Deterministic solution of a rational linear system of polynomials over abstract fields

by

Jonathan Valeriote

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

Waterloo, Ontario, Canada, 2010

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

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

Abstract

Given a matrix A we present a way to decompose A into two matrices U_x and H_x with x being relatively prime to $\det U_x$ and $(x-1)$ being relatively prime to $\det H_x$. We then apply this to design a deterministic algorithm for solving a rational linear system of equations over an abstract field K. Given an $A \in \mathsf{K}[x]^{n\times n}$ and $b \in \mathsf{K}[x]^{n\times 1}$ the algorithm will return $A^{-1}b \in \mathsf{K}(x)^{n\times 1}$. The cost of the algorithm is $O(n^3\mathsf{M}(d) + n\mathsf{B}(nd))$ where d is a bound of degree of A and (degree of b)$/n$, M is the cost of polynomial multiplication, and B is the cost of gcd-like operations. We also present an algorithm using partial linearization to extend the effectiveness of system solvers.

Acknowledgements

I would like to thank my supervisor, Dr. Arne Storjohann, for all of his help and guidance while writing this paper.

Dedication

This is dedicated to my parents.

Contents

List of Figures

Chapter 1

Introduction

Let K be a field. Finding the unique solution to a nonsingular rational system of linear equations is a classical mathematical problem. The nonsingular rational system problem takes as input a nonsingular matrix $A \in \mathsf{K}[x]^{n \times n}$ and vector $b \in \mathsf{K}[x]^{n \times 1}$ and returns as output $A^{-1}b \in \mathsf{K}(x)^{n \times 1}$. There has been much research done [1, 3, 4, 5, 7] on solving nonsingular rational systems and this paper will attempt to improve on some of the previous work. We follow previous authors and analyze algorithms in terms of required number of field operations from K, and give cost estimates in terms of n, $\deg A$ and $\deg b$. We define $\deg A$ to be the maximal degree of any element in the matrix/vector A.

[1] and [3] developed algorithms for solving the nonsingular rational system problem. Fast polynomial multiplication can be incorporated without much difficulty, see for example [5]. Let d be a bound for $\deg A$ and $(\deg b)/n$. Given an $X \in \mathsf{K}[x]$ such that $X \perp \det A$ and $\deg X \leq d$, linear lifting can be used to compute $A^{-1}b$ in $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(d))$ operations from K. Here, M is the cost of polynomial multiplication, and B is the cost of gcd-like operations, see Section 1.1. Using high-order lifting [7] was able to incorporate matrix multiplication to achieve a cost of $O(n^\omega(\log n)\mathsf{B}(d))$ field operations.

Both linear and high-order lifting require as input a small irreducible $X \in \mathsf{K}[x]$, such that $X \perp \det A$, where $X \perp \det A$ means that X and $\det A$ are relatively prime. To support the stated running time bounds we should also have $\deg X \leq \deg A$, or at least $\deg X \in O(d)$. A suitable X can be chosen randomly. Typically, X is chosen to be $X = (x - \gamma)^k$ for $\gamma \in \mathsf{K}$ chosen randomly. Since we require that $X \perp \det A$, γ needs to be selected such that $(x - \gamma)$ is not a factor of $\det A$. We know that $\det A \leq nd$ so there are at most nd roots, if $|\mathsf{K}| \geq 2nd$ the algorithms will select a good prime with probability at least $1/2$. If K is too small such a γ may not exist. To solve this, the algorithms extend K to an extension field, but using an extension field is costly. Randomly selecting the prime causes these algorithms to be Las Vegas probabilistic.

There has also been work done on deterministically solving the nonsingular rational system problem. [4] present an algorithm which has cost $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(nd))$. They are able to solve this deterministically by using the vector b from the system as an oracle which shows them how to find the solution. This forces more work to be done for every new system.

In this paper we instead focus on the matrix A from the system and decompose it into two matrices,U_x and H_x, such that $A = U_x H_x$ with $x \perp \det U_x$ and $(x-1) \perp \det H_x$. Having an algorithm that works this way allows a user to only need one decomposition for the matrix A which then allows them to solve any number of systems by selecting a new b. Using a solver based on high-order lifting requires a small irreducible, so we use the fact that for all fields K, $\mathsf{K}[x]$ has irreducibles x, and $(x-1)$. We use a power of x to solve a system involving U_x, and then use a power of $(x-1)$ to solve a system involving H_x, this allows us to find $A^{-1}b$.

Problems may arise when attempting to use an existing rational solver with the input matrix H_x. H_x may have some some columns with large degree. In fact, for A with $\deg A \leq d$, the matrix H_x, in the decomposition $A = U_x H_x$, may have some columns with degree as large as nd. This can adversely affect the efficiency of computing $A^{-1}b$ because the cost of linear or high-order lifting algorithms is sensitive to the largest degree in the input matrix. We deal with this by presenting a method for transforming the system $Av = b$ into an equivalent system $Du = c$ with D having degree bounded more tightly than the degree of A and with dimension less than $2n$.

The main contributions of this thesis are:

- An algorithm to decompose a matrix, $A \in \mathsf{K}[x]^{n\times n}$ of degree d, into two parts U_x and H_x with $x \perp \det U_x$ and $(x-1) \perp \det H_x$. The cost of the algorithm is $O(n^3\,\mathsf{M}(d))$ field operations from K.

- A method for transforming a system which has some column (or row) degrees higher than most into an equivalent system which has degree equal to the average of the column (or row) degrees, and with dimension less than twice that of the input matrix. The transformation is a rewriting of the input matrix and does not require any computation.

We now give an outline of the rest paper. The algorithm for decomposing a matrix A into U_x and H_x is detailed in Chapter 2. Chapter 3 presents the transformation technique which improves the effectiveness of system solvers. Finally Chapter 4 shows how the two main results can be used together to solve deterministically a nonsingular rational system of linear equations with a cost of $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(nd))$.

1.1 Preliminaries

The cost analysis of our algorithm is given in terms of a bound on the number of required field operations from K on an algebraic random access machine; the operations $+$, $-$, $\times$, and "divide by a nonzero" are considered as unit step operations.

We use M for the cost of multiplying polynomials. Let $\mathsf{M} : \mathbb{Z}_{\geq 0} \longrightarrow \mathsf{N}$ be such that polynomials in $\mathsf{K}[x]$ of degree bounded by d can be multiplied using at most $\mathsf{M}(d)$ field operations. The algorithm of [2] allows $\mathsf{M}(d) \in O(d^{1.59})$. We may also assume that $\mathsf{M}(ab) \leq \mathsf{M}(a)\mathsf{M}(B)$ for a,$b \in \mathbb{Z}_{>1}$.

We also define a function B for polynomial gcd-related computations. We assume that $\mathsf{B}(d)$ is $O(\mathsf{M}(d) \log d)$. Then the extended gcd problem with two polynomials in $\mathsf{K}[x]$ of degree bounded by d can solved with $O(\mathsf{B}(d))$.

Chapter 2

Computing a triangular x-basis matrix factorization

Given a matrix $A \in \mathsf{K}[x]^{n\times n}$ we will factor it as the product of two matrices: $A = U_x H_x$. This factorization will be a variation of the Hermite normal form of A but H_x will have only powers of x along the diagonal. So H_x will remove all of the powers of x from the determinant of A. This allows U_x to have determinant that is not divisible by x. We will call H_x the x-Hermite form of A.

