A BLAS based implementation of nonsingular rational system solving

by

Lawrence G. Barrett

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisors: Ilias Kotsireas and Arne Storjohann

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

We describe an optimized implementation of the linear X-adic lifting presented by Moenck and Carter (1979) to compute the solution to a rational system involving polynomial matrices with coefficients from the field of integers modulo a word size prime p. We optimize the algorithm by reducing polynomial matrix operations to Basic Linear Algebra Subroutine (BLAS) operations on scalar matrices which increases the algorithm's efficiency. As an application, we show how to use the optimized lifting algorithm to solve rational systems involving integer polynomial matrices.

Table of Contents

List of Tables

Chapter 1

Introduction

We develop an algorithm for solving a rational system filled with univariate polynomials with coefficients from the field of integers modulo a word size prime p. We denote the field of integers modulo p by $\mathbb{Z}_p$, and we let $\mathbb{Z}_p[x]$ denote the set of polynomials whose coefficients are elements of $\mathbb{Z}_p$. Our algorithm solves the rational system $Av = db$ where $A \in \mathbb{Z}_p[x]^{n \times n}$ and $b \in \mathbb{Z}_p[x]^{n \times m}$. We solve for a solution pair (v, d) such that $v \in \mathbb{Z}_p[x]^{n \times m}$ and $d \in \mathbb{Z}_p[x]$. For example, the following rational system with $p = 113$,

$$\underbrace{\begin{bmatrix} 92x^2 + 110x + 44 & 95x^2 + 5x + 97 & 58x^2 + 43x + 99 \\ 37x^2 + 68x + 26 & 95x^2 + 16x + 103 & 100x^2 + 89x + 33 \\ 17x^2 + 51x + 55 & 42x^2 + 82x + 24 & 89x^2 + 92x + 59 \end{bmatrix}}_{A \in \mathbb{Z}_{113}[x]^{3\times 3}} v = d \underbrace{\begin{bmatrix} 92x^2 + 13x + 49 \\ 46x^2 + 7x + 105 \\ 45x^2 + 43x + 8 \end{bmatrix}}_{b \in \mathbb{Z}_{113}[x]^{3\times 1}},$$

which we will use as a running example throughout the paper, has solution

$$v = \begin{bmatrix} 13x^6 + 30x^5 + 4x^4 + 30x^3 + 75x^2 + 112x + 73 \\ 101x^6 + 58x^5 + 39x^4 + x^3 + 8x^2 + 82x + 84 \\ 24x^6 + 110x^5 + 57x^4 + 100x^3 + 84x^2 + 35x + 47 \end{bmatrix} \in \mathbb{Z}_{113}[x]^{3\times 1}$$
$$d = x^6 + 98x^5 + 43x^4 + 106x^3 + 52x^2 + 72x + 27 \in \mathbb{Z}_{113}[x]$$

Notice that $\max(\deg v, \deg d) = 6 = 3 \max(\deg A, \deg b)$. In general, the degrees of polynomials in the solution (v, d) will grow linearly with the size and maximum degree of the inputs, where $\max(\deg v, \deg d) = n \max(\deg A, \deg b)$.

n	$\max(\deg A, \deg b)$	$\max(\deg v, \deg d)$
100	10	1000
100	20	2000
1000	10	10000
1000	20	20000

Table 1.1: Degrees of Variables in the Rational System

Much research has already been done on the subject of rational system solving [4, 5, 6, 7, 8], mostly presented for the case where $A \in \mathbb{Z}^{n\times n}$ and $b \in \mathbb{Z}^{n\times m}$, which is analogous to the polynomial case. Our goal is to present an algorithm which solves the rational system efficiently in the polynomial case.

Solving a rational system problem can be difficult because its solution can have very large degree even for small degree inputs, as we can see from Table 1.1. To avoid computation with large degree polynomials, we will follow the definition of linear X-adic lifting presented in [4] and [7], which is the main procedure in our algorithm. The linear lifting procedure we will use is based on a linear X-adic lifting algorithm presented by Moenck and Carter (1979) in [4]. All operations in the Moenck and Carter algorithm bound the degrees of the polynomials, with which we operate in the lifting procedure, by the maximum of the degrees of the inputs A and b, $\max(\deg A, \deg b)$. Linear lifting requires the use of a modulus relatively prime to $\det A$. In the polynomial case, we use as modulus a polynomial $X \in \mathbb{Z}_p[x]$. We can pick any arbitrary polynomial X as long as X is relatively prime to $\det A$ and $\deg X \geq \deg A$. The polynomial we will use in our algorithm will be constructed by randomly picking points α_i in our field $\mathbb{Z}_p$ and constructing the polynomial

$$X \equiv (x-\alpha_1)(x-\alpha_2)(x-\alpha_3)\cdots(x-\alpha_{\deg X}) \pmod{p}$$

that satisfies the aforementioned conditions, where the degree of X is given as input.

We perform X-adic lifting on $A^{-1}b$ which gives us the following X-adic representation of $A^{-1}b$:

$$A^{-1}b = z_0 + z_1X + z_2X^2 + \ldots$$

We can stop the lifting procedure at some precision $k \in \mathbb{Z}_{>0}$ to get

$$A^{-1}b \equiv z_0 + z_1X + z_2X^2 + \cdots + z_{k-1}X^{k-1} \mod X^k,$$

where each $z_i \in \mathbb{Z}_p[x]^{n\times m}$ and $z_i = z_i \bmod X$ which implies $\deg z_i < \deg X$ for all i, $0 \leq i < k$. Using the A and b from our running example and $X = (x-76)(x-58) \equiv$

$x^2 + 92x + 1 \mod 113$, we have the following:

$$A^{-1}b = \begin{bmatrix} 98x+96 \\ 54x+48 \\ 8x+40 \end{bmatrix} + \begin{bmatrix} 108x+56 \\ 26x+62 \\ 75x+9 \end{bmatrix} X + \begin{bmatrix} 12x+49 \\ 61x+54 \\ x+59 \end{bmatrix} X^2 + \begin{bmatrix} 24x+110 \\ 13x+88 \\ 43x+76 \end{bmatrix} X^3 +$$
$$+ \begin{bmatrix} 97x+87 \\ 45x+66 \\ 4x+87 \end{bmatrix} X^4 + \begin{bmatrix} 83x+8 \\ 40x+94 \\ 53x+14 \end{bmatrix} X^5 + \begin{bmatrix} 35x+15 \\ 31x+43 \\ 45x+99 \end{bmatrix} X^6 \mod X^7$$

Every lifting step increases the power of the modulus, and once we have lifted high enough, our congruence becomes equality, thus giving us the solution to the rational system.

