some i. This means that the $f_j = [i + jt]g$, for $j = 1..k$ are all the k-th roots of f and they all belong to $\mathbb{F}_q^*$, as $(i + jt)k \equiv ik \mod (q-1)$ and $q - 1 = kt$.

Thus, in $\mathbb{F}_q$, we have that $Y^k - f = \prod_{j=1}^{k}(Y - f_j)$. Further, $X^{uk} - f = \prod_{j=1}^{k}(X^u - f_j)$ (again, let $Y = X^u$). All the $(X^u - f_j)$ are co-prime, since all the k-th roots are distinct (they have distinct exponent with respect to the generator g, between 0 and $q - 1$).

Then, computing the extended gcd between $Y - f_j$ and $(Y^k - f)/(Y - f_j)$, gives Bezout coefficients U_j and v_j, s.t. $U_j(Y - f_j) + v_j(Y^k - f)/(Y - f_j) = 1$, with $\deg v_j < \deg (Y - f_j) = 1$. This means that $v_j \in \mathbb{F}_q^*$ (and the Bezout relation remains true if $Y = X^u$, for any u). Therefore, if $n = ku$, the Chinese remaindering Lagrange formula now writes: $c \mod (X^n - f) = \prod_{j=1}^{k}(c \mod (X^u - f_j))\, v_j \frac{X^n - f}{X^u - f_j}$.

If the number of invertibles in $\mathbb{F}_q$, $2^m - 1$ is not a Mersenne prime, then it can be written as $2^m - 1 = kt$ with $k \geq t > 1$ (that is $k \geq t \geq 3$) and we can form a k-adic recursion for circulant matrices. This also gives rise to a sub-quadratic algorithm: since $T(n) \leq (1 + 2(k-1))T(n/k) + O(n)$, then $T(n) = \mathcal{O}\left(n^{\log_k(2k-1)}\right) = \mathcal{O}\left(n^{1+\log_k(2-1/k)}\right)$ (resp. $T(n) \leq kT(n/k) + O(n)$, then $T(n) = \mathcal{O}(n\log(n))$ if the field contains a $k^{\lfloor \log_k(n) \rfloor}$-th root of unity). Further, if $k > 1$, then 1 is not the only k-th power (there are $t \geq 3$ of them), and we can built the analogous of the full Algorithm 3. Indeed, by letting $\lambda = (b^k - f)/(f - 1)$, for another k-th power b^k, we can always compute $\mathscr{C}_{\mathrm{f}} = \frac{\lambda}{\lambda+1}\mathscr{C}_1 + \frac{1}{\lambda+1}\mathscr{C}_{\mathrm{b^k}}$, and the accumulation $c = c + \mathscr{C}_{\mathrm{f}}$ can be computed by $c = \frac{\lambda}{\lambda+1}\left(\frac{1}{\lambda}\left((\lambda+1)c + \mathscr{C}_{\mathrm{b^k}}\right) + \mathscr{C}_1\right)$. The conditions are that λ exists, thus $f \neq 1$; λ is invertible, thus f is not a k-th power; and $\lambda + 1$ is invertible, thus $b^k \neq 1$. We have proven:

Proposition 20. *For $A(X)$ and $B(X)$ polynomials of respective degrees N and M over any finite field with even characteristic $q = 2^m$, such that $2^m - 1 = kt$ is composite, there exist an in-place algorithm requiring less than $\mathcal{O}(NM^{\log_k(2-1/k)})$ arithmetic operations (resp. $\mathcal{O}(N\log^2 M)$ if the field contains a $k^{\lfloor \log_k(n) \rfloor}$-th root of unity).*

When the cyclic group is a Mersenne prime, then unfortunately the only power with all its roots in the field is 1 and we thus cannot combine recursive calls. This prevents us for now to have fast algorithms in $\mathbb{F}_2$, $\mathbb{F}_4$, $\mathbb{F}_8$, $\mathbb{F}_{32}$, $\mathbb{F}_{128}$, $\mathbb{F}_{8192}$, ..., (as of today, only 51 Mersenne primes are known*, and we do not know if there are an infinite number of those or not).