Definition 1. *The x-Hermite decomposition of nonsingular $A \in \mathsf{K}[x]^{n\times n}$ is $A = U_x H_x$ where $U_x, H_x \in \mathsf{K}[x]^{n\times n}$ with:*

- *H_x is upper triangular with powers of x along the diagonal.*
- *For $j > i$, $\deg H_x[i,j] < \deg H_x[i,i]$.*
- *If $\det A = x^e g(x)$ where $x \perp g(x)$ then $\det H_x = x^e$.*

To illustrate the process used to factorize A we will use the following example. The matrix

$$A = \begin{bmatrix} 0 & 4x^3+4x^2 & 2x^2+x+3 \\ x & 4x^4+4x^3+2x & 4x^2+3x+4 \\ 4x^2 & x^5+x^4+x^3+4x^2 & x^3+x^2+x+2 \end{bmatrix} \in \mathbb{Z}_5[x]^{3\times 3} \tag{2.1}$$

has Hermite factorization

$$A = \overset{U}{\begin{bmatrix} 0 & 4 & 2 \\ 1 & 4x & 0 \\ 4x & x^2+1 & 4 \end{bmatrix}} \overset{H}{\begin{bmatrix} x & 2x & 4 \\ & x^3+x^2 & x+2 \\ & & x^2+x \end{bmatrix}} \tag{2.2}$$

while it has x-Hermite factorization

$$A = \overset{U_x}{\begin{bmatrix} 0 & 4x+4 & 2x+3 \\ 1 & 4x^2+4x & x \\ 4x & x^3+x^2+x+1 & 4x^2+4x+3 \end{bmatrix}} \overset{H_x}{\begin{bmatrix} x & 2x & 4 \\ & x^2 & 2 \\ & & x \end{bmatrix}} \tag{2.3}$$

There is one main difference between the Hermite normal factorization $A = UH$ and the x-Hermite factorization $A = U_x H_x$. On the one hand the Hermite form has the property that $\det U \in \mathsf{K} \setminus \{0\}$ and $\det H$ is an associate of $\det A$. On the other hand we get that $\det H_x$ is a power of x and $\det U_x$ is an associate of $\det A$ with all powers of x removed.

The algorithm constructs H_x one column at a time which allows for H_x to be viewed as a product of structured matrices which we will write as $H_x = E_n V_n E_{n-1} V_{n-1} \dots E_1 V_1$ where $V_1 = I$.

For our example, we obtain the following:

$$H_x = \begin{bmatrix} x & 2x & 4 \\ & x^2 & 2 \\ & & x \end{bmatrix} = \overset{E_3}{\begin{bmatrix} 1 & & \\ & 1 & \\ & & x \end{bmatrix}} \overset{V_3}{\begin{bmatrix} 1 & & 4 \\ & 1 & 2 \\ & & 1 \end{bmatrix}} \overset{E_2}{\begin{bmatrix} 1 & & \\ & x^2 & \\ & & 1 \end{bmatrix}} \overset{V_2}{\begin{bmatrix} 1 & 2x & \\ & 1 & \\ & & 1 \end{bmatrix}} \overset{E_1}{\begin{bmatrix} x & & \\ & 1 & \\ & & 1 \end{bmatrix}}$$

We can see that the E_j's are constructed based on the diagonal elements of H_x and the V_j's are based on the entries above the diagonal in column j.

Algorithm x-`HermiteForm` allows for H_x^{-1} to be found easily as $H_x^{-1} = V_1^{-1} E_1^{-1} \dots V_{n-1}^{-1} E_{n-1}^{-1} V_n^{-1} E_n^{-1}$. Based on the construction of the V_j's and E_j's it easy to find their inverses. Here is H_x^{-1} constructed in this way

$$\begin{aligned} H_x^{-1} &= \begin{bmatrix} x^{-1} & -2x & -4 \\ & x^{-2} & -2 \\ & & x^{-1} \end{bmatrix} \\ &= \overset{E_1^{-1}}{\begin{bmatrix} x^{-1} & & \\ & 1 & \\ & & 1 \end{bmatrix}} \overset{V_2^{-1}}{\begin{bmatrix} 1 & -2x & \\ & 1 & \\ & & 1 \end{bmatrix}} \overset{E_2^{-1}}{\begin{bmatrix} 1 & & \\ & x^{-2} & \\ & & 1 \end{bmatrix}} \overset{V_3^{-1}}{\begin{bmatrix} 1 & & -4 \\ & 1 & -2 \\ & & 1 \end{bmatrix}} \overset{E_3^{-1}}{\begin{bmatrix} 1 & & \\ & 1 & \\ & & x^{-1} \end{bmatrix}} \end{aligned}$$

2.1 Via polynomial multiplication

We present an algorithm for computing the x-Hermite form of a given matrix over $\mathsf{K}[x]$. The algorithm is iterative and works through the matrix one column at a time. The given

x-`HermiteForm`(A, n)
Input: Nonsingular $A \in \mathsf{K}[x]^{n \times n}$
Output: H_x, U_x such that $A = U_x H_x$ and H_x in x-Hermite Form
$d := \deg A$;
$H_x := 0 \times 0$ matrix; # this will grow in size at each step
$P := I_n$; # will keep track of the row swaps
$B := 0 \times 0$ matrix;# this will store the inverse, starts as empty matrix

$U_x := A$; # in practice, A is modified in place.

for j **from** 0 **to** $n-1$ **do**
Decompose U_x as

$$U_x = \left[\begin{array}{c|c|c} C_1 & w_1 & * \\ \hline C_2 & w_2 & * \end{array}\right] \in \mathsf{K}[x]^{n \times n}$$

where $C_1 \in \mathsf{K}[x]^{j \times j}$, $w_1 \in \mathsf{K}[x]^{j \times 1}$, $C_2 \in \mathsf{K}[x]^{(n-j) \times j}$, and $w_2 \in \mathsf{K}[x]^{(n-j) \times 1}$.
e_{j+1}, $v^{[j+1]}$, u_1, u_2, $P_j :=$ `overDeterminedLifting`$(C_1, w_1, C_2, w_2, n, j, B, d)$;

$H_x := \left[\begin{array}{c|c} H_x & v^{[j+1]} \\ \hline & x^{e_{j+i}} \end{array}\right] \in \mathsf{K}[x]^{(j+1) \times (j+1)}$;
Column$(U_x, j+1) := \left[\begin{array}{c|c} u_1 & u_2 \end{array}\right]^T$;
$U_x := P_j U_x$
$P := P_j P$;

Decompose U_x as

$$U_x = \left[\begin{array}{c|c|c} C & r & * \\ \hline w & a & * \\ \hline * & * & * \end{array}\right] \in \mathsf{K}[x]^{n \times n}$$

where $C \in \mathsf{K}[x]^{j \times j}$, $r \in \mathsf{K}[x]^{j \times 1}$, $w \in \mathsf{K}[x]^{1 \times j}$, and $a \in \mathsf{K}$.
$B :=$ `updateInverse`(C, w, r, a, B, j, d);
od;

