Solving parametric systems using Dixon resultants and sparse interpolation tools

by

Ayoola Isaac Jinadu

M.Sc., University of Saskatchewan, 2017
B.Sc., Federal University of Agriculture, Abeokuta, 2012

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2023

Declaration of Committee

Name:	**Ayoola Isaac Jinadu**
Degree:	**Doctor of Philosophy**
Thesis title:	**Solving parametric systems using Dixon resultants and sparse interpolation tools**
Examining Committee:	**Chair:** Luis Goddyn Professor, Mathematics
	Michael Monagan Supervisor Professor
	Petr Lisoněk Committee Member Professor
	Nils Bruin Internal Examiner Professor
	George Labahn External Examiner Professor David R. Cheriton School of Computer Science Department of Mathematics University of Waterloo
Date Defended:	**August 23, 2023**

Abstract

Many elimination techniques such as Gröbner bases and Triangular sets have been employed to address the growing demand for solving parametric polynomial systems in practice. However, experiments have shown that these elimination methods when used in computer algebra systems such as Maple and Magma often fail on systems that have many parameters; they can take a very long time to execute or run out of memory.

To address this problem, this thesis presents a new interpolation algorithm for solving parametric polynomial systems (systems of n polynomial equations involving n variables and m parameters with rational coefficients) over $\mathbb{Q}$ using Dixon resultants. The Dixon resultant R of a parametric polynomial system is a multiple of the unique monic generator of an elimination ideal of a polynomial system, and it can be expressed as the determinant of a matrix M of polynomial entries called the Dixon matrix.

Given a black box for the Dixon resultant $R = \det(M)$ (we evaluate the Dixon matrix M at integer points modulo primes and compute determinant of integer matrices modulo primes), we present a new Dixon resultant algorithm that interpolates the monic square-free factors R_j of the Dixon resultant R from monic univariate polynomial images of R.

This new Dixon resultant algorithm uses our newly developed sparse multivariate rational function interpolation method over $\mathbb{Q}$ to interpolate the rational function coefficients of the monic square-free factors modulo primes. It further uses rational number reconstruction and Chinese remaindering to recover the rational coefficients of the R_j's.

We have made a hybrid Maple and C implementation of our Dixon resultant algorithm. Our benchmarks show that our new Dixon resultant algorithm can solve many parametric polynomial systems that other algorithms for computing R are unable to solve. However, the new Dixon resultant algorithm may fail to produce an answer, and even when it is successful, the returned answer might be wrong with provably low probability. Consequently, we identify and classify all the causes of failure in our new algorithm, and we give a detailed failure probability analysis and complexity analysis of our new Dixon resultant algorithm.

Furthermore, we consider another related problem. Let $Ax = b$ be a parametric linear system such that the coefficient matrix A is of full rank. In general, the solutions x_i will be rational functions in the parameters. We present a new black box algorithm for interpolating

the entries x_i using our new sparse multivariate rational function interpolation method. We present timing results comparing our hybrid Maple and C implementation of our new algorithm with four other algorithms in Maple for solving $Ax = b$. A failure probability analysis and complexity analysis for our new algorithm is also presented.

Keywords: Parametric Polynomial systems; Parametric Linear Systems; Dixon Resultants; Sparse Multivariate Rational Function Interpolation; Black Box; Kronecker Substitution; Failure Probability.

Acknowledgements

I am incredibly grateful to God who has made the completion of this thesis possible. Thank you Lord for answering me every time I call upon you. I am nothing without your help.

To my beautiful wife Adewunmi and my amazing daughter Oreoluwa, your unwavering support, kindness, patience and love have been the pillars of strength that sustained me during the challenging times. Your belief in my abilities and the countless words of encouragement have kept me motivated throughout my PhD program. Thank you guys for being so patient with me for many countless hours that I spent away while working on this thesis.

Many thanks to my wonderful supervisor Dr. Michael Monagan for his invaluable guidance throughout this PhD journey. Your expertise, technical help and constructive feedback have been instrumental in shaping the direction and quality of this thesis.