In Chapter 2, we define some notation and recall the theory of linear X-adic lifting. We then recall the standard implementation of linear X-adic linear lifting for nonsingular rational system solving in Chapter 3. The algorithm we present in Chapter 3 is heavily based on the RationalSolver algorithm presented in [5]. In the standard algorithm, all matrix operations are computed using the Maple package "modp1", which efficiently performs operations on univariate polynomials whose coefficients are integers modulo $p \in \mathbb{Z}$. We call these polynomials "modp1 polynomials".

In Chapter 4, we develop an optimized version of the standard linear X-adic lifting presented in Chapter 3. We show how to reduce polynomial matrix operations to BLAS operations on scalar matrices modulo p. In addition, we introduce another polynomial Y which is used in the lifting phase to decrease the cost from lifting with X, an idea presented in [7]. Run time comparisons between the standard and optimized algorithms are presented to show the increase in efficiency of the optimized algorithm.

In Chapter 5, we present an application of the optimized lifting algorithm for solving the more general system where $A \in \mathbb{Z}[x]^{n \times n}$ and $b \in \mathbb{Z}[x]^{n \times m}$. With the optimized algorithm, we can produce many images of our solution modulo different primes, and then use a combination of Chinese remaindering and rational reconstruction to obtain the solution pair (v, d).

Finally, we present our conclusions and areas for future work in Chapter 6.

Chapter 2

Lifting

We start by introducing some notation before we proceed. For any polynomials $q, X \in \mathbb{Z}_p[x]$ we let $\text{Rem}\,(q,\, X)$ be the unique polynomial congruent to q modulo X with degree less than $\deg X$. In other words, $\text{Rem}\,(q,\, X) \equiv q \mod X$ and $\deg(\text{Rem}\,(q,\, X)) < \deg X$. We can extend Rem operation to any matrix $Q \in \mathbb{Z}_p[x]^{m\times n}$ by letting $\text{Rem}\,(Q,\, X)$ denote the matrix obtained from applying Rem on each entry in Q.

We let $A \in \mathbb{Z}_p[x]^{n\times n}$ be non-singular and $X \in \mathbb{Z}_p[x]$ such that X is relatively prime to $\det A$ (denoted by $X \perp \det A$). The linear lifting used to compute $\text{Rem}\,(A^{-1},\, X^i)$ up to some precision i is similar to the lifting discussed in [7] for $A \in \mathbb{Z}^{n\times n}$ and $X \in \mathbb{Z}_{>2}$. In this chapter we will reiterate the theorems and results from [7] for the polynomial case, which is analogous to the integer case.

2.1 Linear Lifting

The following is the standard algorithm used for computing the X-adic expansion of A^{-1} as presented in [7]:

$C_0 := \text{Rem}\,(A^{-1},\, X)$;
$R_1 := (1/X)(I - AC_0)$;
for $i = 1$ **to** $k - 1$ **do**
    $C_i := \text{Rem}\,(C_0 R_i,\, X)$;
    $R_{i+1} := (1/X)(R_i - AC_i)$;
od;

Linear X-adic lifting is based on the identity

$$\begin{aligned} A^{-1} &= C_0 + C_1 X + \cdots + C_{i-1} X^{i-1} + A^{-1} R_i X^i \\ &= \operatorname{Rem}\left(A^{-1},\, X^i\right) + A^{-1} R_i X^i \end{aligned}$$

where R_i is equal to

$$R_i = (1/X^i)(I - A\operatorname{Rem}\left(A^{-1},\, X^i\right))$$

As described in [7], the "essence of linear X-adic lifting is that C_i and R_{i+1} can be computed using only A, R_i, and C_0: The other coefficients of the X-adic expansion of A^{-1} are not required." We present the following theorem from [7] which "captures this essential idea of linear X-adic lifting," modified for the polynomial case.

Theorem 1 ([7, Theorem 3]) *Let $A \in \mathbb{Z}_p[x]^{n\times n}$ be nonsingular and $X \in \mathbb{Z}_p[x]$ be relatively prime to $\det A$. If $B, R \in \mathbb{Z}_p[x]^{n\times n}$ satisfy*

(i) $A^{-1} = B + A^{-1} R X^k$

for some $k > 0$, then for any $M \in \mathbb{Z}_p[x]^{n\times n}$ such that $M \equiv A^{-1}R \pmod{X^l}$ for some $l > 0$, we have

(ii) $A^{-1} = B + MX^k + A^{-1} R' X^{k+1}$

where $R' = (1/X^l)(R - AM)$. In particular, we can choose $l = 1$ and $M = \operatorname{Rem}(A^{-1}R,\, X)$.

Chapter 3

Standard X-adic Lifting

The goal for the standard modularized X-adic lifting algorithm is to make use of the X-adic expansion representation of our polynomial matrices and keep them in this representation for most of our matrix operations. To represent the X-adic expansion of a polynomial, we only store the coefficients of the expansion in a table where the i-th term in the table denotes the coefficient of X^i. We can extend X-adic expansion to matrices where the i-th matrix contains the coefficients of X^i for the X-adic expansion of their respective polynomial entries. We will be working with the modp1 Maple package to efficiently multiply polynomials with coefficients modulo p in our matrix operations, as described in the introduction.

The algorithm used in this section is based on the RationalSolver algorithm presented in [5]. We implement the modified algorithm for the case where $A \in \mathbb{Z}_p[x]^{n\times n}$, $b \in \mathbb{Z}_p[x]^{n\times m}$, and $X \in \mathbb{Z}_p[x]$ such that $X \perp \det A$. The computation costs of the algorithm have already been analyzed in [5] for both the integer and the polynomial case.

RationalSolver computes the X-adic expansion of $A^{-1}b$ up to the following precision from [5]:

$$\text{LiftingBound}(N, D) = 2ND$$

where N and D are pre-computed bounds for the numerator and denominator of our solution, which are computed using n, $\deg A$, and $\deg b$. However, it is possible to compute the solution to our system after fewer lifting steps than suggested by $\text{LiftingBound}(N, D)$. For simplicity, we only consider one bound for both the numerator and denominator, which we denote by N.