$U_x := P^{-1} U_x$;

return H_x, U_x;

Figure 2.1: Algorithm x-`HermiteForm`

`overDeterminedLifting`$(C_1, w_1, C_2, w_2, n, j, B, d)$
Input:

- $n \in \mathbb{Z}_{>0}$, $j \in [0, ..., n-1]$, $d \in \mathbb{Z}_{>0}$
- $C_1 \in \mathsf{K}[x]^{j \times j}$, $w_1 \in \mathsf{K}[x]^{j \times 1}$, $B = \text{Rem}(C_1^{-1}, x^d)$
- $C_2 \in \mathsf{K}[x]^{(n-j) \times j}$, $w_2 \in \mathsf{K}[x]^{(n-j) \times 1}$

Output: $k \in \mathbb{Z}_{\geq 0}$, $v \in \mathsf{K}[x]^{j \times 1}$ with degree$(v) < k$, $u_1 \in \mathsf{K}[x]^{j \times 1}$, $u_2 \in \mathsf{K}[x]^{(n-j) \times 1}$, a permutation $P = \text{diag}(I_j, *)$ such that

- $v := \text{Rem}(C_1^{-1} w_1, x^k) \in \mathsf{K}[x]^{j \times 1}$, with k maximal such that $C_2 v \equiv w_2 \mod x^k$
- $\left[\begin{array}{c} u_1 \\ \hline u_2 \end{array}\right] := \left[\begin{array}{c} (w_1 - C_1 v)/x^k \\ \hline (w_2 - C_2 v)/x^k \end{array}\right] \in \mathsf{K}[x]^{n \times 1}$
- $P \left[\begin{array}{c|c} C_1 & u_1 \\ \hline C_2 & u_2 \end{array}\right]$ has principal $(j+1) \times (j+1)$ submatrix nonsingular modulo x

Condition: $\left[\begin{array}{c|c} C_1 & w_1 \\ \hline C_2 & w_2 \end{array}\right] \in \mathsf{K}[x]^{n \times (j+1)}$ has rank $j+1$

1. Use x^d-adic lifting with B to compute $v := \text{Rem}(C_1^{-1} w_1, x^{td})$ for maximal t such that $C_2 v \equiv w_2 \mod x^{td}$.
2. $\left[\begin{array}{c} u_1 \\ \hline u_2 \end{array}\right] := \left[\begin{array}{c} (w_1 - C_1 v)/x^{td} \\ \hline (w_2 - C_2 v)/x^{td} \end{array}\right] \in \mathsf{K}[x]^{n \times 1}$;
3. Let $v_s := \text{Rem}(C_1^{-1} u_1, x^d)$ and find maximal s such that x^s divides $u_2 - C_2 v_s$. $(s < d)$
4. Let P be the permutation matrix for swapping a row which has trailing degree s in $\left[\begin{array}{c} u_1 - C_1 v_s \\ \hline u_2 - C_2 v_s \end{array}\right]$ with row j+1.
5. Let $\bar{v} := \text{Rem}(v_s, x^s)$. Set $v := v + \bar{v} x^{(t+1)d}$ and $k := td + s$.
6. $\left[\begin{array}{c} u_1 \\ \hline u_2 \end{array}\right] := \left[\begin{array}{c} (u_1 - C_1 \bar{v})/x^s \\ \hline (u_2 - C_2 \bar{v})/x^s \end{array}\right] \in \mathsf{K}[x]^{n \times 1}$;

Figure 2.2: Algorithm `overDeterminedLifting`

`updateInverse`(C, w, r, a, B, j, d)
Input:

- $j \in \mathbb{Z}_{\geq 0}$, $d \in \mathbb{Z}_{\geq 0}$
- $C \in \mathsf{K}[x]^{j \times j}$, $w \in \mathsf{K}[x]^{1 \times j}$, $r \in \mathsf{K}[x]^{j \times 1}$, $a \in \mathsf{K}[x]$
- $B = \mathrm{Rem}(C^{-1}, x^d) \in \mathsf{K}[x]^{j \times j}$

Output: $B := \mathrm{Rem}(\bar{C}^{-1}, x^d)$ where $\bar{C} = \left[\begin{array}{c|c} C & w \\ \hline r & a \end{array}\right] \in \mathsf{K}[x]^{(j+1)\times(j+1)}$

$$B = \left[\begin{array}{c|c} \mathrm{Rem}(B + Bw(a - rBw)^{-1}rB, x^d) & \mathrm{Rem}(-Bw(a - rBw)^{-1}, x^d) \\ \hline \mathrm{Rem}(-(a - rBw)^{-1}rB, x^d) & \mathrm{Rem}((a - rBw)^{-1}, x^d) \end{array}\right] \in \mathsf{K}[x]^{(j+1)\times(j+1)}$$

return B;

Figure 2.3: Algorithm `updateInverse`

matrix A is treated as the working matrix and is modified in place, allowing U_x to be equal to the final A up to some row permutations. The algorithm is presented in Figure 2.1.

Theorem 2. *Algorithm* x`-HermiteForm` *is correct.*

Proof. By construction we get that H_x is upper triangular with powers of x along the diagonal and that the degrees of the diagonal entries are strictly larger than the degrees of the other elements in the same column. At the end of the algorithm we have computed $B = U_x^{-1} \mod x^d$, this tells us that $x \perp \det U_x$. If $\det A = x^e g(x)$ with $x \perp g(x)$ then $\det H_x = x^e$ since $\det A = (\det U_x)(\det H_x)$ and $x \perp \det U_x$.

The correctness of `findInverse` follows from the Sherman-Morrison-Woodbury formula. Within this construction the element $(a - rBw)$ is needed to be invertible. After calling `overDeterminedLifting` and updating U_x we have that the $(j+1) \times (j+1)$ principal submatrix of U_x is nonsingular modulo x. Thus the pivot element $(a - rBw)$ from the Sherman-Morrison-Woodbury formula must be invertible modulo x. □

The function `overDeterminedLifting` uses x^d-adic lifting on each column to compute the maximum power of x that can be removed from the column. The lifting also computes what we will be adding to the corresponding column in H_x for the decomposition of A. Refer to Appendix A for the Maple code implementation of the algorithm.

Refer to Figure 2.4 to see how the lifting algorithm works.

Example of `overDeterminedLifting`

Let $A = \left[\begin{array}{c|c} 1 & 2x^2+1 \\ \hline x & x \end{array}\right]$.

Set $C_1 := 1$, $C_2 := x$, $w_1 := 2x^2+1$, $w_2 := x$, and $B := 1$ then `overDeterminedLifting`$(C_1, w_1, C_2, w_2, 2, 2, B, 2)$ is executed in the following way.

$k = 0$: $v = \text{Rem}(w_1, 1) = 0$ so $C_2 v \equiv 0 \mod 1$ and $w_2 \equiv 0 \mod 1$.
$k = 1$: $v = \text{Rem}(w_1, x) = 1$ so $C_2 v \equiv 0 \mod x$ and $w_2 \equiv 0 \mod x$.
$k = 2$: $v = \text{Rem}(w_1, x^2) = 1$ so $C_2 v \equiv x \mod x^2$ and $w_2 \equiv x \mod x^2$.
$k = 3$: $v = \text{Rem}(w_1, x^3) = 2x^2+1$ so $C_2 v \equiv x \mod x^3$ and $w_2 \equiv x \mod x^3$.
$k = 4$: $v = \text{Rem}(w_1, x^4) = 2x^2+1$ so $C_2 v \equiv 2x^3 + x \mod x^4$ and $w_2 \equiv x \mod x^4$.