I would like to thank my PhD thesis committee members: Dr. George Labahn, Dr. Petr Lisonek, and Dr. Nils Bruin, for being present at my thesis defense and providing me with useful feedback.

I would like to thank RCCG Grace Chapel (Surrey Central CFC) for their prayers and words of encouragement during our CFC meetings.

Finally, I would like to thank my advisor Dr. Michael Monagan and the Department of Mathematics at SFU for providing me with travel funding which gave me the opportunity to present my PhD work at many conferences (France, Turkey and Cuba).

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Using sparse polynomial interpolation, sparse rational function interpolation and rational number reconstruction, we develop new algorithms for solving parametric linear and polynomial systems over $\mathbb{Q}$. In particular, this thesis presents a new interpolation algorithm for solving parametric polynomial systems using Dixon resultants and a new algorithm for solving parametric linear systems, both using sparse rational function interpolation.

We use sparse interpolation tools such as sparse rational function interpolation and sparse polynomial interpolation because we want the complexity of our new algorithms to depend on the actual number of terms in the objects that we are computing (a polynomial or a rational function) instead of the maximum number of possible terms. Another reason why sparse interpolation tools are preferred is because many multivariate polynomials and multivariate rational functions that we have to interpolate in practice are sparse, and often very sparse. To give the reader an idea of what sparsity means, we begin with the definition of a sparse polynomial and a sparse rational function.

A non-zero polynomial f with t terms is generally defined as being sparse if t is relatively much less than the maximum number of possible terms in f. However, this definition of a sparse polynomial is somewhat imprecise, so we give the following formal definition.

Definition 1.1. Let f be a non-zero polynomial in $y_1, y_2, \ldots, y_m$. Let t denote the number of non-zero terms in f such that $t \geq 1$ and let $d = \deg(f)$ be the total degree of f. The maximum number of possible terms in f is $M = \binom{m+d}{d}$. We say that f is sparse if $t < \sqrt{M}$.

Example 1.2. Let $f = 3y_1y_2y_5 + 5y_1y_3^3y_4$. Notice that $t = 2, d = 5, m = 5$ and $M = \binom{5+5}{5} = \binom{10}{5} = 252$. Therefore, f is a sparse polynomial since $t = 2 < \sqrt{M} = \sqrt{252} \approx 15.9$.

Definition 1.3. A sparse polynomial f is normally represented as

$$f = \sum_{k=1}^{t} a_k y_1^{d_{k,1}} y_2^{d_{k,2}} \cdots y_m^{d_{k,m}}, \quad \text{where } a_k \neq 0.$$

Definition 1.4. A multivariate rational function f/g with $\gcd(f, g) = 1$ is sparse if both polynomials f and g are sparse.

1.1 Black Box Model

In most science fields, such as cryptography [Goldwasser and Rothblum, 2007], a black box is a device or a system in which its inputs and output are known, but the internal functionality is completely unknown. It could be almost anything as long as its implementation is not transparent.

The first black box model in computer algebra was introduced by Kaltofen and Trager in [Kaltofen and Trager, 1990] where algorithms for computing the greatest common divisor of two polynomials represented by two black boxes were presented. They also argued that black box representations are space efficient. A black box can be constructed for objects such as a polynomial, a rational function, or the determinant of a matrix of polynomials. A function call to the black box is referred to as a **black box probe**.

In all of the problems addressed in this thesis, we aim to compute factors (or some factors) of the determinant of a matrix with multivariate polynomial entries. Let D be a matrix with multivariate polynomial entries and suppose $\det(D)$ is a sparse polynomial. The number of terms in the factors of $\det(D)$ over $\mathbb{Z}$ is typically less than the number of terms in $\det(D)$. So, using a black box model in our proposed algorithms is beneficial because we save the memory space needed to store $\det(D)$ and we save the cost of evaluating $\det(D)$.

