A Modular Greatest Common Divisor Algorithm for Matrix Polynomials

Howard Cheng[*] *George Labahn*

Symbolic Computation Group,

Department of Computer Science,

University of Waterloo,

Waterloo, Canada

`{hchcheng,glabahn}@scg.math.uwaterloo.ca`

Abstract

In this paper we give a modular algorithm to compute one-sided greatest common divisors for matrix polynomials, improving on the fraction-free algorithm by Beckermann and Labahn. We define lucky homomorphisms for the modular algorithm and give bounds on the coefficients in the results computed. In addition, the greatest common left (right) divisor computed by our algorithm is in column (row) reduced form. For computing a greatest common left divisor of two matrix polynomials of dimensions $m \times n_1$ and $m \times n_2$ having degree N in which all the coefficients of the entries have sizes bounded by K, the bit complexity of our algorithm is $O(n(m^2N)^3K^2)$ where $n = n_1 + n_2$. This is a significant improvement over the fraction-free algorithm. Our algorithm also solves the extended one-sided GCD problem, and can be used to transform any matrix polynomial into column reduced form.

1 Introduction

The computation of the greatest common divisor (GCD) of two scalar polynomials is one of the central operations used in computer algebra systems.

[*]Supported by an Ontario Graduate Student Scholarship.

The primary methods for computing scalar GCDs include the classic Euclidean algorithm [11], the subresultant GCD algorithm [7, 10], the modular GCD algorithm [7], and the heuristic algorithm [9]. One-sided GCDs also exist for noncommutative domains, including Ore domains such as the rings of differential or difference operators and domain of matrix polynomials. Algorithms corresponding to the subresultant and modular GCD algorithms for Ore domains have been given by Li [16, 17].

In this paper we consider the problem of computing a one-sided GCD of two matrix polynomials. The one-sided GCD problem for matrix polynomials arises in many applications particularly in the field of control theory and linear systems theory. These include determining irreducible matrix-fraction descriptions and minimal realizations for transfer functions [6, 13] and in computing matrix Padé fractions [2, 8, 14].

The classic algorithms for computing a one-sided GCD of two matrix polynomials $\mathbf{A}(z)$ and $\mathbf{B}(z)$ involve the computation of matrix normal forms. For example, the left GCD (GCLD) can be computed by finding the column Hermite or Popov normal form of the rectangular matrix polynomial $[\mathbf{A}(z)\ \mathbf{B}(z)]$ [8, 13]. Other approaches include performing Gaussian elimination on "generalized Sylvester matrices" of coefficients [6, 13], which take advantage of the structure of these matrices in order to make the elimination more efficient. The main difficulty with these approaches is the same as that encountered using the Euclidean algorithm, that of coefficient growth in the intermediate expressions. Indeed, this is much more severe in the case of matrix polynomials and must be controlled by any practical algorithm. The first algorithm for computing one-sided GCDs for matrix polynomials which controls coefficient growth was given by Beckermann and Labahn [4]. For computing a GCLD of two matrix polynomials of degree N, having dimensions $m \times n_1$ and $m \times n_2$ with the size of the coefficients bounded by K, they gave a fraction-free algorithm having a worst case bit complexity of $O(n(m^2N)^4K^2)$ where $n = n_1 + n_2$. The algorithm, which includes the scalar subresultant algorithm as a special case [3], predicts common factors by examining the linear systems of equations associated with the GCD problem.

In this paper we present a new modular algorithm for computing one-sided GCDs for matrix polynomials. As in most modular algorithms, there are three major issues that must be addressed: the problem of "unlucky" homomorphisms (i.e. when to discard computed results), the number of images required for the reconstruction of the result via Chinese remaindering, and the normalization of the result to ensure a unique answer. Although these issues are well-understood in the case of scalar polynomials, they are nontrivial for matrix polynomials. For example, one-sided GCDs are only unique up to multiplication by a unimodular matrix polynomial and can have varying row or column degrees. The definitions of lucky homomorphisms and the appropriate normalization are not obvious. We overcome these difficulties by reducing the GCD problem to a linear algebra problem and finding a matrix GCD of minimal degree (in our case column degree). The worst case bit complexity of our algorithm is $O(n(m^2N)^3K^2)$, a significant improvement over the fraction-free algorithm. Furthermore, our algorithm also solves the extended one-sided GCD problem, and bounds on the coefficients in the results can be obtained. Finally, our algorithm can also be used to transform any matrix polynomial into column reduced form.

The remainder of this paper is organized as follows. In Section 2, we review the relevant definitions and properties of matrix polynomials and the extended one-sided GCD problem. Section 3 discusses the issues in designing a modular algorithm for matrix polynomials. An algorithm for computing a one-sided GCD over $\mathbb{Z}_p$ is given in Section 4. The definitions of lucky homomorphisms and normalization are discussed in Section 5. In Section 6 we discuss the number of homomorphic images required, and the complete modular algorithm and its complexity are discussed in Section 7. Some experimental results are presented in Section 8. Concluding remarks and future research directions are discussed in the closing section.

2 Preliminaries

Let $\mathbb{D}$ be an integral domain with quotient field $\mathbb{Q}$ and $\mathbb{D}[z]^{m\times n}$, $\mathbb{Q}[z]^{m\times n}$ the sets of $m \times n$ matrix polynomials over $\mathbb{D}$ and $\mathbb{Q}$, respectively. Let $\mathbf{A}(z) \in \mathbb{Q}[z]^{m\times n_1}$ and $\mathbf{B}(z) \in \mathbb{Q}[z]^{m\times n_2}$. Then $\mathbf{G}(z) \in \mathbb{Q}[z]^{m\times m}$ is a *Greatest Common Left Divisor* (GCLD) of $\mathbf{A}(z)$ and $\mathbf{B}(z)$ if

(a) $\mathbf{G}(z)$ is a left divisor of $\mathbf{A}(z)$ and $\mathbf{B}(z)$,

(b) every other left divisor of $\mathbf{A}(z)$ and $\mathbf{B}(z)$ is a left divisor of $\mathbf{G}(z)$.

We say that $\mathbf{A}(z)$ and $\mathbf{B}(z)$ are *left coprime* if their GCLD is unimodular, that is, is an invertible matrix polynomial.

By clearing denominators we may assume that both $\mathbf{A}(z)$ and $\mathbf{B}(z)$ have coefficients in $\mathbb{D}$. Note that this operation does not change the GCLDs of

$\mathbf{A}(z)$ and $\mathbf{B}(z)$. We will make the standard assumption that the matrix $[\mathbf{A}(z)\ \mathbf{B}(z)]$ has full row rank; otherwise, a GCLD may have elements of arbitrarily high degree [13, page 378]. This assumption also ensures that all GCLDs of $\mathbf{A}(z)$ and $\mathbf{B}(z)$ are nonsingular, and that the GCLD of $\mathbf{A}(z)$ and $\mathbf{B}(z)$ is unique up to multiplication on the right by a unimodular matrix [13].

A matrix polynomial is said to be *column (row) reduced* if the leading coefficients of each column (row) form a matrix having full column (row) rank over $\mathbb{Q}$ [13]. For a square matrix polynomial $\mathbf{A}(z) \in \mathbb{D}[z]^{m\times m}$, $\mathbf{A}(z)$ is column reduced if and only if $\deg\det \mathbf{A}(z) = \sum_{i=1}^{m} d_i$, where the d_i's are the degrees of the columns of $\mathbf{A}(z)$.