When $k = 4$, we get a difference in $C_2 v \mod x^k$ and $w_2 \mod x^k$ this tells us that the choice for k is 3.
So then we get

$$v = \text{Rem}(C_1^{-1} w_1, x^k) = 2x^2 + 1$$
$$u_1 = (w_1 - C_1 v)/x^k) = 0$$
$$u_2 = (w_2 - C_2 v)/x^k) = 2$$

And because there are no row swaps $P = I_2$

Figure 2.4: Example of `overDeterminedLifting`

Examining the algorithm we see that at the beginning of step $j+1$ we have the following situation:

$$A = \left[\begin{array}{c|c|c} C & w_1 & * \\ \hline D & w_2 & * \end{array}\right] \in \mathsf{K}[x]^{n\times n}$$

where $C \in \mathsf{K}[x]^{j\times j}, D \in \mathsf{K}[x]^{(n-j)\times j}, w_1 \in \mathsf{K}[x]^{j\times 1}, w_2 \in \mathsf{K}[x]^{(n-j)\times 1}$ We have also updated the inverse matrix B to be such that

$$B = \mathrm{Rem}(C^{-1}, x^d) \in \mathsf{K}[x]^{j\times j}$$

The formula used in the algorithm for computing B is based on the Sherman-Morrison-Woodbury formula for matrix inversion.

Within each step, the algorithm stops when we are unable to pull out a full x^d factor from the working matrix. At this point we are pulling out x^k for some $k < d$.

Once we have completed step $j+1$ we now have the x-Hermite form of the principal $(j+1)\times(j+1)$ submatrix of A.

$$H_x = \begin{bmatrix} x^{e_1} & v_1^{[2]} & v_1^{[3]} & & v_1^{[j+1]} & & \\ & x^{e_2} & v_2^{[3]} & & v_2^{[j+1]} & & \\ & & x^{e_3} & & \vdots & & \\ & & & \ddots & v_j^{[j+1]} & & \\ & & & & x^{e_{j+1}} & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}$$

Lemma 3. $x \perp \det U_x$

Proof. From the x-Hermite form of A we know that $U_x{=}AH_x^{-1}$ so this tells us that $\det U_x = (\det A)(\det H_x^{-1})$. The structure of H_x forces $\det H_x = x^k$ for some $k \in \mathbb{Z}_{\geq 0}$. Having this, we also know that $\det H_x^{-1} = x^{-k}$. So we get that $\det U_x = (\det A)/x^k$ and by the construction of H_x we know that k is maximal (ie, $x^{k+1} \perp \det U_x$). This gives us that $x \perp \det U_x$. □

2.2 Cost Analysis

Lemma 4. *Given $H_x \in \mathsf{K}[x]^{n\times n}$, the x-Hermite form of a matrix $A \in \mathsf{K}[x]^{n\times n}$, $\deg H_x \leq nd$, where $d = \deg A$.*

Proof. Using the x-Hermite factorization we have that $A = U_x H_x$. From Hadamard's bound for determinants we get that $\det(A) \leq nd$. The factorization tells us that $\det(A) = \det(U_x)\det(H_x)$, so when looking at the degrees of this equation we get that

$$\deg(\det A) = \deg(\det U_x) + \deg(\det H_x).$$

Now since the left hand side of this equation is $\leq nd$, we know that the right hand side will also be $\leq nd$. This then gives us that $\deg(\det H_x) \leq nd - \deg(\det U_x)$ and since $\deg(\det U_x) \geq 0$ we get that $\deg(\det H_x) \leq nd$. Since H_x is in x-Hermite form we know that the determinant is just the product of the diagonal elements and that the highest degree element will be somewhere along the diagonal. Hence we get our result that $\deg H_x \leq nd$. □

Lemma 5. *The cost of computing the x-Hermite form of $A \in \mathsf{K}[x]^{n\times n}$ is $O(n^3\,\mathsf{M}(d))$.*

Proof. At each step in the algorithm, we are doing x^d-adic lifting which allows us to pull out d factors of x at a time from A. Since the determinant of A is $\leq nd$ there will be at most n full x^d factors of x being removed. Hence we will only be doing our loop $O(n)$ times. Within the loop the main cost is doing matrix-vector multiplications which have a cost of $n^2\,\mathsf{M}(d)$ where M is the cost to multiply two polynomials of degree $\leq d$. Thus the overall cost of calculating H_x is $O(n^3\,\mathsf{M}(d))$. □

Chapter 3

System solving via partial linearization

Solving a nonsingular rational system $Av = b$ with lifting has cost proportional to $O^{\sim}(n^3 d)$ bit operations where n is the dimension of the system and d is a bound for $\deg A$ and $(\deg b)/n$. Since the cost is sensitive to the largest element in A, there may be cases where there is a waste of significant computational work. This happens when only a few columns/rows have higher degree compared to most. In this chapter we show how to use partial linearization to transform the system into a new one which allows us to extract the solution to the original problem. The new system will have dimension bounded by $O(n)$ and maximum degree bounded by $d = \lceil E/n \rceil$ where E is the sum of the column/row degrees.

3.1 Partial Linearization of Columns

If we are given a system $Av = b$ which has some columns with larger degree than most of the others we can transform it into a new system $Du = c$. Here is an example to show how the process works. Let $\mathsf{K} = \mathbb{Z}/(11)$ with the system $Av = b$ where

$$A := \begin{bmatrix} 5\,x^4 + x^3 + 9\,x + 7 & 4 & 2\,x^2 + 3 \\ 6\,x^4 + 4\,x^3 + 10\,x^2 + 10\,x & x & 6\,x^2 + 2\,x + 8 \\ 10\,x^4 + 10\,x^3 + 5\,x^2 + 3\,x & 6\,x & 8\,x + 5 \end{bmatrix} \in \mathsf{K}[x]^{3\times 3} \qquad b := \begin{bmatrix} 2\,x \\ 5\,x \\ 3 \end{bmatrix} \in \mathsf{K}[x]^{3\times 1}$$

After applying `PartialColumnLinearization` (see Figure 3.1) we get the following new

system $Du = c$ where

$$D := \left[\begin{array}{ccc||ccc|c} 7 & 4 & 3 & 9 & 0 & 1+5\,x & 2\,x \\ 0 & x & 8 & 10 & 10 & 4+6\,x & 2+6\,x \\ 0 & 6\,x & 5 & 3 & 5 & 10+10\,x & 8 \\ \hline -x & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -x & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -x & 1 & 0 \\ \hline 0 & 0 & -x & 0 & 0 & 0 & 1 \end{array}\right] \in \mathsf{K}[x]^{7\times 7} \qquad c := \left[\begin{array}{c} 2\,x \\ 5\,x \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right] \in \mathsf{K}[x]^{7\times 1}$$