For our purposes, the internal functionality of a black box is assumed to be known. That is, we are able to modify a given black box that works over $\mathbb{Q}$ to work modulo a prime p. In particular, the black box for our proposed algorithm for solving parametric polynomial systems over $\mathbb{Q}$ will represent a determinant computation. We will construct the black box **BB** from a given matrix D of polynomials in $x_1, y_1, y_2, \ldots, y_m$ over $\mathbb{Q}$ to evaluate D at a point $\alpha \in \mathbb{Z}_p^{m+1}$ where p is prime and then compute the determinant of the integer matrix $D(\alpha) \bmod p$. We will then input **BB** to our new algorithm for solving parametric polynomial systems over $\mathbb{Q}$ which treats it as a black box and not look inside. That is, our new algorithm for solving parametric polynomial systems over $\mathbb{Q}$ is only aware that $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$.

For our proposed algorithm for solving linear systems involving parameters $y_1, y_2, \ldots, y_m$, we will construct a similar black box **BB** for the corresponding augmented matrix so that it accepts an evaluation point $\alpha \in \mathbb{Z}_p^m$ and a prime p to output a vector of integers modulo p. That is, $\mathbf{BB} : (\mathbb{Z}_p^m, p) \to \mathbb{Z}_p^n$ where n is the rank of the coefficient matrix of the parametric linear system.

We can also build black boxes for known objects such as polynomials and rational functions following our usual black box construction. Therefore, in this thesis, a black box is a computer program that takes a point $\alpha \in \mathbb{Z}_p^m$ and a prime p as two inputs and outputs the evaluation of the represented object modulo p. We note that our black box model is similar to the modular black box model described in [Giesbrecht and Roche, 2010]. In Figure

1.1, the black box for $\det(D) \in \mathbb{Q}[x_1, y_1, y_2, \ldots, y_m]$ takes an evaluation point $\alpha \in \mathbb{Z}_p^{m+1}$ and a prime p as inputs and it outputs $\det(D(\alpha)) \bmod p$.

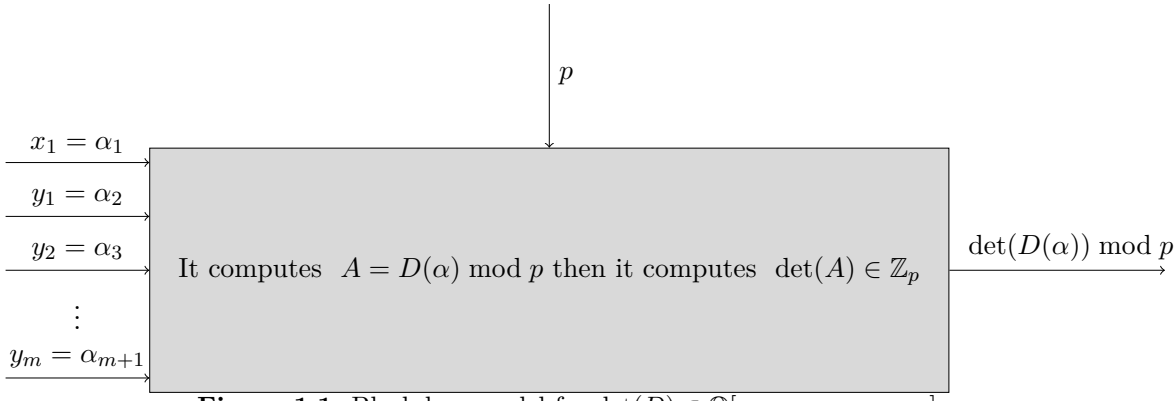

Figure 1.1: Black box model for $\det(D) \in \mathbb{Q}[x_1, y_1, y_2, \ldots, y_m]$

Example 1.5. Let prime $p = 11$ and let

$$D = \begin{pmatrix} x_1 & y_1 \\ x_1 + y_1 & y_2 \end{pmatrix}.$$

Suppose we input the evaluation point $\alpha = (1, 1, 1)$ and p to the black box for $\det(D)$, it outputs the $\det(D)$ evaluated at α over $\mathbb{Z}_p$ which is 10.