Our algorithm takes as input A and b for the polynomial case, and initializes a polynomial X such that $X \perp \det A$. We wish to find a solution pair (v, d) where $v \in \mathbb{Z}_p[x]^{n\times m}$

and $d \in \mathbb{Z}_p[x]$ such that $Av = db$. We use linear lifting to compute the coefficients z_i in the X-adic expansion of $A^{-1}b = z_0 + z_1X + z_2X^2 + \ldots$ up to some precision. Each lifting step increases the precision incrementally, thus for arbitrary precision k we define $z^{(k)} = z_0 + z_1X + z_2X^2 + \cdots + z_{k-1}X^{k-1}$ where $z^{(k)} \equiv \text{Rem}\left(A^{-1}b,\, X^k\right)$. When k is large enough, we have the equality $A^{-1}b = z^{(k)} \mod X^k$ and we can recover our solution pair (v, d) with rational reconstruction [10, Section 5.7]. We want to stop performing lifting steps at some precision j and attempt to recover $(v,\, d)$ from $z^{(j)}$ by making the algorithm output sensitive on the rational reconstruction of $z^{(j)}$. If we fail, we perform more lifting steps and attempt to recover the solution again until rational reconstruction is successful.

Attempting to recover the solution by applying rational reconstruction on all entries of $z^{(j)}$ after every lifting phase would add too much cost to the algorithm. Instead, we use a randomization technique similar to the one used for finding the gcd of many polynomials in [10, Section 6.9]. We start by taking a random combination $s_i = L_i z_i R_i$, where $L_i \in \mathbb{Z}_p^{1\times n}$ and $R_i \in \mathbb{Z}_p^{m\times 1}$ are random integer matrices, for $0 \le i < j$. After computing $s^{(j)} = s_0 + s_1X + \cdots + s_{j-1}X^{j-1} \mod X^J$, which in our Maple implementation is stored as a modp1 polynomial, we can then perform rational reconstruction with the current numerator and denominator bound, N_c, which we update after every lifting phase. The theorem [10, Theorem 6.46] tells us that given a field $\mathbb{Z}_p$ with many elements, or in other words a large prime p, there is a high probability that we will reconstruct our solution pair (v, d) if the rational reconstruction on $s^{(j)}$ is successful.

The pseudocode for the algorithm is presented as follows:

RatSolve$(A, b, p, d_X, N_0, steps)$
Inputs: $A \in \mathbb{Z}_p[x]^{n\times n}$, $b \in \mathbb{Z}_p[x]^{n\times m}$, and $d_X, N_0, steps \in \mathbb{Z}$
with $d_X, steps > 0$ and $N_0 \ge 0$.
Outputs: $v \in \mathbb{Z}_p[x]^{n\times m}$ and $d \in \mathbb{Z}_p[x]$ such that $A(1/d)v = b$.
Condition: $d_X \ge \deg A$.
(1) [Initialize:]
$X, B :=$ LiftInit(A, x, n, p, d_X);
$N :=$ Bound(A, b);
$k_0 :=$ InitialLifting(A, b, N_0, d_X);
$b_{lim} :=$ MaxPrecision(b, d_X);
$c :=$ XadicExpansion(b, X);
(2) [Initial Lifting:]
$k_n := k_0$;
$N_c := N_0$;
$\bar{c} := 0^{n\times m}$;

$z := \text{LinearLift}(A, B, c, \bar{c}, X, 0, k_n)$;
$d_v := \infty$;
(3) [Output Sensitive Rational Reconstruction:]
while $d_v > N_c$ **do**
$k_c := k_n$;
$k_n := k_n + steps$;
$N_c := \text{CurrentBound}(k_n, d_X)$;
$z := \text{LinearLift}(A, B, c, \bar{c}, X, k_c, k_n)$;
$u := \text{randomize}(z)$;
$d := \text{RationalReconstruction}(u)$;
if rational reconstruction fails **then**
$d_v := \infty$;
next;
else
$$v := \text{Rem}\left(d\left(\sum_{j=0}^{k_n-1} z[i]X^i\right),\ X^{k_n}\right);$$
$d_v := \deg v$;
fi;
od;
return v, d;

Although we have omitted some of the details of the algorithm, the above pseudocode describes its general process. For instance, it is important to note that the randomization technique we discussed above for attempting to reconstruct our solution from a random combination of entries in z is performed by the randomize(z) function.

The other functions presented in the algorithm are described below:

- $\text{LiftInit}(A, x, n, p, d_X) \rightarrow$ polynomial X and polynomial matrix B such that $B = \text{Rem}\,(A^{-1},\ X)$. The computation of B requires $X \perp \det A$.
- $\text{Bound}(A, b) \rightarrow \max\left((n-1)\deg A + \deg b\,,\ n \deg A\right)$ [5]
- $\text{InitialLifting}(A, b, N_0, d_X) \rightarrow \lceil \max\left(2N_0\,,\ N_0 + \deg A\,,\ N_0 + \deg b\right) \rceil$
- $\text{MaxPrecision}(b, d_X) \rightarrow \lceil \frac{\deg b + 1}{d_X} \rceil$
- $\text{XadicExpansion}(b, X) \rightarrow$ table c, such that $b = c[0] + c[1]X + \ldots c[k]X^k$

- CurrentBound$(k_n, d_X) \to \min\left(\lceil \frac{k_n d_X}{2} \rceil - 10\,,\, 1\right)$

The bound N_c produced by CurrentBound(k_n, d_X) has the property

$$2N_c << \deg M - 10$$

where $M = X^{k_n}$ is the modulus of the current X-adic expansion of $A^{-1}b$. This property ensures that if the lifting is not high enough (i.e., $N_c > \max(\deg v, \deg d)$), then the trail rational reconstruction will quickly report fail, avoiding the expensive cost of reconstructing v from all of z in the early lifting steps of the algorithm.