Now using a linear system solver we get the following solution vectors

$$v := \left[\begin{array}{c} \frac{x^4+3\,x^3+7\,x^2+8\,x+3}{7\,x^7+8\,x^6+2\,x^5+5\,x^4+3\,x^3+8\,x^2+9\,x} \\ \frac{90\,x^7+30\,x^4+50\,x^3+100\,x^2+70\,x+80}{7\,x^7+8\,x^6+2\,x^5+5\,x^4+3\,x^3+8\,x^2+9\,x} \\ \frac{10\,x^5+7\,x^4+x^3+5\,x^2+x}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \end{array}\right] \in K[x]^{3\times 1} \qquad u := \left[\begin{array}{c} \frac{x^4+3\,x^3+7\,x^2+8\,x+3}{7\,x^7+8\,x^6+2\,x^5+5\,x^4+3\,x^3+8\,x^2+9\,x} \\ \frac{90\,x^7+30\,x^4+50\,x^3+100\,x^2+70\,x+80}{7\,x^7+8\,x^6+2\,x^5+5\,x^4+3\,x^3+8\,x^2+9\,x} \\ \frac{10\,x^5+7\,x^4+x^3+5\,x^2+x}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \\ \frac{x^4+3\,x^3+7\,x^2+8\,x+3}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \\ \frac{x^5+3\,x^4+7\,x^3+8\,x^2+3\,x}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \\ \frac{x^6+3\,x^5+7\,x^4+8\,x^3+3\,x^2}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \\ \frac{10\,x^6+7\,x^5+x^4+5\,x^3+x^2}{7\,x^6+8\,x^5+2\,x^4+5\,x^3+3\,x^2+8\,x+9} \end{array}\right] \in \mathsf{K}[x]^{7\times 1}$$

We can then see that the u contains v in its upper three entries.

Theorem 6. *Algorithm* `PartialColumnLinearization` *is correct.*

Proof. Property (a) holds by construction. To show that properties (b) and (c) hold we demonstrate nonsingular matrices T_1, T_2 and T_3 such that $D^{-1} = T_3T_2T_1$ with A^{-1} evidently the principal submatrix of $T_3T_2T_1$.

Let

$$T_1 = \left[\begin{array}{c||c|c|c} I_n & -C_1B_1^{-1} & \cdots & -C_nB_n^{-1} \\ \hline & B_1^{-1} & & \\ \hline & & \ddots & \\ \hline & & & B_n^{-1} \end{array}\right]$$

Note that

$$-B_i^{-1}x^d\,\mathrm{col}(I_{m_i}, 1) = \left[\begin{array}{c} -x^d \\ \vdots \\ -x^{m_id} \end{array}\right] \in \mathsf{K}[x]^{m_i\times 1}. \tag{3.1}$$

Each B_i is unit upper triangular, so $\det T_1 = 1$.

If we transform D on the left with T_1 we obtain

$$T_1 D = \left[\begin{array}{c|c|c||c|c|c} \mathrm{col}(A,1) & \cdots & \mathrm{col}(A,n) & & & \\ \hline b_1 & & & I & & \\ \hline & \ddots & & & \ddots & \\ \hline & & b_n & & & I \end{array}\right], \tag{3.2}$$

where b_i denotes the column vector on the right hand side of (3.1). Now continue to transform the matrix on the right hand side of (3.2) with $T_2 := \mathrm{diag}(A^{-1}, I)$ to obtain

$$\left[\begin{array}{c|c|c||c|c|c} \mathrm{col}(I_n,1) & \cdots & \mathrm{col}(I_n,n) & & & \\ \hline b_1 & & & I & & \\ \hline & \ddots & & & \ddots & \\ \hline & & b_n & & & I \end{array}\right]. \tag{3.3}$$

Finally, let

$$T_3 = \left[\begin{array}{c|c|c||c|c|c} \mathrm{col}(I_n,1) & \cdots & \mathrm{col}(I_n,n) & & & \\ \hline -b_1 & & & I & & \\ \hline & \ddots & & & \ddots & \\ \hline & & -b_n & & & I \end{array}\right],$$

the inverse of (3.3). Note that $\det T_2 = \det A^{-1}$, and $\det T_3 = 1$ since it is unit lower triangular. We then have $T_3 T_2 T_1 = D^{-1}$ which tells us that

$$\det D^{-1} = (\det T_3)(\det T_2)(\det T_1) = \det A^{-1}.$$

From this property (b) follows. □

Now given this partial lineariz ation algorithm we can create solvers for solving rational systems.

Theorem 7. *Let nonsingular $A \in \mathsf{K}[x]^{n\times n}$ and $b \in \mathsf{K}[x]^{n\times 1}$ be given. The problem of computing $A^{-1}b$ can be transformed to that of computing $D^{-1}c$ for a matrix $D \in \mathsf{K}[x]^{\bar{n}\times\bar{n}}$ of degree bounded $\lceil \frac{E}{n} \rceil$ where*

- *E can be the sum of the column degrees of A or*
- *E can be the sum of the row degrees of A*

and $\bar{n} < 2n$.

Proof. If we let D be the output of `PartialColumnLinearization` then we have the following lemma.

Lemma 8. $\dim(D) < 2n$ *and* $\deg D \leq \lceil \frac{E}{n} \rceil$.

$$\begin{aligned}
\dim(D) &= n + m_1 + m_2 + \ldots + m_n \\
&= n + \max(0, \lceil (\deg \mathrm{col}(A,1) - d)/d \rceil) + \ldots + \max(0, \lceil (\deg \mathrm{col}(A,n) - d)/d \rceil) \\
&< n + \deg \mathrm{col}(A,1)/d + \ldots + \deg \mathrm{col}(A,n)/d \\
&\text{since } \max(0, \lceil (a-d)/d \rceil) \leq \tfrac{a}{d} \\
&= n + \sum_{i=1}^{n} (\deg \mathrm{col}(A,i)/d) \\
&= n + \frac{1}{d} \sum_{i=1}^{n} (\deg \mathrm{col}(A,i)/d) \\
&= n + (\frac{1}{d})E \\
&\leq n + \frac{1}{d}(nd) - \frac{n}{d} \qquad \text{since } d = \lceil \tfrac{E}{n} \rceil \\
&= 2n
\end{aligned}$$

Degree bound follows from the construction of D.