1.2 Motivation and Contributions

This thesis addresses two main problems, namely,

(1) Solving parametric polynomial systems by computing their Dixon resultants and

(2) Solving $Ax = b$ with parameters using sparse rational function interpolation.

1.2.1 Solving parametric polynomial systems using Dixon resultants

Let $X = \{x_1, \ldots, x_n\}$ be the set of variables and let $Y = \{y_1, \ldots, y_m\}$ be the set of parameters with $n \geq 2$ and $m \geq 1$. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Q}[Y][X]$ such that $|\mathcal{F}| = |X|$. We refer to $\mathcal{F}$ as a parametric polynomial system where each $\hat{f}_i$ is a polynomial in variables X with coefficients in the polynomial ring $\mathbb{Q}[Y]$. Let $I = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle$ be the ideal generated by $\mathcal{F}$ and let $J = I \cap \mathbb{Q}(Y)[x_1]$ be the elimination ideal in $\mathbb{Q}(Y)[x_1]$.

In this thesis, solving a parametric polynomial system $\mathcal{F}$ means we aim to eliminate variables $x_2, x_3, \ldots, x_n$ from $\mathcal{F}$ in order to obtain the resultant $\overline{R}$ of $\mathcal{F}$. For our purposes, we define the resultant of a parametric polynomial system $\mathcal{F}$ as follows.

Definition 1.6. The resultant $\overline{R}$ of a parametric polynomial system $\mathcal{F}$ is defined to be the unique monic generator of the elimination ideal $J = I \cap \mathbb{Q}(Y)[x_1]$ and any factor $\tilde{e}$ is called an extraneous factor if $g = \tilde{e}\overline{R}$ for any non-zero $g \in J$.

Note that Definition 1.6 agrees with the resultant definitions given by Kapur and Saxena in [Kapur and Saxena, 1997] and by Minimair in [Minimair, 2014]. Based on our definition of the resultant, some obvious facts follow: The zero polynomial is an element of the elimination ideal J. The resultant $\overline{R}$ may be irreducible or reducible over $\mathbb{Q}(Y)$. If a monic polynomial in the elimination ideal J is irreducible over $\mathbb{Q}(Y)$, then it must be the resultant $\overline{R}$. In this thesis, we do not aim to compute the roots of $\overline{R}$.

Example 1.7. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2\} \subset \mathbb{Q}[y_1, y_2][x_1, x_2]$ where

$$\hat{f}_1 = x_1x_2 + y_1, \quad \hat{f}_2 = x_2y_1 + y_2^2x_1.$$

By eliminating variable x_2 from $\mathcal{F}$, we may compute the resultant

$$\overline{R} = x_1^2 - \frac{y_1^2}{y_2^2} \in \mathbb{Q}(y_1, y_2)[x_1]$$

which is clearly reducible over $\mathbb{Q}(y_1, y_2)$.

The general methods used in elimination theory are triangular sets [Maza, 2000, Kalkbrener, 1991, Wu], Gröbner bases [Buchberger, 2006] and resultant methods. Resultants are classical algebraic objects for determining whether a polynomial system has a solution (common root) or not without actually solving for the solutions of the system explicitly. To the best of our knowledge, the resultant methods that are widely used for solving parametric polynomial systems are based on the Sylvester, Dixon [Dixon, 1908, 1909], Macaulay [Macaulay, 1902, 1916] and sparse resultant formulations [Bhayani et al., 2020]. These resultant methods all construct matrices (resultant matrices) whose determinant is the resultant or a multiple of the resultant of a given parametric polynomial system. We refer to the matrix obtained due to Dixon's method as the Dixon matrix and its determinant as the Dixon resultant.