The procedure LinearLift is the main focus of the algorithm as it is where all the lifting takes place. The pseudocode for LinearLift is as follows:

LinearLift$(A, B, c, \bar{c}, X, k_c, k_n)$
Inputs: $A, B \in \mathbb{Z}_p[x]^{n\times n}$, $X \in \mathbb{Z}_p[x]$, $c[i] \in \mathbb{Z}_p[x]^{n\times m}$ for $0 \le i < b_{lim}$, and $k_c, k_n \in \mathbb{Z}_{\ge 0}$.
Outputs: $z[i] \in \mathbb{Z}_p[x]^{n\times m}$ for $0 \le i < k_n$ and $\bar{c} \in \mathbb{Z}_p[x]^{n\times m}$.

```
for i = k_c to k_n − 1 do
    if i < b_lim then
        c̄ := c̄ + c[i]
    fi;
    z[i] := Rem(Bc̄, X);
    c̄ := (1/X)(c̄ − Az[i]);
od;
return z;
```

We will show a couple steps of linear lifting given the inputs from our running example with $p = 113$ and $X = x^2 + 92x + 1$:

First we compute

$$B = \text{Rem}\left(A^{-1}, X\right) = \begin{bmatrix} 84x+71 & 110x+42 & 72x+23 \\ 57x+61 & 83x+85 & 78x+61 \\ 110x+59 & 33x+59 & 100x+31 \end{bmatrix}$$

Then we let table c contain the coefficients in the X-adic representation of b:

$$b = \underbrace{\begin{bmatrix} 24x+70 \\ 69x+59 \\ 84x+76 \end{bmatrix}}_{c[0]} + \underbrace{\begin{bmatrix} 92 \\ 46 \\ 45 \end{bmatrix}}_{c[1]} X \mod X^2$$

We let $\bar{c} := c[0]$. Then the first coefficient in the X-adic expansion of $A^{-1}b$ is

$$z[0] = \text{Rem}\,(B\bar{c},\, X) = \begin{bmatrix} 98x + 96 \\ 54x + 48 \\ 8x + 40 \end{bmatrix}$$

Using $z[0]$, we compute the correction term $\bar{c}$

$$\bar{c} := (1/X)(\bar{c} - Az[0]) = \begin{bmatrix} 80x + 112 \\ 49x \\ 100x + 98 \end{bmatrix}$$

Now we add the correction term to the next coefficient in the X-adic expansion of b

$$\bar{c} := \bar{c} + c[1] = \begin{bmatrix} 80x + 91 \\ 49x + 46 \\ 100x + 30 \end{bmatrix}$$

Then the second coefficient in the X-adic expansion of $A^{-1}b$ is

$$z[1] = \text{Rem}\,(B\bar{c},\, X) = \begin{bmatrix} 108x + 56 \\ 26x + 62 \\ 75x + 9 \end{bmatrix}$$

From these two lifting steps, we get the following congruence:

$$A^{-1}b \equiv z[0] + z[1]X \equiv \begin{bmatrix} 98x + 96 \\ 54x + 48 \\ 8x + 40 \end{bmatrix} + \begin{bmatrix} 108x + 56 \\ 26x + 62 \\ 75x + 9 \end{bmatrix} X \mod X^2$$

3.1 Condition on Degree of X

Let z be the table representing the X-adic expansion of $A^{-1}b$ and c be the table representing the X-adic expansion of b. To show that the condition $\deg X \geq \deg A$ must be satisfied for RatSolve to work, we must look at the procedure LinearLift which performs a specified number of lifting steps and has two outputs: the current coefficient in the X-adic expansion of $A^{-1}b$, stored in $z[i]$, and a correction term for the next coefficient in the X-adic expansion of b, denoted $\bar{c}$.

Suppose we obtain $\bar{c}$ from the previous lifting step $i-1$ (for $i=0$ we see from the RatSolve algorithm that $\bar{c} = c[0]$). Then for the next lifting step i we do the following:

$$\begin{aligned}
&\bar{c} := \bar{c} + c[i] \\
&z[i] := \mathrm{Rem}\,(B\bar{c},\, X) \\
&\bar{c} := (1/X)(\bar{c} - Az[i])
\end{aligned}$$

From the above operations we see $\deg \bar{z}[i] < \deg X$. Since $\bar{c} = c[0]$ in the first lifting step, for any other lifting step we have

$$\deg \bar{c} < \deg A + \deg X - \deg X = \deg A$$

If $\deg X < \deg A$ then the correction term would need to correct more than the next coefficient in c. Thus we add the condition $\deg X \geq \deg A$ to limit our correction to the next coefficient only.

Chapter 4

Optimized Linear Lifting by reduction to BLAS

Our next goal is to replace polynomial matrix operations in the standard algorithm with Level 2 and Level 3 BLAS operations. We rely on picking a polynomial X such that

$$X \equiv (x - \alpha_0)(x - \alpha_1) \cdots (x - \alpha_{d_X - 1}) \pmod{p},$$

where $\alpha_i \in \mathbb{Z}_p$ are distinct for all i, $0 \leq i \leq d_X - 1$ and $X \perp \det A$.

We use the roots $\{\alpha_0, \alpha_1, \ldots, \alpha_{d_X-1}\}$ of X to represent polynomial matrices in an "X basis representation". This is more easily seen through an example: we will show how we represent $A \in \mathbb{Z}_p[x]^{n \times n}$ in the X basis.

Let $A_{x=a}$ denote the matrix A with polynomial entries evaluated at a modulo p. For each root α_i of X, we compute $A_{x=\alpha_i}$, which is equivalent to computing $\text{Rem}\,(A,\, x - \alpha_i)$. We can then store these matrices in a table, denoted A_x such that $A_x[i] = A_{x=\alpha_i}$.

For example, let $p = 113$ and $X = (x - 76)(x - 58)(x - 15)$. Then for

$$A = \begin{bmatrix} 92x^2 + 110x + 44 & 95x^2 + 5x + 97 & 58x^2 + 43x + 99 \\ 37x^2 + 68x + 26 & 95x^2 + 16x + 103 & 100x^2 + 89x + 33 \\ 17x^2 + 51x + 55 & 42x^2 + 82x + 24 & 89x^2 + 92x + 59 \end{bmatrix}$$

we have

$$A_x[0] = A_{x=76} = \begin{bmatrix} 108 & 17 & 53 \\ 25 & 68 & 74 \\ 84 & 22 & 72 \end{bmatrix}, A_x[1] = A_{x=58} = \begin{bmatrix} 77 & 64 & 68 \\ 70 & 30 & 109 \\ 85 & 72 & 30 \end{bmatrix},$$
$$A_x[2] = A_{x=15} = \begin{bmatrix} 20 & 77 & 8 \\ 105 & 22 & 25 \\ 12 & 82 & 107 \end{bmatrix}$$

With the X basis representation, we can replace any polynomial matrix operation with BLAS operations involving integer matrices. Suppose we also had a matrix $B \in \mathbb{Z}_p[x]^{n \times n}$ represented in the X basis by the table B_x. Then,

$$\begin{aligned} (A^{-1})_x[\,i\,] &= (A_x[\,i\,])^{-1} \pmod{p} \\ (AB)_x[\,i\,] &= A_x[\,i\,]B_x[\,i\,] \pmod{p} \\ (A + \alpha B)_x[\,i\,] &= A_x[\,i\,] + \alpha B_x[\,i\,] \pmod{p} \qquad (\text{for } \alpha \in \mathbb{Z}) \end{aligned}$$

for $0 \leq i < d_X$.