Now using

- `SolveViaPartialColumnLinearization` for the column case
- `SolveViaPartialRowLinearization` for the row case

will produce $A^{-1}b$ because D is constructed to have A^{-1} as its $n \times n$ submatrix. Then since we set up c in the new system to have its top n entries equal to b, it is clear that the top of entries of $D^{-1}c$ will be equal to $A^{-1}b$. □

`PartialColumnLinearization`(A, n, d)
Input: Nonsingular $A \in \mathsf{K}[x]^{n\times n}$ and $d \in \mathbb{Z}_{>0}$.
Output: $D \in \mathsf{K}[x]^{\bar{n}\times\bar{n}}$ and $\bar{n} \in \mathbb{Z}_{\geq n}$ such that

(a) $\deg D \leq d$,

(b) $\det D = \det A$, and

(c) A^{-1} is equal to the principal $n \times n$ submatrix of D^{-1}.

for i **to** n **do**
$m_i := \max(0, \lceil(\deg \mathrm{col}(A, i) - d)/d\rceil)$;

$$B_i := \begin{bmatrix} 1 & & & & \\ -x^d & 1 & & & \\ & -x^d & \ddots & & \\ & & \ddots & 1 & \\ & & & -x^d & 1 \end{bmatrix} \in \mathsf{K}[x]^{m_i\times m_i}.$$

Let C_i be an $n \times m_i$ matrix.
if $m_i = 0$ **then**
$v_i := \mathrm{col}(A, i)$.
else
$v_i := \mathrm{Rem}(\mathrm{col}(A, i), x^d)$.
for j **from** 1 **to** $m_i - 1$ **do**
$C_i[*, j] := \mathrm{Rem}(\mathrm{Quo}(\mathrm{col}(A, i), x^{dj}), x^d)$
od;
$C_i[*, m_i] := \mathrm{Quo}(\mathrm{col}(A, i), x^{dm_i})$.
fi;
Comment $\mathrm{col}(A, i) = v_i + C_i[*, 1]x^d + \cdots + C_i[*, m_i]x^{dm_i}$.
od;
$\bar{n} := n + m_1 + m_2 + \cdots + m_n$.

$$D := \left[\begin{array}{c|c|c||c|c|c} v_1 & \cdots & v_n & C_1 & \cdots & C_n \\ \hline -x^d\,\mathrm{col}(I_{m_1}, 1) & & & B_1 & & \\ \hline & \ddots & & & \ddots & \\ \hline & & -x^d\,\mathrm{col}(I_{m_n}, 1) & & & B_n \end{array}\right] \in \mathsf{K}[x]^{\bar{n}\times\bar{n}}.$$

return $D, \bar{n}$
Note: An $n \times 0$ vector is the empty column vector.

Figure 3.1: Algorithm `PartialColumnLinearization`

`SolveViaPartialColumnLinearization`(A, b, n, X)
Input: Nonsingular $A \in \mathsf{K}[x]^{n \times n}$, $b \in \mathsf{K}[x]^{n \times 1}$, $X \in \mathsf{K}[x]$.
Output: $A^{-1}b$
Condition: $X \perp \det A$

1. [Construct partially linearized system]
   Let E be the sum of the column degrees of A.
   $d := \lceil \frac{E}{n} \rceil$;
   $D, \bar{n} :=$ `PartialColumnLinearization`(A, n, d);
   $c := \left[\begin{array}{cccccccc} b_1 & b_2 & \ldots & b_n & 0 & 0 & \ldots & 0 \end{array} \right]^T \in \mathsf{K}[x]^{\bar{n} \times 1}$;

2. [Compute solution of transformed system]
   $u :=$ `RationalSolve`(D, c, X);

3. [Recover solution of original system]
   $v := u[1..n]$;
   return v;

Figure 3.2: Algorithm `SolveViaPartialColumnLinearization`

`SolveViaPartialRowLinearization`(A, b, n, X)
Input: Nonsingular $A \in \mathsf{K}[x]^{n \times n}$, $b \in \mathsf{K}[x]^{n \times 1}$, $X \in \mathsf{K}[x]$.
Output: $A^{-1}b$
Condition: $X \perp \det A$

1. [Construct partially linearized system]
   Let E be the sum of the row degrees of A.
   $d := \lceil \frac{E}{n} \rceil$;
   $D^T, \bar{n} :=$ `PartialColumnLinearization`(A^T, n, d);
   $c := \begin{bmatrix} b_1 & b_2 & \dots & b_n & 0 & 0 & \dots & 0 \end{bmatrix}^T \in \mathsf{K}[x]^{\bar{n} \times 1}$;

2. [Compute solution of transformed system]
   $u :=$ `RationalSolve`(D, c, X);

3. [Recover solution of original system]
   $v := u[1..n]$;
   return v;

Figure 3.3: Algorithm `SolveViaPartialRowLinearization`

Chapter 4

Deterministic rational system solving over $\mathsf{K}[x]$

Given $U_x = AH_x^{-1}$ where H_x is the x-Hermite form of A then the following two facts holds.

Lemma 9. $\deg(\text{Adjoint}(H_x)) \leq \deg(\det H_x)$.

Proof. Let $D := \deg(\det H_x)$. Each element of $\text{Adjoint}(H_x)$ is a determinant of an $(n-1)\times(n-1)$ minor of H_x. We also know that the $\text{Adjoint}(H_x)$ is an upper-triangular matrix. So there are three cases for the (i,j)-minors of H_x.

Case 1: When $i = j$ then the (i,j)-minor is upper triangular and of the same form as H_x. This tells us that the determinant in this case is x^k for some $k \leq D$.

Case 2: When $i < j$ the (i,j)-minor is no longer upper-triangular but the determinant is the sum over all permutations of a product of elements coming from each column. Since we are looking for the degree of this element we just need to look at the largest possible product in the sum. In our case this would occur when each element chosen for the product is from the original diagonal of A. These are the largest elements in each column because of the form of H_x. This then tells us that the degree of the determinant of this minor is $\leq D$.

Case 3: When $i > j$ the (i,j)-minor is upper-triangular but will have a zero along the diagonal. This then forces the determinant of this minor to be zero.

Thus we get that $\deg(\text{Adjoint}(H_x)) \leq \deg(\det H_x)$.

□

Fact 10. $\deg U_x \leq d$ *where* $d = \deg A$

Proof. We have that $\deg H_x \leq nd$ from Fact 4. As well,

$$U_x = AH_x^{-1} = (A)(\frac{\text{Adjoint}(H_x)}{\det(H_x)})$$

and so

$$\deg U_x \leq \deg A + [\deg(\text{Adjoint}(H_x)) - \deg(\det H_x)](*).$$

Using Lemma 9 we get that $\deg(\text{Adjoint}(H_x)) - \deg(\det H_x) \leq 0$, and using this with (*) we get that $\deg U_x \leq \deg A = d$. □

Theorem 11. *Let nonsingular* $A \in \mathsf{K}[x]^{n\times n}$ *with degree bounded by* d *be given, together with a* $b \in \mathsf{K}[x]^{n\times 1}$. *If* $(\deg b)/d = O(n)$, *then the unique rational vector* $A^{-1}b$ *can be computed with* $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(nd))$ *field operations*

Proof. Given A and b, if we can solve for v in the system $Av = b$ then we have found the unique vector $A^{-1}b$. We start off by computing the x-Hermite form of A using the algorithm from Figure 2.1. This allows us to factor A as $A = U_xH_x$. We can then write our system as $U_x(H_xv) = b$ and by setting $w := H_xv$, work on the system $U_xw = b$.
We know that $\deg U_x \leq d$ from Fact 10 and that $x \perp \det(U_x)$ from Lemma 3. Thus we can quickly solve for the vector $w = U_x^{-1}b$ using our system solver.
We then move on to the system $H_xv = w$. All we know about the degree of H_x is that $\deg H_x \leq nd$ but we need to have the degree bounded by d. To solve this problem we can use `SolveViaPartialColumnLinearization` with H_x, b, and $x - 1$. Since $\det(H_x) = x^k$ we get that $(x - 1) \perp \det(H_x)$ and thus is suitable for the system solver. This algorithm will give us $A^{-1}b$.