Definition 1.8. A collection of polynomials $\{f_1, f_2, \ldots, f_{n+1}\}$ is said to be generic n-degree if there exist non-negative integers $d_1, d_2, \cdots, d_n$ such that for $i \neq j$,

$$d_k = \deg(\hat{f}_i, x_k) = \deg(\hat{f}_j, x_k) \text{ for } 1 \leq k \leq n,$$

and

$$\hat{f}_j = \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n} a_{j,i_1,i_2,\cdots,i_n} x_1^{i_1} \cdots x_n^{i_n} \text{ for } 1 \leq j \leq n+1,$$

where each coefficient $a_{j,i_1,i_2,\cdots,i_n}$ is a distinct indeterminate (think of parameters!) and the total degree of f_j is $\sum_{j=1}^{n} d_j$. In plain language, generic polynomials are polynomials that

have every possible coefficient, all coefficients are indeterminates, and they have the same degrees in each variable.

Using the size bounds given in [Kapur and Saxena, 1995], we compare the size of three resultant matrices (Dixon, Macaulay and sparse resultant) using two measures, namely, the total degree and the partial degree in each variable for four generic polynomial systems. The number of polynomials, the total degree, and the partial degree of the polynomials in each variable of these systems are denoted by $n, d, d_{\max}$ respectively in Tables 1.1 & 1.2.

n	d	**Dixon**	**Macaulay**	**Sparse resultant**
2	2	6	15	15
2	3	15	36	36
3	2	20	56	56
3	3	84	220	220

Table 1.1: Matrix size of four generic systems using their total degrees

n	$d_{\max}$	**Dixon**	**Macaulay**	**Sparse resultant**
2	2	8	66	36
2	3	18	153	81
3	2	48	2024	512
3	3	162	7140	1728

Table 1.2: Matrix size of four generic systems using their partial degrees

As the reader can see from the data reported in Tables 1.1 & 1.2, the size of the Dixon matrix is smaller when compared to the matrices obtained from the sparse and Macaulay resultant methods. Therefore, we are interested in the Dixon resultant formulation because of its computational advantage.

Our main motivation for studying Dixon resultants stems from the sets of parametric polynomial systems listed in Lewis's papers [Lewis, 2017, 2018b, 2020]. Lewis tried to solve these polynomial systems using Gröbner bases and triangular sets in Maple and Magma, but they often failed badly; they took a very long time to execute and often ran out of memory. The failure of these methods is due to the intermediate expression swell (the rational function coefficients in the parameters of the intermediate polynomials over $\mathbb{Q}$ are huge) caused by the parameters.

In Table 1.3, we report the timings obtained when we attempted to solve some of the parametric polynomial systems from [Lewis, 2017] using Gröbner bases and triangular sets in Maple and Magma. The names of the real parametric polynomial systems are labelled as **Names** in Table 1.3. The quantity $\#\overline{R}$ denotes the number of terms in the resultant

$\overline{R}$ when its fractions in the parameters are cleared, n is the number of polynomials and m is the number of parameters in the input parametric systems. Columns **Maple-Gröbner** and **Maple-Triangular** contain the timings obtained using Gröbner basis and Triangular sets in Maple. Column **Magma** contains the timings obtained for solving these systems in Magma using Gröbner bases. The input parametric polynomial systems in Table 1.3 are listed in the Appendix A of this thesis with the monomial ordering used in our experiments.

Table 1.3: Timings showing the performance of 3 elimination methods on real parametric systems

Names	n	m	$\#\overline{R}$	**Maple-Gröbner**	**Maple-Triangular**	**Magma**
Heron3d	6	6	23	0.05 s	0.37s	1.35s
Heron4d	10	10	131	0.08 s	!	42s
Heron5d	15	15	823	>5 days	!	2.6 days
Circle	4	9	-	>5 days	>5 days	>5 days
Robot-x_1	4	7	44	!	!	>5 days

The notation ! means Maple ran out of memory and $-$ means unknown

As the reader can see in Table 1.3, these elimination techniques are ineffective on the selected parametric systems and our timings are consistent with the findings reported in [Lewis, 2017].