By replacing polynomial matrix operations with integer matrix operations, we decrease the computational cost of the algorithm, especially since there are many polynomial matrix operations that occur in the lifting phase of the standard algorithm.

As a note for any timings we compute in this paper, we are running Maple 18 on a machine with the following specifications: Scale 5027R 2U Xeon Dual Socket Server, with 128G RAM, 4 × 2T drives, and OS Ubuntu 14.04 LTS AMD ALT 64bit.

We can show it is worthwhile to use the X basis representations by comparing the time it takes to compute the product AB of polynomial matrices to computing the product $A_x[i]B_x[i]$ for $0 \leq i < d_X$ where $A \in \mathbb{Z}_p[x]^{n \times n}$, $B \in \mathbb{Z}_p[x]^{n \times n}$, and $\deg A = \deg B = d_X = 10$:

n	AB	$A_x[i]B_x[i]$ for $0 \leq i < d_X$	Factor of Speedup
200	28 s	0.816 s	34.6
300	103 s	0.950 s	109
400	246 s	4.36 s	56.5
500	584 s	12.1 s	48.1

Table 4.1: Polynomial Matrix operations vs Level 3 BLAS operations (CPU time)

Thus we can see that, as we increase the dimension n, we save a lot of time on polynomial matrix operations by using their X basis representations.

4.1 Evaluation

The standard method for evaluation of polynomial matrices at some integer value is to evaluate each polynomial in the matrix at that value. However, we can reduce evaluation to a matrix-matrix multiplication. To efficiently evaluate polynomial matrices at each root of X, we require the use of Vandermonde matrices.

Definition 1 ([10, Section 5.2]) *Let $u_0, u_1, \ldots, u_k \in F$ for some field F and let f be a polynomial of degree less than or equal to n. Then we define the* Vandermonde Matrix V *to be*

$$V = VDM(u_0, \ldots, u_k) = \begin{bmatrix} 1 & u_0 & u_0^2 & \cdots & u_0^n \\ 1 & u_1 & u_1^2 & \cdots & u_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & u_k & u_k^2 & \cdots & u_k^n \end{bmatrix} \in F^{n \times n}$$

Suppose we have $Q \in \mathbb{Z}_p[x]^{m \times n}$ where $d_Q = \deg Q$, and we want to evaluate Q at each root of X. Let $S = \{\alpha_0, \alpha_1, \ldots, \alpha_{d_X}\}$ be the set of roots of X where $\alpha_i \in \mathbb{Z}_p$ for $0 \leq i \leq k$ and $d_X = \deg X$. We define $Q_i \in {\mathbb{Z}_p}^{m \times n}$ for $0 \leq i \leq d_Q$, such that

$$Q = Q_0 + Q_1 x + Q_2 x^2 + \ldots Q_{d_Q} x^{d_Q}$$

Thus each Q_i represents the n by m matrix of coefficients of the term x^i of the polynomials in Q.

We will use a Vandermonde matrix to perform evaluation at each point in S. Since each polynomial in Q has degree less than or equal to d_Q, the Vandermonde matrix of the set of points S, which we will denote V_x, is defined to be

$$V_x = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{d_Q} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{d_Q} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_{d_X} & \alpha_{d_X}^2 & \cdots & \alpha_{d_X}^{d_Q} \end{bmatrix} \in \mathbb{Z}_p^{d_X \times (d_Q+1)}$$

If we were to attempt evaluation by using each Q_i separately, we would need to perform many scalar multiplications of matrices and matrix additions. However, by reshaping each Q_i into vectors and assigning each vector to a row, we can perform evaluation as a single matrix multiplication. To visualize this, let us consider the case where $n = 1$ so that Q and Q_i for all i, $0 \leq i \leq d_Q$ are column vectors. Then the matrix Q' we construct from all Q_i's is

$$Q' = \begin{bmatrix} {Q_0}^T \\ {Q_1}^T \\ \vdots \\ {Q_{d_Q}}^T \end{bmatrix} \in \mathbb{Z}_p^{(d_Q+1)\times nm}$$

We can think of each column j of Q' as the column vector of coefficients of the j-th entry polynomial in Q. The evaluation step becomes

$$V_x Q' = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{d_Q} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{d_Q} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_{d_X} & \alpha_{d_X}^2 & \cdots & \alpha_{d_X}^{d_Q} \end{bmatrix} \begin{bmatrix} {Q_0}^T \\ {Q_1}^T \\ \vdots \\ {Q_{d_Q}}^T \end{bmatrix} \in \mathbb{Z}_p^{d_X \times nm}$$

Where $Q_x[i]$ is obtained by reshaping row i of the above matrix multiplication into an n by m matrix.

Using our running example we can split our matrix A into integer matrices for the coefficients of distinct terms as follows:

$$A = \underbrace{\begin{bmatrix} 44 & 97 & 99 \\ 26 & 103 & 33 \\ 55 & 24 & 59 \end{bmatrix}}_{A_0} + \underbrace{\begin{bmatrix} 110 & 5 & 43 \\ 68 & 16 & 89 \\ 51 & 82 & 92 \end{bmatrix}}_{A_1} x + \underbrace{\begin{bmatrix} 92 & 95 & 58 \\ 37 & 95 & 100 \\ 17 & 42 & 89 \end{bmatrix}}_{A_2} x^2$$

Using the A_i matrices, we can construct the matrix

$$A' = \begin{bmatrix} 44 & 26 & 55 & 97 & 103 & 24 & 99 & 33 & 59 \\ 110 & 68 & 51 & 5 & 16 & 82 & 43 & 89 & 92 \\ 92 & 37 & 17 & 95 & 95 & 42 & 58 & 100 & 89 \end{bmatrix}$$

Since $X = (x - 76)(x - 58)(x - 15)$ we have the Vandermonde matrix

$$V_x = \begin{bmatrix} 1 & 76 & 76^2 \\ 1 & 58 & 58^2 \\ 1 & 15 & 15^2 \end{bmatrix} \equiv \begin{bmatrix} 1 & 76 & 13 \\ 1 & 58 & 87 \\ 1 & 15 & 112 \end{bmatrix} \pmod{113}$$

Then we multiply on the right by V_x to get

$$V_x A' = \begin{bmatrix} 108 & 25 & 84 & 17 & 68 & 22 & 53 & 74 & 72 \\ 77 & 70 & 85 & 64 & 30 & 72 & 68 & 109 & 30 \\ 20 & 105 & 12 & 77 & 22 & 82 & 8 & 25 & 107 \end{bmatrix} \pmod{113}$$

where the first row can be reshaped to give $A_{x=76}$, the second row can be reshaped to give $A_{x=58}$, and the third row can be reshaped to give $A_{x=15}$.