Cost Analysis: $O(n^3\,\mathsf{M}(d))$ for computing the x-Hermite form. We make two calls to a system solver and since we have irreducibles for both of them we know that we can solve them in $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(nd))$. So overall we get a cost of $O(n^3\,\mathsf{M}(d) + n\,\mathsf{B}(nd))$. □

4.1 Worked Example

I will demonstrate the ideas presented in this paper on the following system:

$$A = \begin{bmatrix} x^2 & x \\ x & 2x^2+1 \end{bmatrix} \in \mathbb{Z}_5^{2\times 2} \qquad\qquad b = \begin{bmatrix} 2x \\ 3x^2 \end{bmatrix} \in \mathbb{Z}_5^{2\times 1}$$

x-`HermiteForm`(A) gives us the following decomposition for A

$$A = U_x H_x = \begin{bmatrix} x & 3 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x & 2x^2+1 \\ 0 & x^3 \end{bmatrix}$$

We then need to solve for $w := U_x^{-1} B$ and since $\det U_x = 2$ we can use x as an irreducible for the system solver. This gives us

$$w = \begin{bmatrix} 3x^2 \\ 4x^3 + 4x \end{bmatrix}$$

Now we can solve for $H_x^{-1} w$. Looking at H_x we see that it has a column of degree greater than 2. This means we can use `SolveViaPartialColumnLinearization`$(H_x, w, 2, x-1)$ to get our solution.
This function returns

$$H_x^{-1} w = \begin{bmatrix} \frac{3x^2+1}{x^3} \\ \frac{4x^2+4}{x^2} \end{bmatrix}$$

And since

$$A^{-1} b = (H_x^{-1} U_x^{-1}) b = H_x^{-1} (U_x^{-1} b) = H_x^{-1} w = \begin{bmatrix} \frac{3x^2+1}{x^3} \\ \frac{4x^2+4}{x^2} \end{bmatrix}$$

We are done.

Chapter 5

Conclusions

We have considered the problem of solving a nonsingular rational system of linear equations. We have extended the effectiveness of existing solvers by removing the randomness as well as allowing multiple systems to be solved with one decomposition. To do this we introduced the notion of the x-Hermite factorization of a matrix which allowed us to easily find suitable lifting moduli for the solvers. We also introduced the idea of partial linearization to extend the effectiveness and allow systems with some large degrees to be able to be solved quickly.

The partial linearization ideas extend immediately to the case of integer matrices. Further work could be done to improve the exponent of n in the cost down to ω by using fast matrix multiplication.