We are interested in solving a type of *extended matrix GCLD problem.* That is, we wish to find a GCLD $\mathbf{G}(z)$ of $\mathbf{A}(z)$ and $\mathbf{B}(z)$, as well as matrices $\mathbf{S}(z)$, $\mathbf{T}(z)$, $\mathbf{U}(z)$, $\mathbf{V}(z)$ of appropriate sizes, such that

$$\begin{aligned} \mathbf{A}(z)\cdot\mathbf{S}(z) + \mathbf{B}(z)\cdot\mathbf{T}(z) &= \mathbf{G}(z), \\ \mathbf{A}(z)\cdot\mathbf{U}(z) + \mathbf{B}(z)\cdot\mathbf{V}(z) &= \mathbf{0}. \end{aligned} \tag{1}$$

In this case, both $\mathbf{A}(z)\cdot\mathbf{U}(z)$ and $\mathbf{B}(z)\cdot\mathbf{V}(z)$ are Least Common Right Multiples (LCRM) of $\mathbf{A}(z)$ and $\mathbf{B}(z)$. Furthermore, if $\mathbf{B}(z)$ is square and nonsingular, then

$$\mathbf{B}^{-1}(z)\cdot\mathbf{A}(z) = -\mathbf{V}(z)\cdot\mathbf{U}(z)^{-1} \tag{2}$$

as a matrix rational function, such that $\mathbf{U}(z)$ and $\mathbf{V}(z)$ are right coprime. The matrix rational function $-\mathbf{V}(z)\cdot\mathbf{U}(z)^{-1}$ is called an *irreducible right matrix-fraction description* (MFD) of the matrix rational function [13].

Let $\mathbf{F}(z) = [\mathbf{A}(z)\ \mathbf{B}(z)] \in \mathbb{D}[z]^{m\times n}$ with $n = n_1 + n_2$. If we can find a unimodular matrix $\mathbf{M}(z) \in \mathbb{D}[z]^{n\times n}$ such that

$$\mathbf{F}(z)\cdot\mathbf{M}(z) := \mathbf{R}(z) = [\mathbf{G}(z)\ \mathbf{0}] \tag{3}$$

for some $\mathbf{G}(z) \in \mathbb{D}[z]^{m\times m}$, then $\mathbf{G}(z)$ is a GCLD of $\mathbf{A}(z)$ and $\mathbf{B}(z)$ [13, Lemma 6.3–3, page 377]. Partitioning $\mathbf{M}(z)$ into submatrices of appropriate dimensions

$$\mathbf{M}(z) = \begin{bmatrix} \mathbf{M}_{11}(z) & \mathbf{M}_{12}(z) \\ \mathbf{M}_{21}(z) & \mathbf{M}_{22}(z) \end{bmatrix} \tag{4}$$

and setting

$$\begin{aligned} \mathbf{S}(z) &= \mathbf{M}_{11}(z) \qquad & \mathbf{T}(z) &= \mathbf{M}_{21}(z) \\ \mathbf{U}(z) &= \mathbf{M}_{12}(z) \qquad & \mathbf{V}(z) &= \mathbf{M}_{22}(z) \end{aligned} \tag{5}$$

solves the extended GCLD problem (1). This is how matrix GCLDs are computed, for example, by finding matrix normal forms for $\mathbf{F}(z)$.

Similarly, we can define the *Greatest Common Right Divisor* (GCRD) and the corresponding *extended GCRD problem.* To compute a GCRD of $\mathbf{A}(z)$ and $\mathbf{B}(z)$, we can compute $\mathbf{G}(z)^T$ where $\mathbf{G}(z)$ is a GCLD of $\mathbf{A}(z)^T$ and $\mathbf{B}(z)^T$. We can also solve the extended GCRD problem and obtain an irreducible left MFD in the same way. If the algorithm for computing a GCLD gives a column reduced GCLD, then the GCRD computed is row reduced.

3 Issues in Designing a Modular Algorithm

For the remainder of this paper, we will assume that $\mathbb{D} = \mathbb{Z}$ and the reduction homomorphisms used are ϕ_p where p is a prime and $\phi_p(\mathbf{A}(z))$ is the result of reducing each entry of $\mathbf{A}(z)$ to $\mathbb{Z}_p[z]$. Our results can easily be generalized to other choices of $\mathbb{D}$. For example, if $\mathbb{D} = R[x]$ for some coefficient ring R, the reduction homomorphisms are evaluations of the matrix polynomial at $x = \alpha$. If $\mathbb{D}$ is a multivariate polynomial ring, our modular algorithm can be applied recursively to eliminate one variable at a time.

In order to design a modular algorithm, we must recognize when the computed result over $\mathbb{Z}_p$ can be used to reconstruct the final result. This is the problem of recognizing lucky primes. In the scalar case, it is well-known that p is lucky if p does not divide the leading coefficient of $\mathbf{A}(z)$ and $\mathbf{B}(z)$, and that $\deg \mathbf{G}(z) = \deg \mathbf{G}_p(z)$, where $\mathbf{G}_p(z)$ is the GCD computed over $\mathbb{Z}_p$. In the matrix polynomial case, it can be shown that if p does not divide the leading coefficients of $\det \mathbf{A}(z)$ and $\det \mathbf{B}(z)$, then

$$\deg \det \mathbf{G}(z) \leq \deg \det \mathbf{G}_p(z). \tag{6}$$

Unfortunately it is nontrivial to determine if p divides the leading coefficients of $\det \mathbf{A}(z)$ and $\det \mathbf{B}(z)$ unless the input polynomials are column reduced. The criteria for the scalar case does not extend to the matrix case easily.

We also need to normalize the computed results over $\mathbb{Z}_{p_i}$ so that each of them is an image of the same result over $\mathbb{Z}$. In the scalar case one simply needs to scale the input appropriately [11]. For matrix polynomials, however, simply requiring that $\deg \det \mathbf{G}(z) = \deg \det \mathbf{G}_p(z)$ as in the scalar case is insufficient. While the scaling may be sufficient to normalize $\det \mathbf{G}_p(z)$, it is not obvious how to normalize $\mathbf{G}_p(z)$ itself.

Finally, we must compute a bound on the number of primes required in the modular algorithm to ensure that the reconstructed result is correct.

4 Computing a Matrix GCLD over $\mathbb{Z}_p$

The computation of a matrix GCLD is nontrivial in comparison to the scalar case as there is no obvious Euclidean-like algorithm. For our purpose, we choose the Fast Fraction-Free Gaussian (FFFG) elimination algorithm [4] as a basis for the algorithm to compute a solution to the extended GCLD problem (1). The FFFG elimination algorithm computes a solution to (1) over any integral domain in a fraction-free way, such that the GCLD computed is column reduced [5]. It also computes a unimodular matrix satisfying (3) in a fraction-free manner. The algorithm can be viewed as an efficient method for solving a linear system of equations in which the coefficient matrix is structured. It is similar to the fraction-free Gaussian elimination algorithm by Bareiss [1] and computes the Cramer solutions to the linear systems, but it also takes advantage of the matrix structure to make the elimination more efficient.

In our case, we will perform the same arithmetic operations modulo p. It is unusual to use a fraction-free algorithm for computation over $\mathbb{Z}_p$ because there is no coefficient growth. However, this ensures that the result computed over $\mathbb{Z}_p$ is the image of the Cramer solutions computed over $\mathbb{Z}$. This solves part of the normalization problem.

4.1 FFFG

We first give some definitions to facilitate the description of the algorithm. Let $N = \deg \mathbf{F}(z)$ and write $\mathbf{F}(z) = \sum_{i=0}^{N} \mathbf{F}_i z^i$ where $\mathbf{F}_i \in \mathbb{D}^{m \times n}$ for all $0 \leq i \leq N$. We denote the jth column of $\mathbf{F}_i$ by $\mathbf{F}_i^{(\cdot,j)}$. Let $\vec{v} \in \mathbb{N}^n$ be a *multi-index*, $\vec{v}^{(i)}$ be the ith component of $\vec{v}$, and $|\vec{v}| = \sum_{i=1}^{n} \vec{v}^{(i)}$. We also define $z^{\vec{v}}$ to be the $n \times n$ matrix polynomial with $z^{\vec{v}^{(i)}}$ in entry (i,i) and 0 elsewhere.