To address this problem, a new interpolation algorithm for solving parametric polynomial systems using the Dixon resultant formulation [Dixon, 1908, 1909] is proposed in this thesis. The Dixon resultant R of a parametric polynomial system $\mathcal{F}$ in x_1 is a multiple of the unique monic generator of the elimination ideal $J = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle \cap \mathbb{Q}(Y)[x_1]$ [Dixon, 1908, 1909, Kapur et al., 1994] and it can be expressed as the determinant of a matrix of polynomial entries called the Dixon matrix (See Chapter 2). It is used to eliminate $n-1$ variables from a polynomial system with n polynomial equations involving n variables.

We begin with the historical developments of Dixon resultants, and we provide examples whenever necessary. Bezout developed a method to compute the resultant of two univariate polynomials in 1764. His method involves the construction of a resultant matrix (the Bezout matrix) whose determinant produces the resultant (the Bezout resultant). The Bezout matrix is known to be smaller in size than the Sylvester matrix. Many years later, Cayley noticed that Bezout's method was complicated and reformulated it [Cayley, 1857, 1865], leading to the *first case of the Dixon resultant formulation.*

We describe Cayley's reformulation of Bezout's method as follows. Let $\hat{f}(x)$ and $\hat{g}(x)$ be two univariate polynomials of degree d. Form the polynomials $\hat{f}(z)$ and $\hat{g}(z)$ by replacing x with a new variable z. Then construct the polynomial

$$\Delta(x,z) = \begin{vmatrix} \hat{f}(x) & \hat{g}(x) \\ \hat{f}(z) & \hat{g}(z) \end{vmatrix} = \hat{f}(x)\hat{g}(z) - \hat{f}(z)\hat{g}(x).$$

Observe that $\Delta(z,z) = 0$. So, the polynomial $\Delta(x,z)$ can be written as the product of $(x-z)$ and a polynomial δ whose partial degree in variables x and z is $d-1$. Next, compute the degree $d-1$ polynomial

$$\delta(x,z) = \frac{\Delta(x,z)}{x-z} = \frac{\hat{f}(x)\hat{g}(z) - \hat{f}(z)\hat{g}(x)}{x-z}$$

which vanishes at every root of $\hat{f}(x)$ and $\hat{g}(x)$ regardless of the value of z.

Thus, at every common root of $\hat{f}(x)$ and $\hat{g}(x)$, every coefficient of δ in z^i, which is a polynomial in x, must be zero (See Example 1.9). Therefore, a system of homogeneous linear equations can be generated by extracting the coefficients of $\delta(x,z)$ in z^i. The coefficient matrix of the formed linear system is the resultant matrix, while the right-hand side vector is the zero vector. Note that the column vector containing the powers of x can be viewed as a vector whose entries are indeterminates, i.e., $u_i = x^i$. Since this column vector will have an entry $u_0 = x^0 = 1$, it follows that the linear system has a non-trivial solution. Therefore, the determinant of the coefficient matrix must be zero, and this gives a necessary condition for $\hat{f}(x)$ and $\hat{g}(x)$ to have a solution [Kapur et al., 1994, Lewis, 1996].

Example 1.9. Let

$$\hat{f}(x) = (x-3)(x-5)(x+4) \text{ and } \hat{g}(x) = (x-3)(x+7)(x+8).$$

The common root of both univariate polynomials is clearly 3. We have

$$\begin{aligned}\Delta(x,z) &= \begin{vmatrix} (x-3)(x-5)(x+4) & (x-3)(x+7)(x+8) \\ (z-3)(z-5)(z+4) & (z-3)(z+7)(z+8) \end{vmatrix} \\ &= 4\,(z-3)\,(x-3)\,(4xz+19x+19z+61)\,(-z+x)\,.\end{aligned}$$

Thus,

$$\delta(x,z) = \underbrace{\left(16x^2+28x-228\right)}_{-\hat{f}+\hat{g}} z^2 + \underbrace{\left(28x^2-68x-48\right)}_{(-x-12)\hat{f}+(-4+x)\hat{g}} z + \underbrace{\left(-228x^2-48x+2196\right)}_{(3x+45)\hat{f}+(-3x+3)\hat{g}}.$$

Hence, Cayley's resultant matrix

$$M = \begin{bmatrix} 16 & 28 & -228 \\ 28 & -68 & -48 \\ -228 & -48 & 2196 \end{bmatrix}$$

and $\det(M) = 0$.