For instance, we show how this can be acheived when the cyclic group cardinality is divisble by 3 in Algorithm 15 (this includes, e.g., $\mathbb{F}_{16}$, which contains 5 distinct cubes and 15 distinct cubic roots). We here give only the part of the algorithm where the dimension is a multiple of 3 and where f is a cube. Other dimensions and non-cubes would have to be dealt with similar techniques as in Algorithm 3.

*https://www.mersenne.org/primes

Algorithm 15 Tri-adic in-place convolution in characteristic 2

Input: $A(X)$, $B(X)$, $C(X)$ of degree $m-1$, with $m \equiv 0 \mod 3$.
Input: (j_0, j_1, j_2), s.t. $(Y-j_0)(Y-j_1)(Y-j_2) = Y^3 - f$.
Output: $C(X) \mathrel{+}= A(X)B(X) \mod (X^m - f) \mod 2$.

1: Let $a_0 = \vec{a}_{1..m/3}$, $a_1 = \vec{a}_{m/3+1..2m/3}$, $a_2 = \vec{a}_{2m/3+1..m}$;
2: Let $b_0 = \vec{b}_{1..m/3}$, $b_1 = \vec{b}_{m/3+1..2m/3}$, $b_2 = \vec{b}_{2m/3+1..m}$;
3: Let $c_0 = \vec{c}_{1..m/3}$, $c_1 = \vec{c}_{m/3+1..2m/3}$, $c_2 = \vec{c}_{2m/3+1..m}$;
4: $\bar{a}_0 \leftarrow a_0 + a_1 j_0 + a_2 j_0^2$; $\bar{a}_1 \leftarrow a_0 + a_1 j_1 + a_2 j_1^2$; $\bar{a}_2 \leftarrow a_0 + a_1 j_2 + a_2 j_2^2$;
5: $\bar{b}_0 \leftarrow b_0 + b_1 j_0 + b_2 j_0^2$; $\bar{b}_1 \leftarrow b_0 + b_1 j_1 + b_2 j_1^2$; $\bar{b}_2 \leftarrow b_0 + b_1 j_2 + b_2 j_2^2$;
6: $\bar{c}_0 \leftarrow c_0 + c_1 j_0 + c_2 j_0^2$; $\bar{c}_1 \leftarrow c_0 + c_1 j_1 + c_2 j_1^2$; $\bar{c}_2 \leftarrow c_0 + c_1 j_2 + c_2 j_2^2$;
7: $\bar{c}_0(X) \mathrel{+}= \bar{a}_0(X) \cdot \bar{b}_0(X) \mod (X^{m/3} - j_0)$; {Recursive call}
8: $\bar{c}_1(X) \mathrel{+}= \bar{a}_1(X) \cdot \bar{b}_1(X) \mod (X^{m/3} - j_1)$; {Recursive call}
9: $\bar{c}_2(X) \mathrel{+}= \bar{a}_2(X) \cdot \bar{b}_2(X) \mod (X^{m/3} - j_2)$; {Recursive call}
10: $\begin{cases} a_0 \leftarrow \bar{a}_0 + \bar{a}_1 + \bar{a}_2; & a_1 \leftarrow \bar{a}_0 j_0^{-1} + \bar{a}_1 j_1^{-1} + \bar{a}_2 j_2^{-1}; \\ & a_2 \leftarrow \bar{a}_0 j_0^{-2} + \bar{a}_1 j_1^{-2} + \bar{a}_2 j_2^{-2}; \end{cases}$
11: $\begin{cases} b_0 \leftarrow \bar{b}_0 + \bar{b}_1 + \bar{b}_2; & b_1 \leftarrow \bar{b}_0 j_0^{-1} + \bar{b}_1 j_1^{-1} + \bar{b}_2 j_2^{-1}; \\ & b_2 \leftarrow \bar{b}_0 j_0^{-2} + \bar{b}_1 j_1^{-2} + \bar{b}_2 j_2^{-2}; \end{cases}$
12: $\begin{cases} c_0 \leftarrow \bar{c}_0 + \bar{c}_1 + \bar{c}_2; & c_1 \leftarrow \bar{c}_0 j_0^{-1} + \bar{c}_1 j_1^{-1} + \bar{c}_2 j_2^{-1}; \\ & c_2 \leftarrow \bar{c}_0 j_0^{-2} + \bar{c}_1 j_1^{-2} + \bar{c}_2 j_2^{-2}; \end{cases}$
13: **return** $\vec{c}$.