For $\sigma > 0$, the *striped Krylov matrix* $\mathbf{K}(\mathbf{F}, \vec{v}, \sigma)$ associated to the GCLD problem is defined to be the matrix consisting of the first σ rows of

$$\mathbf{K}(\mathbf{F},\vec{v}) = \left[\begin{array}{cccccc} \mathbf{F}_N^{(\cdot,1)} & & & \mathbf{F}_N^{(\cdot,n)} & & \\ \vdots & \ddots & & \vdots & \ddots & \\ \mathbf{F}_0^{(\cdot,1)} & & \mathbf{F}_N^{(\cdot,1)} \quad \cdots & \mathbf{F}_0^{(\cdot,n)} & & \mathbf{F}_N^{(\cdot,n)} \\ & \ddots & \vdots & & \ddots & \vdots \\ & & \mathbf{F}_0^{(\cdot,1)} & & & \mathbf{F}_0^{(\cdot,n)} \end{array}\right] \tag{7}$$

$$\underbrace{\qquad\qquad\qquad}_{\vec{v}^{(1)}} \qquad \underbrace{\qquad\qquad\qquad}_{\vec{v}^{(n)}}$$

We can view $\mathbf{K}(\mathbf{F},\vec{v},\sigma)$ as a matrix having n stripes, each containing the columns of $\mathbf{F}(z)$ multiplied by $z^{\vec{v}^{(l)}-1}, \ldots, z, 1$. This is a generalization of the Sylvester matrix in the scalar case, and is a special case of the striped Krylov matrix defined in [4] for matrix rational interpolation problems. The computation of $\mathbf{M}(z)$ and $\mathbf{G}(z)$ in (3) can be viewed as performing Gaussian elimination on $\mathbf{K}(\mathbf{F},\vec{v})$ for some $\vec{v}$ until the last column of $n-m$ stripes is zero. The column operations performed are represented in $\mathbf{M}(z)$.

Let $\mathbf{K}(\mathbf{F})$ be the infinite matrix

$$\mathbf{K}(\mathbf{F}) = \left[\begin{array}{cccccccc} \mathbf{F}_N^{(\cdot,1)} & & & & \cdots & \mathbf{F}_N^{(\cdot,n)} & & \\ \vdots & \ddots & & & \cdots & \vdots & \ddots & \\ \mathbf{F}_0^{(\cdot,1)} & & \mathbf{F}_N^{(\cdot,1)} & & \cdots & \mathbf{F}_0^{(\cdot,n)} & & \mathbf{F}_N^{(\cdot,n)} \\ & \ddots & \vdots & \ddots & \cdots & & \ddots & \vdots & \ddots \\ & & \mathbf{F}_0^{(\cdot,1)} & & \cdots & & & \mathbf{F}_0^{(\cdot,n)} \\ & & & \ddots & \cdots & & & & \ddots \end{array}\right]. \tag{8}$$

Then $\mathbf{K}(\mathbf{F},\vec{v})$ is a submatrix of $\mathbf{K}(\mathbf{F})$ for any $\vec{v}$. We define the sequence of integers $(\sigma(j))_{j=0,1,\ldots}$ to be the maximal sequence of linearly independent rows of $\mathbf{K}(\mathbf{F})$. For $j \geq 0$, the sequence satisfies

(a) rows $\sigma(0), \sigma(1), \ldots, \sigma(j)$ of $\mathbf{K}(\mathbf{F})$ are linearly independent, and

(b) rows $\sigma(0), \ldots, \sigma(j-1), i$ of $\mathbf{K}(\mathbf{F})$ are linearly dependent for all $0 \leq i < \sigma(j)$.

That is, the sequence is the lexicographically smallest sequence among all possible choices of linearly independent rows. The matrix $\mathbf{K}^*(\mathbf{F},\vec{v},j)$ is defined to be the submatrix of $\mathbf{K}(\mathbf{F},\vec{v})$ consisting of rows $\sigma(0), \sigma(1), \ldots, \sigma(j-$

1). Intuitively, these are the rows that require elimination when we perform Gaussian elimination on $\mathbf{K}(\mathbf{F})$. We also define $d^*(\vec{v}) = \det \mathbf{K}^*(\mathbf{F}, \vec{v}, |\vec{v}|)$. A multi-index $\vec{v}$ is *para-normal* if $d^*(\vec{v}) \neq 0$, and is *σ-normal* if it is para-normal and $\sigma(|\vec{v}| - 1) < \sigma \leq \sigma(|\vec{v}|)$.

The algorithm (which we call FF_GCLD) for computing matrix polynomial GCLDs makes use of a subroutine (Algorithm 1) adapted from the FFFG elimination algorithm [4, 5]. We have modified the algorithm to maintain the sign of the determinant as it is required in the modular algorithm described later. We will also call the variant of Algorithm 1 using arithmetic operations in $\mathbb{Z}_p$ FF_ReduceModp.

The FF_Reduce algorithm (Algorithm 1) transforms $\mathbf{F}(z)$ into $\mathbf{R}(z)$ with m nonzero columns by performing column operations, which are recorded in $\mathbf{M}(z)$. The matrix polynomial $\mathbf{M}(z)$ can be viewed as solutions to systems of linear systems of equations whose coefficient matrices are submatrices of $\mathbf{K}(\mathbf{F})$. The algorithm takes advantage of the structure of $\mathbf{K}(\mathbf{F})$ by maintaining only one column per stripe. The multi-index $\vec{v}$ indicates the number of times each column of $\mathbf{F}(z)$ has been used as a pivot in the elimination process. In the algorithm, the coefficients of z^N are eliminated one row at a time, followed by the coefficients of z^{N-1}, and so on. The elimination process ensures that the entries of $\mathbf{R}(z)$ and $\mathbf{M}(z)$ remain in $\mathbb{D}[z]$ at each step by dividing by the previous pivot, in a similar way as Bareiss' fraction-free Gaussian elimination [1]. At the end of each iteration, we have

$$\mathbf{M}^{(i,j)}(z) = z^{-\vec{v}^{(i)} + (1-\delta_{i,j})} m_{i,j}(x) \tag{9}$$

where $\delta_{i,j}$ is the Kronecker δ and $m_{i,j}(x)$ is a polynomial of degree at most $\vec{v}^{(i)} - (1 - \delta_{i,j})$. Moreover, the constant coefficient of $m_{i,i}(x)$ is d. The matrix polynomial $\mathbf{M}(z)$ computed by Algorithm 1 is called the *Mahler system of type $\vec{v}$* [4]. The rank of the coefficient matrix formed from the leading coefficients of each column is computed in *rank*, which allows us to determine when the nonzero columns of $\mathbf{R}(z)$ is column reduced. Note that if we let $\mathbf{B}(z)$ be an "empty" matrix polynomial with $n_2 = 0$, the FF_GCLD algorithm transforms $\mathbf{A}(z)$ into column reduced form.

4.2 Properties of the Algorithm

Some properties of the fraction-free GCLD algorithm are stated below to facilitate the development of the modular algorithm. See [4] for the proofs

Algorithm 1 Fraction-free Reduction for a Matrix Polynomial

Input: $\mathbf{F}(z) \in \mathbb{D}[z]^{m\times n}$ such that $\mathbf{F}(z)$ has full row rank.