Thus we have reduced the process of evaluation to a single matrix multiplication. We can use a similar process for interpolation by starting with the X basis representation, creating the bigger matrix where each row is a reshaping of a basis matrix, and then multiplying on the right by $(V_x)^{-1}$.

The evaluation and interpolation processes described here are useful for efficiently converting from polynomial matrices to their X basis representations for linear X-adic lifting, and back to polynomial matrices for rational reconstruction.

4.2 Y Basis

Following an idea presented in [7], we can further increase the efficiency of RatSolve by introducing a new polynomial Y with $d_Y = \deg Y$, where

$$Y = (x - \beta_0)(x - \beta_1) \cdots (x - \beta_{d_Y - 1}) \pmod{p}$$

such that $\beta_i \in \mathbb{Z}_p$ are distinct for all i, $0 \leq i \leq d_Y - 1$.

Using the same argument for showing $\deg X \geq \deg A$ for the polynomial X from before, we also have $\deg Y \geq \deg A$. However, unlike the polynomial X, we do not require $Y \perp \det A$ since we are not computing $\text{Rem}\,(A^{-1}b, Y)$ and A is already nonsingular modulo X. Instead we will be using Y to convert between X basis representation to Y basis representation, specifically for use in the lifting phase of RatSolve. The reasoning behind implementing a change of basis is due to the fact that, depending on our choices for X and Y, performing the linear lifting in the Y basis may be less costly than performing it in the X basis.

Since we are working with polynomial matrices where $\deg A \leq \deg X$ and $\deg A \leq \deg Y$, all the polynomials we deal with in A have degree less than or equal to $\max(\deg X, \deg Y)$. Assuming $\deg X \geq \deg Y$ (which is not a necessary condition) let us define a few Vandermonde matrices we will use in the algorithm. Let the sets $\{\alpha_0, \alpha_1, \ldots, \alpha_{d_X - 1}\}$ and $\{\beta_0, \beta_1, \ldots, \beta_{d_Y - 1}\}$ be the roots of X and Y respectively. Then

- $\left.\begin{array}{l} V_x = \mathrm{VDM}(\alpha_0, \alpha_1, \ldots, \alpha_{d_X-1}) \in \mathbb{Z}_p[x]^{d_X \times d_X} \\ V_y = \mathrm{VDM}(\beta_0, \beta_1, \ldots, \beta_{d_Y-1}) \in \mathbb{Z}_p[x]^{d_Y \times d_Y} \end{array}\right\}$ Square matrices
- $\left.\begin{array}{l} V_{xy} = \mathrm{VDM}(\alpha_0, \alpha_1, \ldots, \alpha_{d_X-1}) \in \mathbb{Z}_p[x]^{d_X \times d_Y} \\ V_{yx} = \mathrm{VDM}(\beta_0, \beta_1, \ldots, \beta_{d_Y-1}) \in \mathbb{Z}_p[x]^{d_Y \times d_X} \end{array}\right\}$ Non-square matrices

We switch between the X basis and the Y basis through the use of Vandermonde matrices $V_{x\to y}$ and $V_{y\to x}$ which are defined by

$$V_{x\to y} = V_{yx}(V_x)^{-1} \in \mathbb{Z}_p[x]^{d_Y \times d_X} \text{ and } V_{y\to x} = V_{xy}(V_y)^{-1} \in \mathbb{Z}_p[x]^{d_X \times d_Y}$$

By definition of Vandermonde matrices, we are guaranteed that V_x and V_y are nonsingular since all the roots of X and all the roots of Y are distinct. Using the same technique for evaluation and interpolation, we can change bases using $V_{x\to y}$ and $V_{y\to x}$ with a single matrix multiplication. Thus switching between bases is reduced to integer matrix multiplication modulo p.

The pseudocode for the optimized version of the algorithm, called RatModSolve, is similar to the pseudocode for RatSolve where polynomial matrix operations are replaced with their equivalent X and Y basis operations, and we convert from X basis representation to polynomial matrix representation for the rational reconstruction.

4.3 Optimal degree of X

Now that we have introduced a second polynomial Y to perform linear X-adic lifting, we want $\deg Y$ to be as small as possible for more efficient lifting. Since $\deg Y \geq \deg A$, we let $\deg Y = \deg A$. Different choices for $\deg X$ give different timings on our optimized algorithm. This is because a larger $\deg X$ gives us a larger degree modulus X^k for any precision k in the lifting procedure of our algorithm, which results in a decrease in the amount of lifting needed to get to a solution. However, there is a trade off since we also need to compute $\deg X$ inverses to get the X basis for $B = \mathrm{Rem}\,(A^{-1},\, X)$, denoted by B_x.

In general, the optimal $\deg X$ should be a function of n, m, $\deg A$, and $\deg b$. For simplicity, we attempt to find the best $\deg X$ as a multiple of $\deg A$, and we hope that this is close enough to the optimal $\deg X$ given the inputs of the rational system. By running a few test instances for varying n and $\deg A$, we produce the following tables:

n	$\deg A$	$\deg X$	Optimized Algorithm (CPU time)
200	5	5	18.2 s
200	5	7	17.1 s
200	5	9	18.3 s
200	**5**	**11**	**16.9 s**
200	5	13	19.1 s
200	5	15	19.8 s
200	**5**	**17**	**16.9 s**
200	5	19	18.6 s
200	5	21	18.0 s
200	5	23	20.2 s
200	5	25	24.2 s
200	5	27	22.2 s
200	5	29	27.2 s
200	5	31	28.3 s

Table 4.2: Timings for several $\deg X$ for $n = 200$ and $\deg A = 5$

n	$\deg A$	$\deg X$	Optimized Algorithm (CPU time)
300	5	5	46.8 s
300	5	7	39.2 s
300	5	9	38.5 s
300	5	11	39.8 s
300	5	13	39.8 s
300	**5**	**15**	**35.3 s**
300	**5**	**17**	**36.5 s**
300	5	19	46.3 s
300	5	21	44.3 s
300	5	23	41.9 s
300	5	25	43.0 s
300	5	27	43.9 s
300	5	29	47.5 s
300	5	31	55.1 s