APPENDICES

Appendix A

x-Hermite form maple code

```
xHermiteForm := proc(A,p)
local H,l,i_min,temp,B,j,d,n;

n := LinearAlgebra[ColumnDimension] (A);
convertModp1(A,p,n);
d := findDegree(A,p,n);
H := Matrix(n,n,modp1(Zero(x),5));

l,i_min := trailingDegree(A[..,1],p,d,n);
temp := A[1,..];
A[1,..] := A[i_min,..];
A[i_min,..] := temp;

columnShift(A,p,n,l,1);

H[1,1] := modp1(ConvertIn(x^l,x),p);
modp1(Gcdex(A[1,1],ConvertIn(x^d,x),'s'),p);
B := <<s>>;

for j to n-1 do
    H[1..j+1,j+1] := overDeterminedLifting(A,B,j,p,d,n);
    B := updateInverse(A,B,p,d,j);
od;

convertOut(H,p,n);
```

```
return H;
end;

convertOut := proc(A,p,n)
    local i,j;
    for i to n do
        for j to n do
            A[i,j] := modp1(ConvertOut(A[i,j],x),p);
        od;
    od;
end;

overDeterminedLifting := proc(A,B,j,p,d,n)
    local c,M,z,v_column,V,C,D,v1,v2,k,count,Bv,CBv,DBv,w1,w2,l,
    i_min,s,v_lower,v_upper,q,newAdd,i,temp,newLowAdd;

    v_column := Vector(j,modp1(Zero(x),p));
    V := Vector(j+1);

    C := A[1..j,1..j];
    D := A[j+1..n,1..j];
    v1 := A[1..j,j+1];
    v2 := A[j+1..n,j+1];

    k := d;
    count := 0;
    Bv := matrixVectorProduct(B,v1,p,d,j,j);
    CBv := specmatrixVectorProduct(C,Bv,p,d,j,j);
    DBv := specmatrixVectorProduct(D,Bv,p,d,n-j,j);

    w1 := vectorSubtract(v1,CBv,j,p);
    w2 := vectorSubtract(v2,DBv,n-j,p);

    l,i_min := canShift(w2,p,d,n-j);

    if l = -1 then

    for s from 1 to j do
         w1[s] := modp1(Shift(w1[s],-d),p);
    od;
```

```
for s from 1 to n-j do
    w2[s] := modp1(Shift(w2[s],-d),p);
od;

for s from 1 to j do
    v_column[s] :=
    modp1(Add(v_column[s],Multiply(Bv[s],Power(ConvertIn(x,x),d*count))),p);
od;

fi;

while l = -1 do
    Bv := matrixVectorProduct(B,w1,p,d,j,j);
    CBv := specmatrixVectorProduct(C,Bv,p,d,j,j);
    DBv := specmatrixVectorProduct(D,Bv,p,d,n-j,j);
    w1 := vectorSubtract(w1,CBv,j,p);
    w2 := vectorSubtract(w2,DBv,n-j,p);

    l,i_min := canShift(w2,p,d,n-j);

    if l = -1 then
    count := count + 1;
    #Now shift
    for s from 1 to j do
        w1[s] := modp1(Shift(w1[s],-d),p);
    od;

    for s from 1 to n-j do
        w2[s] := modp1(Shift(w2[s],-d),p);
    od;

    for s from 1 to j do
        v_column[s] :=
        modp1(Add(v_column[s],Multiply(Bv[s],Power(ConvertIn(x,x),d*count))),p);
    od;

    fi;

    k := k + d;
```

```
    count := count + 1;
od;

k := k - (d - l);

v_lower := Vector(j);
v_upper := Vector(j);
q := modp1(ConvertIn(x^l,x),p);

for s from 1 to j do
    v_lower[s] := modp1(Rem(Bv[s],q),p);
    v_upper[s] := modp1(Multiply(Quo(Bv[s],q),q),p);
    v_column[s] :=
    modp1(Add(v_column[s],Multiply(v_lower[s],Power(ConvertIn(x,x),d*count))),p);
od;

newAdd := specmatrixVectorProduct(C,v_upper,p,d,j,j); ## fixing up work matrix. #
newLowAdd := specmatrixVectorProduct(D,v_upper,p,d,n-j,j);#

for i from 1 to n - j do
    w2[i] := modp1(Add(w2[i],newLowAdd[i]),p);
od;

for i from 1 to j do
    w1[i] := modp1(Add(w1[i],newAdd[i]),p);
od;

#Now shift to fix overshoot
for s from 1 to j do
     w1[s] := modp1(Shift(w1[s],-l),p);
od;

for s from 1 to n-j do
    w2[s] := modp1(Shift(w2[s],-l),p);
od;

A[1..j,j+1] := w1;
A[j+1..n,j+1] := w2;

temp := A[j+1,..];
```

```
    A[j+1,..] := A[i_min + j,..];
    A[i_min + j,..] := temp;

    V[j+1] := modp1(ConvertIn(x^k,x),p);
    V[1..j] := v_column;

    return V;
end;


updateInverse := proc(A,B,p,d,j)
    local r,a,w,W,X,Y,Z,Inv,Bw,rBw,rB,i,neg_Bw,G,k;

    W := Matrix(j,j);
    X := Vector(j);
    Y := Matrix(1,j);
    Inv := Matrix(j+1,j+1);
    convertModp1(Inv,p,j+1);

    r := A[j+1,1..j];
    w := A[1..j,j+1];
    a := A[j+1,j+1];

    Bw := matrixVectorProduct(B,w,p,d,j,j);
    rBw := dotProduct(r,Bw,p,j);

    Z := modp1(Gcdex(Subtract(a,rBw),ConvertIn(x^d,x),'s'),p);
    Z := s;

    rB := negateMatrix(vectorMatrixProduct(r,B,p,d,j),p,1,j);

    for i to j do
        Y[1,i] := modp1(Multiply(Z,rB[1,i]),p);
    od;

    neg_Bw := negateVector(Bw,p,j,1);

    for i from 1 to j do
        X[i] := modp1(Multiply(neg_Bw[i],Z),p);
    od;
```

```
    G := outerProduct(neg_Bw,Y,p,d,j); # both negative so should make positive

    for i from 1 to j do
        for k from 1 to j do
            W[i,k] := modp1(Add(B[i,k], G[i,k]),p);
        od;
    od;

    Inv[1..j,1..j] := W;
    Inv[1..j,j+1] := X;
    Inv[j+1,1..j] := Y;
    Inv[j+1,j+1] := Z;

    modPower(Inv,p,d,j+1);
    return Inv;
end;

findDegree := proc(A,p,n)
local i,temp,deg,j ;

deg := modp1(Degree(A[1,1]),p);

for i from 2 to n do
    if modp1(Degree(A[1,i]),p) > deg then
        deg := modp1(Degree(A[1,i]),p);
    fi;
od;

for i from 2 to n do
    for j from 1 to n do
        if modp1(Degree(A[i,j]),p) > deg then
            deg := modp1(Degree(A[i,j]),p);
        fi;
    od;
od;

return deg;

end;
```

```
convertModp1 := proc(A,p,n)
    local i,j;
    for i to n do
        for j to n do
            A[i,j] := modp1(ConvertIn(A[i,j],x),p);
        od;
    od;
end;


dotProduct := proc(a,b,p,n)
    local result, i;
    result := modp1(Zero(x),p);
    for i to n do
        result := modp1(Add(result,Multiply(a[i],b[i])),p);
    od;
    return result;
end;

matrixVectorProduct := proc(A,v,p,d,m,n)
    local b,i;
    b := Vector(m);
    for i to m do
        b[i] := modp1(Rem(dotProduct(A[i,..],v,p,n),ConvertIn(x^d,x)),p);
    od;
    return b;
end;

specmatrixVectorProduct := proc(A,v,p,d,m,n)
    local b,i;
    b := Vector(m);
    for i to m do
        b[i] := dotProduct(A[i,..],v,p,n);
    od;
    return b;
end;

vectorMatrixProduct := proc(v,A,p,d,n)
```

```
    local b,i;
    b := Matrix(1,n);
    for i to n do
        b[1,i] := modp1(Rem(dotProduct(v,A[..,i],p,n),ConvertIn(x^d,x)),p);
    od;
    return b;
end;

outerProduct := proc(u,v,p,d,n)
    local b,B,i,j;
    B := Matrix(n,n);
    for i to n do
        for j to n do
            B[i,j] := modp1(Rem(Multiply(u[i],v[1,j]),ConvertIn(x^d,x)),p);
        od;
    od;
    return B;
end;

negateMatrix := proc(A,p,n,m)
    local B,neg,i,j;
    neg := modp1(Constant(-1,x),p);
    B := copy(A);
    for i to n do
        for j to m do
            B[i,j] := modp1(Multiply(neg,A[i,j]),p);
        od;
    od;
    return B;
end;

negateVector := proc(v,p,n)
    local b,neg,i;
    neg := modp1(Constant(-1,x),p);
    b := copy(v);
    for i to n do
        b[i] := modp1(Multiply(neg,v[i]),p);
    od;
    return b;
end;
```

```
trailingDegree := proc(v,p,d,n)
    local deg,i,temp,allZero,i_min;

    allZero := 1;

    if v[1] <> modp1(Zero(x),p) then

        deg := modp1(Ldegree(v[1]),p);
        i_min := 1;
        allZero := 0;
    else
        #deg := -1;
        i_min := 1;
    fi;

    for i from 2 to n do

        if v[i] <> modp1(Zero(x),p) then

            if allZero = 1 then
                deg := modp1(Ldegree(v[i]),p);
                i_min := i;
                allZero := 0;
            else
                allZero := 0;
                temp := modp1(Ldegree(v[i]),p);

                if temp < deg then
                    deg :=  temp;
                    i_min := i;
                fi;
            fi;
        fi;
    od;

    if allZero = 1 then
        return -1,-1;
    else
        return deg,i_min;
```

```
    fi;
end;

columnShift := proc(A,p,n,d,j)
    local i;
    for i to n do
        A[i,j] := modp1(Shift(A[i,j],-d),p);
    od;

end;

vectorSubtract := proc(u,v,n,p)
    local w,i;

    w := Vector(n);
    for i to n do
        w[i] := modp1(Subtract(u[i],v[i]),p);
    od;
    return w;
end;

canShift := proc(v,p,d,n)
    local deg,i,temp,allZero,i_min;

    allZero := 1;

    if v[1] <> modp1(Zero(x),p) then

        deg := modp1(Ldegree(v[1]),p);
        i_min := 1;
        allZero := 0;

    fi;

    for i from 2 to n do

        if v[i] <> modp1(Zero(x),p) then
            allZero := 0;
            temp := modp1(Ldegree(v[i]),p);
```

```
            if temp < deg then
                deg :=  temp;
                i_min := i;
            fi;
        fi;
    od;

    if allZero = 1 then
        return -1,-1;
    else
        if deg < d then
            return deg,i_min;
        else
            return -1,-1;
        fi;
    fi;
end;

modPower := proc(A,p,d,n)
    local i,j;
    for i to n do
        for j to n do
            A[i,j] := modp1(Rem(A[i,j],ConvertIn(x^d,x)),p);
        od;
    od;
end;
```

Bibliography

[1] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982. 1

[2] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics-Doklady*, 7:595–596, 1963. 3

[3] 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. 1

[4] T. Mulders and A. Storjohann. Rational solutions of singular linear systems. In C. Traverso, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '00*, pages 242–249. ACM Press, New York, 2000. 1, 2

[5] T. Mulders and A. Storjohann. Certified dense linear system solving. *Journal of Symbolic Computation*, 37(4):485–510, 2004. 1

[6] A. Storjohann. High–order lifting. Extended Abstract. In T. Mora, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '02*, pages 246–254. ACM Press, New York, 2002. 35

[7] A. Storjohann. High–order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3–4):613–648, 2003. Extended abstract in [6]. 1