Output: $\vec{v} \in \mathbb{N}^n$, $\vec{s} \in \mathbb{N}^{|\vec{v}|}$, $\mathbf{M}(z) \in \mathbb{D}[z]^{n\times n}$, and $\mathbf{R}(z) \in \mathbb{D}[z]^{m\times n}$ such that $\det \mathbf{M}(z) = d^*(\vec{v})^n$, $\vec{s}^{(i)} = \sigma(i-1)$, $\mathbf{F}(z)\cdot\mathbf{M}(z) = \mathbf{R}(z)$, and $\mathbf{R}(z)$ has m nonzero columns and is column reduced.

```
procedure FF_Reduce($\mathbf{F}(z)$)
  $N \leftarrow \deg \mathbf{F}(z)$
  $(\vec{v}, \vec{s}, \mathbf{M}(z), \mathbf{R}(z)) \leftarrow (\vec{0}, \vec{0}, \mathbf{I}_n, \mathbf{F}(z), 0)$
  $(\sigma, d, \epsilon, rank) \leftarrow (0, 1, 1, 0)$
  while $rank <$ the number of nonzero columns of $\mathbf{R}(z)$ do
    if $\sigma \equiv 0 \bmod m$ then
      $rank \leftarrow 0$
    end if
    $r_\ell \leftarrow \mathrm{coeff}(\mathbf{R}^{((\sigma \bmod m)+1,\ell)}(z), z^{N-\sigma \operatorname{div} m})$ for $\ell = 1, \ldots, n$
    $\Lambda = \{\ell \in \{1, \ldots, n\} : r_\ell \neq 0\}$
    if $\Lambda \neq \{\}$ then
      $\pi \leftarrow \min\{\ell \in \Lambda : \vec{v}^{(\ell)} = \min_{\mu\in\Lambda} \vec{v}^{(\mu)}\}$
      $p_\ell \leftarrow \mathrm{coeff}(\mathbf{M}^{(\ell,\pi)}(z), z^{1-\vec{v}^{(\ell)}})$ for $\ell = 1, \ldots, n$, $\ell \neq \pi$
      $\vec{v}^{(\pi)} \leftarrow \vec{v}^{(\pi)} + 1$
      $\vec{s}^{(|\vec{v}|)} \leftarrow \sigma + 1$
      $rank \leftarrow rank + 1$
      for all $\ell = 1, \ldots, n$, $\ell \neq \pi$ do
        $\mathbf{M}^{(\cdot,\ell)}(z) \leftarrow \left(r_\pi \cdot \mathbf{M}^{(\cdot,\ell)}(z) - r_\ell \cdot \mathbf{M}^{(\cdot,\pi)}(z)\right) \cdot d^{-1}$
        $\mathbf{R}^{(\cdot,\ell)}(z) \leftarrow \left(r_\pi \cdot \mathbf{R}^{(\cdot,\ell)}(z) - r_\ell \cdot \mathbf{R}^{(\cdot,\pi)}(z)\right) \cdot d^{-1}$
      end for
      $\mathbf{M}^{(\cdot,\pi)}(z) = \left(r_\pi \cdot \frac{1}{z} \cdot \mathbf{M}^{(\cdot,\pi)}(z) - \sum_{\ell\neq\pi} p_\ell \cdot \mathbf{M}^{(\cdot,\ell)}(z)\right) \cdot d^{-1}$
      $\mathbf{R}^{(\cdot,\pi)}(z) = \left(r_\pi \cdot \frac{1}{z} \cdot \mathbf{R}^{(\cdot,\pi)}(z) - \sum_{\ell\neq\pi} p_\ell \cdot \mathbf{R}^{(\cdot,\ell)}(z)\right) \cdot d^{-1}$
      $d \leftarrow r_\pi$, $\epsilon \leftarrow \epsilon \cdot (-1)^{\sum_{i=\pi+1}^{n} \vec{v}^{(i)}}$
    end if
    $\sigma \leftarrow \sigma + 1$
  end while
  $(\mathbf{M}(z), \mathbf{R}(z)) \leftarrow (\epsilon \cdot \mathbf{M}(z) \cdot z^{\vec{v}}, \epsilon \cdot \mathbf{R}(z) \cdot z^{\vec{v}})$
```

and details. We first state a result which is used in the definition of lucky primes in Section 5.

Theorem 4.1 *Let $(\vec{w}_k)_{k=0,1,2\ldots}$ be the sequence of multi-indices defined by*

$$\begin{aligned}\vec{w}_0 &= \vec{0}\\ \vec{w}_{k+1} &= \vec{w}_k + \vec{e}_\pi, \text{ where } \pi = \min_{1\le i\le n}\left\{ i : \vec{w}_k^{(i)} = \min_{1\le j\le n} \vec{w}_k^{(j)} \right\}.\end{aligned}$$

After σ iterations of the loop in FF_Reduce (or FF_ReduceModp), the multi-index $\vec{v}$ is the unique closest σ-normal point to sequence $(\vec{w}_k)_k$. That is, if $\vec{u}$ is σ-normal, then

$$\left|\max\{\vec{0}, \vec{w}_k - \vec{v}\}\right| \le \left|\max\{\vec{0}, \vec{w}_k - \vec{u}\}\right| \qquad \textit{for } k \ge 0, \tag{10}$$

where max *is defined component-wise. At the end of FF_Reduce (or FF_ReduceModp), $\vec{v}$ is the unique closest σ-normal point to the sequence $(\vec{w}_k)_k$, with $\sigma = \sigma(|\vec{v}| - 1) + 1$.* □

The next result allows us to obtain a bound on the number of images required in Section 6.

Theorem 4.2 *At the end of FF_Reduce (or FF_ReduceModp), $d^*(\vec{v}) \ne 0$. Also, $\deg \mathbf{M}^{(i,j)}(z) \le \vec{v}^{(i)}$ and we can write*

$$\mathbf{M}^{(i,j)}(z) = \\ (-1)^{\sum_{k=j+1}^{n} \vec{v}^{(k)}} \cdot \det \left[\frac{\mathbf{K}^*(\mathbf{F}, \vec{v} + \vec{e}_j, |\vec{v}|)}{0, \ldots, 0, z^{\vec{v}^{(j)}}, \ldots, z^{\vec{v}^{(j)} - \vec{v}^{(i)} + 1 - \delta_{i,j}}, 0, \ldots, 0} \right], \tag{11}$$

where the last row of the determinant has nonzero entries in the ith stripe, and $\vec{e}_j$ is the jth unit vector in the standard basis. □

Theorem 4.3 *Let K be an upper bound on the size of the coefficients appearing in $\mathbf{A}(z)$ and $\mathbf{B}(z)$, and $N = \max(\deg \mathbf{A}(z), \deg \mathbf{B}(z))$. Then FF_Reduce (or FF_ReduceModp) terminates in at most $m(N + mN + 1) \in O(m^2 N)$ iterations. The worst case bit complexity of FF_GCLD is $O(n(m^2N)^4K^2)$, assuming that the product of two elements in $\mathbb{D}$ of size k can be computed in $O(k^2)$ bit operations.* □

Theorem 4.3 can be applied to any integral domain $\mathbb{D}$ in which products can be computed in quadratic time. In particular, it is applicable when $\mathbb{D} = \mathbb{Z}$ or $\mathbb{D} = R[x]$ in which multiplications in R can be done in constant time. Even for small matrices, the run time can be significant in the worst case due to coefficient growth. The coefficient growth is greatest when the number of steps in the elimination process is large, corresponding to the case when the degree of the GCLD is small compared to N.