Table 4.3: Timings for several $\deg X$ for $n = 300$ and $\deg A = 5$

n	$\deg A$	$\deg X$	Optimized Algorithm (CPU time)
200	50	50	525 s
200	50	70	475 s
200	**50**	**90**	**475 s**
200	50	110	521 s
200	50	130	513 s
200	50	150	503 s
200	**50**	**170**	**440 s**
200	50	190	516 s
200	50	210	571 s
200	50	230	483 s
200	50	250	550 s
200	50	270	575 s
200	50	290	617 s
200	50	310	696 s

Table 4.4: Timings for several $\deg X$ for $n = 200$ and $\deg A = 50$

n	$\deg A$	$\deg X$	Optimized Algorithm (CPU time)
800	5	5	685 s
800	5	7	510 s
800	5	9	394 s
800	5	11	378 s
800	5	13	357 s
800	**5**	**15**	**353 s**
800	**5**	**17**	**355 s**
800	5	19	366 s
800	5	21	376 s
800	5	23	357 s
800	5	25	405 s
800	5	27	374 s
800	5	29	408 s
800	5	31	407 s

Table 4.5: Timings for several $\deg X$ for $n = 800$ and $\deg A = 5$

In the above tables, we have highlighted the two best CPU times. We see that for small n and small $\deg A$ in Table 4.2 and Table 4.3, choosing $\deg X$ to be about 2 or 3 times $\deg A$ gives lower CPU times. In Table 4.4 we increase $\deg A$, and the best CPU times seem to occur again with $\deg X$ equaling 2 or 3 times $\deg A$, at least for small n. In Table 4.5, we increase n and see that $\deg X$ roughly 3 times $\deg A$ gives the best CPU times.

The results from the tables suggest that $\deg X$ about 3 times $\deg A$ should give lower CPU times for our algorithm. However, there is still a little discrepancy on the affect the size of $\deg X$ has on our algorithm for different size and degree inputs. This suggests that the optimal $\deg X$ is affected by more than just $\deg A$, but, for the sake of comparing the efficiency of our algorithms, we will use $\deg X = 3 \deg A$ for the rest of our timings in this paper.

4.4 Standard versus Optimized Timings

We recall that in the standard algorithm, we represent polynomial matrices in their X-adic representation, and the polynomial matrix-matrix operations are performed using the modp1 Maple package for univariate polynomials with coefficients modulo p. The optimized algorithm represents polynomial matrices in their X basis, which reduces polynomial matrix-matrix operations to integer matrix-matrix BLAS operations. In this section, we compare the costs of these algorithms for different values of n, where the optimized algorithm makes use of the orthogonal polynomials X and Y such that $\deg Y = \deg A$ and $\deg X = 3 \deg Y$.

n	$\deg A$	Standard	Optimized	Factor of Speedup
200	5	351 s	34.3 s	10.2
300	5	1248 s	58.4 s	21.4
400	5	4084 s	98.8 s	41.7
500	5	7581 s	162 s	46.5
800	5	35789 s	537 s	66.7

Table 4.6: Standard vs Optimized Algorithm (CPU time)

From Table 4.6, we can see that the optimized algorithm performs much better than the standard algorithm. Therefore, the reduction to BLAS operations is very effective in increasing the algorithm's efficiency.

4.5 External BLAS Libraries

Since we have reduced most of the operations in our algorithm to level 3 BLAS operations, we can use external libraries for efficient matrix-matrix multiplication and matrix inversion. We use the OpenBLAS 0.2.14 library [11] for matrix-matrix multiplication and the integer matrix library (IML 1.0.4) [2] for inversion of integer matrices. While the BLAS included with the Maple distribution is a precompiled binary, OpenBLAS has been compiled on and optimized for the particular machine we are using for the timings. To use the OpenBLAS library, we use the following command:

```
fgemm := define_external('cblas_dgemm',
                ORDER::integer[8],
                TRANSA::integer[8],
                TRANSB::integer[8],
                M::integer[8],
                N::integer[8],
                K::integer[8],
                ALPHA::float[8],
                A::(ARRAY(1..M,1..K,float[8])),
                LDA::integer[8],
                B::(ARRAY(1..K,1..N,float[8])),
                LDB::integer[8],
                BETA::float[8],
                C::(ARRAY(1..M,1..N,float[8])),
                LDC::integer[8],
                'THREAD_SAFE',
                LIB="/u4/astorjohann/software/lib/libopenblas.so"):
```

A similar command can be used to call the integer matrix library. We will mainly focus on the use of the OpenBLAS library in this section since the commands for using either library are similar. In the following table, we compare matrix-matrix multiplication of n by n integer matrices using Maple BLAS operations and OpenBLAS operations.

n	Maple BLAS	OpenBLAS	Factor of Speedup
200	0.245 s	0.091 s	2.70
400	0.345 s	0.124 s	2.79
800	0.728 s	0.209 s	3.48
1600	3.346 s	0.990 s	3.38

Table 4.7: Maple BLAS vs OpenBLAS matrix-matrix multiplication (CPU time)

From Table 4.7 we can see that using external libraries increases the efficiency of matrix-matrix multiplication for integer matrices. Since most of the operations performed in our algorithm involve matrix-matrix multiplication, we can further increase our algorithms efficiency by using these external libraries.

In the following table, we see a decrease in CPU time from using these libraries:

n	$\deg A$	Standard	Opt. w/ Maple BLAS	Opt. w/ Ext. Libraries
200	5	351 s	34 s	20 s
300	5	1248 s	58 s	35 s
400	5	4084 s	98 s	75 s
500	5	7581 s	163 s	149 s
800	5	35789 s	537 s	353 s

Table 4.8: Optimized Algorithm with Maple BLAS vs with External Libraries (CPU time)

Chapter 5

Application

We now have an algorithm RatModSolve which gives us a solution to $Av = db$ modulo any prime p for $A \in \mathbb{Z}_p[x]^{n\times n}$ and $b \in \mathbb{Z}_p[x]^{n\times m}$. We can use RatModSolve to solve the more general problem where $A \in \mathbb{Z}[x]^{n\times n}$ and $b \in \mathbb{Z}[x]^{n\times m}$. Once we compute enough images modulo distinct primes we can use Chinese Remaindering to compute the solution pair (v, d) such that $v \in \mathbb{Z}[x]^{n\times m}$ and $d \in \mathbb{Z}[x]$. However, only using Chinese Remaindering will not give us the solution. We must take into account the coefficients of polynomials in v and of the polynomial d. We need to re-describe the problem once more:

We have $Av = db$. In general, our solution will have $v \in \mathbb{Q}[x]^{n\times m}$ and $d \in \mathbb{Q}[x]$. Let D be an integer such that $Dv \in \mathbb{Z}[x]^{n\times m}$ and $Dd \in \mathbb{Z}[x]$. Then we have $A(Dv) = (Dd)b$. Thus the (v, d) solution pair we are looking for will need to make use of the integer D.