Example 4.4 *Let*

$$\mathbf{A}(z) = \begin{bmatrix} 3\,z^4 + 3\,z^3 + 4\,z^2 - 2\,z - 4 & z^4 + 5\,z^3 + 3\,z^2 + 3\,z + 1 \\ 3\,z^4 + 3\,z^2 + 14\,z + 8 & z^4 + 7\,z^3 + 6\,z^2 + z + 1 \end{bmatrix},$$
$$\mathbf{B}(z) = \begin{bmatrix} z^3 + 9\,z^2 + 5\,z + 1 & z^5 + z^4 + 2\,z^3 + 3\,z^2 + 2\,z + 1 \\ z^3 + 15\,z^2 + 19\,z + 5 & z^5 + z^3 + 7\,z^2 + 6\,z + 1 \end{bmatrix}.$$

FF_GCLD returns

$$\mathbf{G}(z) = \begin{bmatrix} 2480256\,z^2 - 4960512\,z - 9921024 & 3720384\,z \\ 14881536\,z^2 + 34723584\,z + 19842048 & 7440768\,z + 7440768 \end{bmatrix},$$

with FF_Reduce returning

$$\vec{v} = [5, 4, 3, 2],$$
$$\vec{s} = [0, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

and $d^*(\vec{v}) = -2480256 = -2^7 \cdot 3^2 \cdot 2153$. *Note that* $\mathbf{G}(z)$ *is column reduced. The matrices* S, T, U, V *are not shown.* □

5 Lucky Primes and Normalization

Let $(\vec{v}, \vec{s}, \mathbf{M}(z), \mathbf{R}(z))$ be the results obtained by FF_Reduce when the operations are performed over the integers, and $\mathbf{G}(z)$ be the $m \times m$ submatrix of $\mathbf{R}(z)$ containing the nonzero columns. Similarly, let $(\vec{v}_p, \vec{s}_p, \mathbf{M}_p(z), \mathbf{R}_p(z))$ and $\mathbf{G}_p(z)$ be the results computed modulo a prime p. Note that $\mathbf{M}_p(z)$ and $\mathbf{R}_p(z)$ can be used in the reconstruction if

$$\phi_p(\mathbf{M}(z)) = \mathbf{M}_p(z) \quad (12)$$
$$\phi_p(\mathbf{R}(z)) = \mathbf{R}_p(z). \quad (13)$$

If this is not the case, we say that p is unlucky. Since we require the computed results to be the exact images of $\mathbf{M}(z)$ and $\mathbf{R}(z)$, we have solved the normalization problem as well.

Formally, we define a lucky prime in the following way.

Definition 5.1 *Let p be a prime. Then p is* lucky *if $d^*(\vec{v}) \not\equiv 0 \bmod p$ and $|\vec{v}| = |\vec{v}_p|$. Otherwise, p is* unlucky*.* $\square$

Since the matrix polynomial $\mathbf{R}_p(z)$ computed by FF_Reduce is in column reduced form, this definition implies that $\deg \det \mathbf{G}(z) = \deg \det \mathbf{G}_p(z)$. We also require that the results computed modulo p to be the exact image of the result over $\mathbb{Z}$ instead of the image of an associate of the result. This eliminates the need to additionally normalize the result computed modulo p. Thus, it is necessary to maintain the sign of the determinant in FF_Reduce as it was stated in Algorithm 1. Note also that if $\deg \phi_p(\mathbf{F}(z)) < \deg \mathbf{F}(z)$, then p is unlucky as row $\vec{s}^{(1)}$ of $\phi_p(\mathbf{K}^*(\mathbf{F}, \vec{v}, |\vec{v}|))$ consists only of zeros and therefore $d^*(\vec{v}) \equiv 0 \bmod p$.

We now show that this definition is sufficient. To facilitate the proofs, we let $\mathbf{K}^*(\mathbf{F}, \vec{v}, |\vec{v}|)$ be the submatrix of the associated striped Krylov matrix with rows indexed by $\vec{s}$, and $\mathbf{K}_p^*(\phi_p(\mathbf{F}), \vec{v}_p, |\vec{v}_p|)$ be the corresponding matrix with rows indexed by $\vec{s}_p$ over $\mathbb{Z}_p$. We denote their determinants by $d^*(\vec{v})$ and $d_p^*(\vec{v}_p)$, respectively. We first prove a lemma which will be used for detecting whether a prime is lucky.

Lemma 5.2 *Let p be a prime such that $\deg \mathbf{F}(z) = \deg \phi_p(\mathbf{F}(z))$. If $|\vec{v}_p| = |\vec{v}|$, then*

(a) $\vec{v}$ is at least as close to $(\vec{w}_k)_k$ as $\vec{v}_p$.

(b) $\vec{s} \leq_{lex} \vec{s}_p$, where $\leq_{lex}$ compares two vectors in lexicographical order.

$\square$

Proof.

(a) Let $\sigma = \sigma(|\vec{v}| - 1) + 1$. Then $\vec{v}_p$ is σ-normal over $\mathbb{Z}_p$ and so $d_p^*(\vec{v}_p) \neq 0$. Hence, $d^*(\vec{v}_p) \neq 0$ and $\vec{v}_p$ is σ-normal over $\mathbb{Z}$. The result now follows from Theorem 4.1.

(b) The rows indexed by $\vec{s}_p$ in $\mathbf{K}(\phi_p(\mathbf{F}))$ are linearly independent over $\mathbb{Z}_p$. The same rows in $\mathbf{K}(\mathbf{F})$ are also linearly independent over $\mathbb{Z}$. By definition, $\vec{s}$ is the lexicographically first set of rows such that $\mathbf{K}(\mathbf{F})$ are linearly independent over $\mathbb{Z}$. It follows that $\vec{s} \leq_{lex} \vec{s}_p$.

□

We now give an equivalent definition of lucky primes which is more useful for the detection of unlucky primes.

Theorem 5.3 *Let p be a prime such that* $\deg \mathbf{F}(z) = \deg \phi_p(\mathbf{F}(z))$. *Then p is lucky if and only if $\vec{v}_p = \vec{v}$ and $\vec{s}_p = \vec{s}$.* □

Proof. Suppose p is lucky. Since $d^*(\vec{v}) \not\equiv 0 \bmod p$, $\phi_p(\mathbf{K}^*(\mathbf{F}, \vec{v}, |\vec{v}|))$ is nonsingular over $\mathbb{Z}_p$. Thus, the rows indexed by $\vec{s}$ are linearly independent over $\mathbb{Z}_p$, so that $\vec{s}_p \leq_{lex} \vec{s}$. But $\vec{s} \leq_{lex} \vec{s}_p$ by Lemma 5.2(b), hence $\vec{s}_p = \vec{s}$. Now let $\sigma = \sigma(|\vec{v}| - 1) + 1 = \sigma(|\vec{v}_p| - 1) + 1$. Then $\vec{v}$ is σ-normal over $\mathbb{Z}$, and hence it is also σ-normal over $\mathbb{Z}_p$ because $d^*(\vec{v}) \not\equiv 0 \bmod p$. Thus, $\vec{v}_p$ is at least as close to $(\vec{w}_k)_k$ as $\vec{v}$ by Theorem 4.1. On the other hand, $\vec{v}$ is the unique closest σ-normal point to $(\vec{w}_k)_k$. Therefore, $\vec{v}_p = \vec{v}$.

Conversely, assume that $\vec{v}_p = \vec{v}$ and $\vec{s}_p = \vec{s}$. Clearly $|\vec{v}| = |\vec{v}_p|$. Moreover, $\phi_p(\mathbf{K}^*(\mathbf{F}, \vec{v}, |\vec{v}|)) = \mathbf{K}_p^*(\phi_p(\mathbf{F}), \vec{v}_p, |\vec{v}_p|)$, so $d_p^*(\vec{v}_p) = d_p^*(\vec{v}) \not\equiv 0 \bmod p$. □

Finally, we show that Definition 5.1 provides the property that the computed results modulo p are the desired images.

Theorem 5.4 *If p is lucky, then $\phi_p(\mathbf{M}(z)) = \mathbf{M}_p(z)$ and $\phi_p(\mathbf{R}(z)) = \mathbf{R}_p(z)$.* □

Proof. Suppose that p is a lucky prime, so that $\vec{v} = \vec{v}_p$ and $\vec{s} = \vec{s}_p$ by Theorem 5.3. Then $\phi_p(\mathbf{K}^*(\mathbf{F}, \vec{v} + \vec{e}_j, |\vec{v}|)) = \mathbf{K}_p^*(\phi_p(\mathbf{F}), \vec{v}_p + \vec{e}_j, |\vec{v}_p|)$ for all j. It follows by (11) that $\phi_p(\mathbf{M}(z)) = \mathbf{M}_p(z)$. Also,

$$\begin{aligned} \phi_p(\mathbf{R}(z)) &= \phi_p(\mathbf{F}(z) \cdot \mathbf{M}(z)) = \phi_p(\mathbf{F}(z)) \cdot \phi_p(\mathbf{M}(z)) \\ &= \phi_p(\mathbf{F}(z)) \cdot \mathbf{M}_p(z) = \mathbf{R}_p(z). \end{aligned} \tag{14}$$

□

We need to check that $\vec{v}_p = \vec{v}$ and $\vec{s}_p = \vec{s}$ to determine if p is lucky. However, $\vec{v}$ and $\vec{s}$ are not known in advance. Instead, we can compare the results computed modulo two primes and detect which, if any, are unlucky.