The following is pseudocode for the algorithm RatQSolve:

RatQSolve$(A, b, d_X, d_Y, N_0, imagenumber, steps)$
Inputs: $A \in \mathbb{Z}[x]^{n\times n}$, $b \in \mathbb{Z}[x]^{n\times n}$, and $d_Y, d_X, N_0, imagenumber, steps \in \mathbb{Z}$
with $d_Y, d_X, imagenumber, steps > 0$ and $N_0 \geq 0$.
Outputs: $v \in \mathbb{Z}[x]^{n\times m}$ and $d \in \mathbb{Z}[x]$ such that $Av = db$.
Conditions: $d_Y \geq \deg A$ and $d_X \geq \deg A$.
$\alpha := \max\big(\text{Size}(A)\,,\, \text{Size}(b)\big)$;
$p :=$ MaxModulus(n);
do
$p := prevprime(p)$;
$Amod := A \mod p$;
$bmod := b \mod p$;

$v', d' :=$ RatModSolve$(Amod, bmod, p, d_X, d_Y, N_0, steps)$;
if maximum degree of denominators $<$ deg d' **then**
Discard all previous images.
$P := p$;
Add (v', d', p) to our set of images.
Compute more images.
elif maximum degree of denominators $=$ deg d' **then**
$P := P \cdot p$;
Add (v', d', p) to our set of images.
fi;
if we have *imagenumber* images **then**
$N := NumeratorBound(A, \alpha, n, P)$;
$VV, DD :=$ Chinese Remaindering on images (v', d', p);
$VV', DD', drat :=$ IntegerRationalReconstruction(VV, DD, P);
if IntegerRationalReconstruction FAILS **then**
Compute more images.
fi;
$v := drat \cdot VV'$ (mods P);
$d := drat \cdot DD'$ (mods P);
else
Compute more images.
fi;
while Size$(v) \geq N$ **or** Size$(d) \geq N$ **od**;
return v, d

The integer D we referred to earlier is denoted by *drat* in the above pseudocode for RatQSolve. We compute *drat* through the rational reconstruction of each coefficient of VV and DD. We start with $drat = 1$ and if rational reconstruction returns the fraction a/b, then $drat := drat \cdot b$. Then we pre-multiply the next coefficient in VV or DD by *drat* and perform rational reconstruction with the appropriate decrease in the denominator bound. After going through every coefficient of VV and DD, we obtain our integer *drat* such that $drat \cdot VV' \in \mathbb{Z}[x]^{n \times m}$ and $drat \cdot DD' \in \mathbb{Z}[x]$.

The functions in the pseudocode for RatQSolve are described below:

- Size$(A) \rightarrow$ Maximum magnitude of coefficients of the polynomials in A
- MaxModulus$(n) \rightarrow \left\lfloor \sqrt{\frac{2^{53}-1}{n}} + 1 \right\rfloor$

- NumeratorBound$(A, \alpha, n, P) \to \min\left(\left\lfloor \frac{P-2}{2n(\deg A+1)\alpha} - 1 \right\rfloor, \left\lfloor \sqrt{\frac{P}{10000}} \right\rfloor\right)$

The first of the minimum of two values for NumeratorBound is from [9, Lemma 2.1]. The reason we have a minimum of two values for the NumeratorBound is to prevent needless computations that occur if the bound is too large [1]. Instead, we compute more images of our solution until the computed NumeratorBound we use gives a successful rational reconstruction.

Chapter 6

Conclusion

We have developed an algorithm, based on algorithms from previous research in the field, that efficiently computes the solution to the rational system $Av = db$. After starting with the standard algorithm, we describe the modifications made to increase its efficiency and perform timings on different instances of the inputs to show that the optimized algorithm is more efficient at finding the solution pair (v, d) such that $Av = db$. For an input of dimension $n = 200$, and degrees of entries less than or equal to 5, the optimized algorithm using external libraries computes the solution 17.7 times faster than the standard algorithm. An application of our algorithm was then presented to find solutions to the more general rational system where $A \in \mathbb{Z}[x]^{n \times n}$ and $b \in \mathbb{Z}[x]^{n \times m}$, as opposed to finding solutions over the field $\mathbb{Z}_p$.

More work can be done to improve the efficiency of the algorithm. In Section 4.3 we attempted to find a relation between $\deg X$ and $\deg A$, but our results showed that the optimal $\deg X$ is affected by more than just $\deg A$. In general, it is a function of n, m, $\deg A$, and $\deg b$. Further testing will need to be done to find the optimal $\deg X$ which would allow us to auto-tune this parameter in our algorithm given the sizes and degrees of the inputs.

We can also explore other methods of lifting, such as DoublePlusOneLifting [7] which combines linear and quadratic lifting into a single step. This produces a quadratic increase in the power of the modulus in the lifting phase after each step, as opposed to a linear increase produced from our algorithm, meaning we would arrive to a solution much faster.

References

[1] S. Cabay. Exact solution of linear systems. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, pages 248—253, 1971.

[2] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'05*, pages 92–99. ACM Press, New York, 2005.

[3] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms forComputer Algebra*. Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[4] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proc. EUROSAM '79, volume 72 of Lecture Notes in Compute Science*, pages 65–72, Berlin-Heidelberg-New York, 1979. Springer-Verlag.

[5] T. Mulders and A. Storjohann. Diophantine Linear System Solving. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'09*, pages 281–288. ACM Press, New York, 1999.

[6] T. Mulders and A. Storjohann. Certified Dense Linear System Solving. *Journal of Symbolic Computation*, 37(4):485–510, 2004.

[7] C. Pauderis and A. Storjohann. Deterministic unimodularity certification. In J. van der Hoeven and M. van Hoeij, editors, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'12*, pages 281–288. ACM Press, New York, 2012.

[8] A. Storjohann. High–Order Lifting and Integrality Certification. *Journal of Symbolic Computation*, 36(3–4):613–648, 2003.

[9] A. Storjohann. On the complexity of inverting integer and polynomial matrices. *Computational Complexity*, 2010. Accepted for publication.

[10] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, 3rd edition, 2013.

[11] Z. Xianyi. OpenBLAS: an optimized BLAS library. http://www.openblas.net/, 2011–2015.