Theorem 5.5 *Let p and q be primes such that* $\deg \mathbf{F}(z) = \deg \phi_p(\mathbf{F}(z)) = \deg \phi_q(\mathbf{F}(z))$, *and let $(\vec{v}_p, \vec{s}_p, \mathbf{M}_p(z), \mathbf{R}_p(z))$, $(\vec{v}_q, \vec{s}_q, \mathbf{M}_q(z), \mathbf{R}_q(z))$ be the results computed by FF_Reduce modulo p and q, respectively. Then p is unlucky if any of the following holds:*

(a) $|\vec{v}_p| \neq |\vec{v}_q|$;

(b) $|\vec{v}_p| = |\vec{v}_q|$ *and* $\vec{v}_q$ *is closer to* $(\vec{w}_k)_k$ *than* $\vec{v}_p$*;*

(c) $|\vec{v}_p| = |\vec{v}_q|$ *and* $\vec{s}_p >_{lex} \vec{s}_q$*.*

Furthermore, if p *is unlucky and* q *is lucky, then at least one of the above must hold.* $\square$

Proof. Condition (a) follows from Definition 5.1, (b) and (c) follow from Lemma 5.2. Letting $\vec{v}_q = \vec{q}$ and $\vec{s}_q = \vec{s}$, we see that one of the conditions must hold if p is unlucky, since either $\vec{v}_p \neq \vec{v}$ or $\vec{s}_p \neq \vec{s}$ by Theorem 5.3. $\square$

Note that when q is lucky but $|\vec{v}_p| \neq |\vec{v}_q|$, we cannot determine whether q is lucky or not by the criteria above. Thus, we must discard both p and q even though q may be lucky. Also, even when $\phi_p(\mathbf{F}(z))$ does not have full row rank, it is not difficult to see that $\mathbf{M}_p(z)$ and $\mathbf{R}_p(z)$ computed by FF_Reduce are solutions to the extended GCLD problem (1) over $\mathbb{Z}_p$. If p is also lucky then $\mathbf{M}_p(z)$ and $\mathbf{R}_p(z)$ can still be used to reconstruct the final result.

6 Number of Images Required

We need to determine the number of lucky primes required to reconstruct the final result. This can be done by obtaining a bound on the coefficients appearing in $\mathbf{M}(z)$ and $\mathbf{R}(z)$. The key observation is that these coefficients are simply determinants of various submatrices of $\mathbf{K}(\mathbf{F})$. Therefore, we can apply Hadamard's inequality [12]

$$|\det \mathbf{A}| \leq \prod_{i=1}^{n} \left(\sum_{j=1}^{n} |\mathbf{A}^{(i,j)}|^2 \right)^{1/2} \tag{15}$$

for any $n \times n$ matrix $\mathbf{A}$ to obtain the bounds on the coefficients.

Theorem 6.1 *Let* κ *be an upper bound on the absolute values of the coefficients of* $\mathbf{A}(z)$ *and* $\mathbf{B}(z)$*. Then the absolute values of the coefficients of* $\mathbf{M}(z)$ *and* $\mathbf{R}(z)$ *are bounded by* $t^{t/2}\kappa^t$*, where* $t = m(N + mN + 1) + 1$*.* $\square$

Proof. Let $s = |\vec{v}|$. By Theorem 4.2, we see that the coefficients of $\mathbf{M}(z)$ are simply minors of $\mathbf{K}(\mathbf{F})$ of order s. Since $\mathbf{F}(z) \cdot \mathbf{M}(z) = \mathbf{R}(z)$, we have

$$\mathbf{R}^{(i,j)}(z) = (-1)^{\sum_{k=j+1}^{n} \vec{v}^{(k)}} \sum_{\ell=1}^{n} \det \left[\frac{\mathbf{K}^*(\mathbf{F}, \vec{v} + \vec{e}_j, |\vec{v}|)}{\mathbf{E}^{(\ell)}(z)} \right], \tag{16}$$

where

$$\mathbf{E}^{(\ell)}(z) = [0, \ldots, 0, z^{\vec{v}(j)} \cdot \mathbf{F}^{(\cdot,\ell)}, \ldots, z^{\vec{v}(j)-\vec{v}(\ell)+1-\delta_{i,j}} \cdot \mathbf{F}^{(\cdot,\ell)}, 0, \ldots 0]. \tag{17}$$

The sum in (16) can be written as a single determinant of order $s+1$ by multilinearity of determinants. Thus, the coefficients of $\mathbf{R}(z)$ are minors of $\mathbf{K}(\mathbf{F})$ of order $s+1$. By Hadamard's inequality, we see that the absolute values of the coefficients of $\mathbf{M}(z)$ and $\mathbf{R}(z)$ are bounded by

$$(s+1)^{(s+1)/2}\kappa^{(s+1)}. \tag{18}$$

From Theorem 4.3, we have $s \leq m(N + mN + 1)$. The result now follows by setting $t = s + 1$. □

Note that Theorem 6.1 provides an upper bound on the size of the coefficients in the computed solution to the extended GCLD problem (1). Therefore, we must have enough lucky primes $p_1, \ldots, p_\tau$ such that $\prod_{i=1}^{\tau} p_i \geq 2t^{t/2}\kappa^t$. If we choose primes that have approximately the same magnitude as κ, then we need $O(t \log t + t)$ lucky primes. In practice we may assume that $t \ll \kappa$ and in that case, so that $\log_\kappa t$ behaves like a constant and we only need $O(t)$ lucky primes [11, 18].

Finally, we wish to bound the number of unlucky primes. It can be shown that if p is unlucky, then p divides $d^*(\vec{v}')$ for some $\vec{v}'$ encountered when FF_Reduce is run over the integers, or p divides the product of some other minors of sizes $1, 2, \ldots, |\vec{v}|$. Since each of these minors have size bounded by $t^{t/2}\kappa^t$ and there are $O(t)$ of them, it follows that the number of unlucky primes $p \geq \kappa$ is at most $O(t^2 \log t)$. An analogous result in the scalar case is that an unlucky prime divides the product of the contents of the subresultants [11, Theorem 7.5]. In practice, however, this upper bound is grossly pessimistic and we rarely encounter any unlucky prime.

7 Complete Algorithm and Complexity

We now give the complete modular extended GCLD algorithm (Algorithm 2). We will denote by FF_ReduceModp the algorithm that is the same as FF_Reduce (Algorithm 1) except that arithmetic operations are performed modulo p. We also assume that there is a CRA algorithm that updates the reconstructed matrices by Chinese remaindering after each additional image has been computed. For example, Garner's algorithm can be applied to the individual coefficients of $\mathbf{M}(z)$ or $\mathbf{R}(z)$ [11].

Algorithm 2 The Modular Matrix GCLD Algorithm

Input:

$\mathbf{A}(z) \in \mathbb{D}[z]^{m\times n_1}$, $\mathbf{B}(z) \in \mathbb{D}[z]^{m\times n_2}$ such that $[\mathbf{A}(z)\ \mathbf{B}(z)]$ has full rank.

Output:

- $\mathbf{G}(z)$, a column reduced GCLD of $\mathbf{A}(z)$ and $\mathbf{B}(z)$;
- $\mathbf{S}(z), \mathbf{T}(z)$ such that $\mathbf{A}(z)\cdot\mathbf{S}(z) + \mathbf{B}(z)\cdot\mathbf{T}(z) = \mathbf{G}(z)$;
- $\mathbf{U}(z), \mathbf{V}(z)$ such that $\mathbf{A}(z)\cdot\mathbf{U}(z) + \mathbf{B}(z)\cdot\mathbf{V}(z) = \mathbf{0}$.

```
procedure Mod_GCLD($\mathbf{A}(z)$, $\mathbf{B}(z)$)
  Compute $\kappa$
  $(q, \vec{v}, \vec{s}, \mathbf{F}(z), \mathbf{M}(z), \mathbf{R}(z), done) \leftarrow (1, \vec{0}, \vec{0}, [\mathbf{A}(z)\ \mathbf{B}(z)], \mathbf{0}, \mathbf{0}, false)$
  while NOT $done$ do
    repeat
      $p \leftarrow$ a new prime with the same magnitude as $\kappa$
    until $\deg \phi_p(\mathbf{F}(z)) = \deg \mathbf{F}(z)$
    $(\vec{v}_p, \vec{s}_p, \mathbf{M}_p(z), \mathbf{R}_p(z)) \leftarrow$ FF_ReduceModp($\phi_p([\mathbf{A}(z)\ \mathbf{B}(z)]), p$)
    if $q = 1$ then
      $(q, \vec{v}, \vec{s}, \mathbf{M}(z), \mathbf{R}(z)) \leftarrow (p, \vec{v}_p, \vec{s}_p, \mathbf{M}_p(z), \mathbf{R}_p(z))$
    else
      if $|\vec{v}| \neq |\vec{v}_p|$ OR $\vec{v}$ is further from $(\vec{w}_k)_k$ than $\vec{v}_p$ OR $\vec{s} >_{lex} \vec{s}_p$ then
        $(q, \vec{v}, \vec{s}, \mathbf{M}(z), \mathbf{R}(z)) \leftarrow (1, \vec{0}, \vec{0}, \mathbf{0}, \mathbf{0})$
      end if
      if $q = 1$ OR $(\vec{v}, \vec{s}) = (\vec{v}_p, \vec{s}_p)$ then
        $(\mathbf{M}'(z), \mathbf{R}'(z)) \leftarrow (\mathrm{CRA}(\mathbf{M}(z), \mathbf{M}_p(z), q, p), \mathrm{CRA}(\mathbf{R}(z), \mathbf{R}_p(z), q, p))$
        $q \leftarrow qp$
        if $(\mathbf{M}(z), \mathbf{R}(z)) = (\mathbf{M}'(z), \mathbf{R}'(z))$ AND $\mathbf{F}(z)\cdot\mathbf{M}(z) = \mathbf{R}(z)$ then
          $done \leftarrow true$
        end if
        $(\mathbf{M}(z), \mathbf{R}(z)) \leftarrow (\mathbf{M}'(z), \mathbf{R}'(z))$
      end if
    end if
  end while
  Permute the columns of $\mathbf{M}(z)$ and $\mathbf{R}(z)$ so that the nonzero columns of $\mathbf{R}(z)$
  are in the first $m$ columns
  $\mathbf{G}(z) \leftarrow \mathbf{R}^{(1\ldots m, 1\ldots m)}(z)$
  $\mathbf{S}(z) \leftarrow \mathbf{M}^{(1\ldots n_1, 1\ldots m)}(z)$, $\mathbf{T}(z) \leftarrow \mathbf{M}^{(n_1+1\ldots n_1+n_2, 1\ldots m)}(z)$
  $\mathbf{U}(z) \leftarrow \mathbf{M}^{(1\ldots n_1, m+1\ldots n_1+n_2)}(z)$, $\mathbf{V}(z) \leftarrow \mathbf{M}^{(n_1+1\ldots n_1+n_2, m+1\ldots n_1+n_2)}(z)$.
```

We modified our modular algorithm so that when $\mathbf{M}(z)$ and $\mathbf{R}(z)$ have not changed for an additional prime, we compute $\mathbf{F}(z) \cdot \mathbf{M}(z)$ and compare it to $\mathbf{R}(z)$. If the two quantities are equal then we may terminate the algorithm and accept the results constructed. Although $\mathbf{M}(z)$ and $\mathbf{R}(z)$ may not be the same as the ones computed by the FF_GCLD algorithm, they still satisfy the extended matrix GCLD problem (1). This idea of early termination is similar to the trial division technique commonly used in the case of modular algorithms for scalar polynomial GCD, and is useful in practice because the Hadamard's bound is usually too pessimistic.

We now discuss the complexity of Mod_GCLD. For this analysis, we will assume that $t \ll \kappa$, where t and κ are defined in Theorem 6.1. We will also assume that we rarely encounter unlucky primes, so that there are at most $O(t)$ unlucky primes. Both assumptions are satisfied in most applications in practice.

Theorem 7.1 *Let K be a bound on the size of the coefficients appearing in $\mathbf{A}(z)$ and $\mathbf{B}(z)$, and $N = \max(\deg \mathbf{A}(z), \deg \mathbf{B}(z))$. The worst case bit complexity of Mod_GCLD is $O(n(m^2N)^3K^2)$, assuming that the product of two k-bit numbers can be computed in $O(k^2)$ bit operations.* $\square$

Proof. Let t be defined as in Theorem 6.1. Since each prime p has size approximately K, we need to use $O(t)$ lucky primes in Mod_GCLD (see remarks after Corollary 6.1). By the assumption on the number of unlucky primes, the total number of primes required is $O(t)$. Since $t \in O(m^2N)$, a total of $O(m^2N)$ primes are needed.

For each prime p, we can compute $\phi_p(\mathbf{F}(z))$ in $O(mnNK^2)$ bit operations. Using the same analysis as done in Theorem 4.3, the bit complexity for each invocation of FF_ReduceModp is $O(n(m^2N)^2K^2)$ (see [4]). Thus, the total bit complexity over all primes is $O(n(m^2N)^3K^2)$.

Finally, we need to perform the reconstruction of the final result by Chinese remaindering. By Theorem 6.1, each coefficient has size $O(tK)$ because $t \ll \kappa$ and $K \in O(\log \kappa)$. Thus, each coefficient in $\mathbf{M}(z)$ and $\mathbf{R}(z)$ can be reconstructed in $O(t^2K^2)$ bit operations [19]. There are $O(nt)$ potentially nonzero coefficients in $\mathbf{M}(z)$ and $O(mnN)$ potentially nonzero coefficients in $\mathbf{R}(z)$. Since $t \in O(m^2N)$, it follows that the total cost for the reconstruction is $O(n(m^2N)^3K^2)$ bit operations. Therefore, the total number of bit operations required is in $O(n(m^2N)^3K^2)$. $\square$

Compared to the $O(n(m^2N)^4K^2)$ worst case complexity of the fraction-free algorithm (Theorem 4.3), the modular algorithm is at least one order of magnitude faster. The modular algorithm provides a significant improvement when m or N is large. The same complexity applies if $\mathbb{D} = R[x]$ for some ring R in which multiplication can be performed in constant time. In that case, K is the maximum degree in x of all coefficients in $\mathbf{A}(z)$ and $\mathbf{B}(z)$. We also remark that our algorithm computes a solution to the extended GCLD problem (1), and the complexity is not reduced if only a GCLD is needed.

Example 7.2 *Let $\mathbf{A}(z)$ and $\mathbf{B}(z)$ be the those given in Example 4.4. We saw that $\vec{v} = [5, 4, 3, 2]$. When $p = 2$, FF_ReduceModp returns $\vec{v}_2 = [3, 3, 2, 2]$, and when $p = 3$, FF_ReduceModp returns $\vec{v}_3 = [3, 2, 2, 2]$. These two primes are unlucky because they both divide $d^*(\vec{v})$. Since $|\vec{v}_2| \neq |\vec{v}_3|$, both are considered unlucky and their corresponding results are discarded. The prime $p = 5$ is lucky. However, $\vec{v}_7 = [4, 3, 3, 2]$ and so $p = 7$ is unlucky as $|\vec{v}_7| \neq |\vec{v}|$. Since $|\vec{v}_5| \neq |\vec{v}_7|$, both results are discarded. The primes $p = 11, 13, \dots, 37$ are all lucky, and we can terminate the algorithm because $\mathbf{F}(z) \cdot \mathbf{M}(z) = \mathbf{R}(z)$. The computed results are the same as in Example 4.4.* □

8 Experimental Results

Both the fraction-free and modular extended GCLD algorithms have been implemented in Maple 7. Instead of using primes of size K, we chose primes that are half the machine word size, so that modular arithmetic can be performed efficiently. We used the `modp1` representation for polynomials for the efficient implementation of the modular algorithm.

We now present some experimental results to support the complexity analysis given in Theorem 7.1. We chose $m = n_1 = n_2$ in our experiments. In the first set of experiments, we chose various values of m, n_1, n_2, N, and κ, and generated $\mathbf{A}(z)$ and $\mathbf{B}(z)$ randomly. In these cases, $\mathbf{A}(z)$ and $\mathbf{B}(z)$ are usually left coprime. The experimental results are presented in Table 1. In the second set of experiments, we randomly generated $\mathbf{C}(z)$ of degree d. We generated $\mathbf{A}(z)$ and $\mathbf{B}(z)$ by multiplying $\mathbf{C}(z)$ on the left to random matrix polynomials of degree $N - d$. This allows us to have some control over the degree of the computed GCLD[1]. The results in the second set of experiments are presented in Table 2. We see that as m, n_1, n_2, N, and κ increases, the advantage of the modular algorithm over the fraction-free algorithm becomes clear. Moreover, this advantage is also apparent when

[1]To precisely control the degree, we would have to control the degree of $\det \mathbf{C}(z)$.

m, n_1, n_2	N	κ	Size	FF_GCLD (s)	Mod_GCLD (s)	Ratio
1	20	10^4	302	1.53	3.93	0.39
1	40	10^4	599	14.21	17.63	0.81
1	60	10^4	905	61.36	44.82	1.37
1	20	10^9	699	5.12	9.82	0.52
1	40	10^9	1428	86.58	47.70	1.82
1	60	10^9	1533	114.58	60.04	1.91
2	20	10^4	622	28.16	79.10	0.36
2	40	10^4	1210	413.55	342.67	1.21
2	60	10^4	1680	1511.62	940.47	1.61
2	20	10^9	1507	158.79	231.82	0.68
2	40	10^9	2751	1664.87	827.86	1.99
2	60	10^9	3933	6432.79	2191.53	2.94

Table 1: Comparison of FF_GCLD and Mod_GCLD for various values of m, n_1, n_2, N, and κ. Also shown is the size (in number of decimal digits) of the largest coefficient in the result.

d	Size	FF_GCLD (s)	Mod_GCLD (s)	Ratio
1	3858	6895.10	2315.51	2.98
10	3213	3381.31	1432.72	2.36
20	2719	2200.21	1130.49	1.95
40	1425	212.83	256.36	0.83

Table 2: Comparison of FF_GCLD and Mod_GCLD for various values of d with $m = n_1 = n_2 = 2$, $N = 60$ and $\kappa = 10^9$. Also shown is the size (in number of decimal digits) of the largest coefficient in the result.

the degree of $\mathbf{C}(z)$ is small. These conditions encourage coefficient growth, and so the modular algorithm is significantly better in these cases. For small values of these parameters, the modular algorithm is slower because of the additional overhead performed.

9 Conclusions and Future Work

In this paper we developed a modular algorithm for computing one-sided GCDs for matrix polynomials. Our contribution includes the definition of lucky primes and the normalization of the matrix GCLDs, as well as determining the number of images required. The algorithm is significantly faster than the fraction-free algorithm, as shown by both theoretical analysis and experimental data. Our algorithm also solves the extended one-sided GCD problem, and can be used to transform any matrix polynomial into column reduced form.

The GCLD computed by our algorithm is column reduced, but unfortunately, there are infinitely many GCLDs in column reduced form. The GCLD computed is normalized in the sense that the computed Mahler system is normalized as the Cramer solutions to the associated linear systems of equations. This means that we normalize the cofactors $\mathbf{S}(z)$, $\mathbf{T}(z)$, $\mathbf{U}(z)$, $\mathbf{V}(z)$ instead of the computed GCLD. We intend to investigate algorithms which directly compute a GCLD in a normal form such as the Hermite form or the Popov form [13]. We would also like to investigate modular algorithms computing only a GCLD without the cofactors. This is especially useful in the cases in which a GCLD with small coefficients exists. It may also be possible to develop early termination criteria such that the reconstructed result is provably correct without reaching the Hadamard's bound or performing explicit verification [15].

There are also a number of interesting extensions to our modular algorithm. It is possible to apply the same technique to the rational matrix interpolation problem in [4] to obtain a modular algorithm. It is also of interest to extend this algorithm for matrices of Ore polynomials. This may generalize the modular algorithm by Li [16] for two Ore polynomials.

References

[1] E. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comp.*, 22:565–578, 1968.

[2] B. Beckermann, S. Cabay, and G. Labahn. Fraction-free computation of matrix Padé systems. In *Proceedings of ISSAC 1997*, pages 125–132. ACM Press, 1997.

[3] B. Beckermann and G. Labahn. Effective computation of rational approximants and interpolants. *Reliable Computing*, 6(365–390), 2000.

[4] B. Beckermann and G. Labahn. Fraction-free computation of matrix rational interpolants and matrix GCDs. *SIAM J. Matrix Anal. and Appl.*, 22(1):114–144, 2000.

[5] B. Beckermann and G. Labahn. On the fraction-free computation of column-reduced matrix polynomials via FFFG. Technical Report ANO436, Laboratoire ANO, University of Lille, 2001. Available at `http://ano.univ-lille1.fr/pub/2001/ano436.ps.Z`.

[6] R. R. Bitmead, S. Y. Kung, B. D. O. Anderson, and T. Kailath. Greatest common divisors via generalized Sylvester and Bezout matrices. *IEEE Trans. Automat. Contr.*, AC–23:1043–1046, 1978.

[7] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *Journal of the ACM*, 18(4):478–504, October 1971.

[8] A. Bultheel and M. van Barel. A matrix Euclidean algorithm and the matrix Padé approximation problem. In C. Brezinski, editor, *Continued Fractions and Padé Fractions*. Elsevier, North-Holland, 1990.

[9] B. W. Char, K. O. Geddes, and G. H. Gonnet. GCDHEU: Heuristic polynomial GCD algorithm based on integer GCD computation. *Journal of Symbolic Computation*, 9:31–48, 1989.

[10] G. E. Collins. Subresultants and reduced polynomial remainder sequences. *Journal of the ACM*, 14(1):128–142, January 1967.

[11] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[12] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

[13] T. Kailath. *Linear Systems.* Prentice-Hall, 1980.

[14] G. Labahn and S. Cabay. Matrix Padé fractions and their computation. *SIAM Journal of Computing*, 18:639–657, 1989.

[15] G. Labahn and S. Cabay. A modular algorithm where all primes are lucky. *Submitted to Journal of Symbolic Computation*, 2001. 25 pages.

[16] Z. Li. *A Subresultant Theory for Linear Differential, Linear Difference and Ore Polynomials, with Applications.* PhD thesis, RISC-Linz, Johannes Kepler University, Linz, Austria, 1996.

[17] Z. Li and I. Nemes. A modular algorithm for computing greatest common right divisors of ore polynomials. In *Proceedings of ISSAC 1997*, pages 282–289, 1997.

[18] J. D. Lipson. *Elements of Algebra and Algebraic Computing.* Addison-Wesley, 1981.

[19] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.