Dixon presented a method to obtain the resultant of three generic polynomials in two variables of bi-degree (d_1, d_2), which is a generalization of the Cayley-Bezout method for

computing the resultant of two univariate polynomials [Dixon, 1908, 1909]. Dixon also described how his method can be extended to compute the Dixon resultant of a system of $n+1$ generic n-degree polynomials in n variables (See Definition 1.8 for the definition of generic n-degree polynomials). Dixon proved that the vanishing of the Dixon resultant is a necessary and sufficient condition for the existence of a common root to a system of $n+1$ generic n-degree polynomials in n variables [Kapur et al., 1994]. However, for geometric problems arising in practice, the Dixon resultant is almost always zero because these problems do not have a generic degree shape [Kapur et al., 1994]. The singularity of a Dixon matrix is a major problem because no useful information is provided about the existence of a common root to a given polynomial system. Therefore, Dixon's necessary and sufficient condition is of little use in practice. Another shortcoming of Dixon's approach is that it is limited to a small class of polynomial systems. In particular, if it is applied to a polynomial system that is not a generic n-degree polynomial system, one may obtain a rectangular Dixon matrix [Kapur and Saxena, 1995] with no suitable way to compute the Dixon resultant.

The reader should note that a non $n+1$ generic, non n-degree polynomial system can be converted to a $n+1$ generic n-degree polynomial system so that Dixon's method can be applied. But most often, this approach is inefficient, as higher degree polynomials are created and the new Dixon matrix created relative to the generic polynomial system is still singular (See [Kapur et al., 1994, Section 2.3, Page 105]). These limitations stunted the use and popularity of the Dixon resultant formulation for years, since it can only be used to solve a small class of polynomial systems. But in 1994, Kapur, Saxena and Yang in [Kapur et al., 1994] addressed the shortcomings of Dixon's method.

Kapur, Saxena and Yang proved that the determinant of any square sub-matrix M of the Dixon matrix D such that $\text{rank}(M) = \text{rank}(D)$ is an element of the elimination ideal $J = I \cap \mathbb{Q}(Y)[x_1]$. Thus, once a Dixon matrix is constructed, we may find any maximal rank sub-matrix M of the Dixon matrix D, and compute its determinant. Therefore, the requirement for a polynomial system $\mathcal{F}$ to be generic n-degree is no longer necessary.

Extracting a sub-matrix M of maximal rank from the Dixon Matrix D

A sub-matrix M of maximal rank can be determined from the Dixon matrix D using fraction-free Gaussian elimination [Kapur et al., 1994] in $\mathbb{Q}[Y, x_1]$. However, this can result in expression swell. Instead, we select a sub-matrix M of maximal rank from D using the following Monte-Carlo method. We pick a random 62 bit prime p and choose an evaluation point $\beta \in \mathbb{Z}_p^{m+1}$ at random where m is the number of parameters. Then we compute $B = D(\beta)$ over $\mathbb{Z}_p$ and identify M from B in D with high probability.

Our idea requires performing ordinary Gaussian elimination over $\mathbb{Z}_p$ only, and in contrast to [Kapur et al., 1994] crucially avoids doing polynomial arithmetic in $\mathbb{Q}[Y, x_1]$. A new algorithm (Algorithm 5) for extracting a sub-matrix of maximal rank from a Dixon matrix

is presented in Subsection 2.2.4, and we give a failure probability bound for the algorithm using the Schwartz-Zippel Lemma [Schwartz, 1980, Zippel, 1979].

Previous methods for computing Dixon resultants

Assuming the Dixon matrix obtained from a given parametric polynomial system $\mathcal{F}$ is neither singular nor rectangular, the dominating step in computing Dixon resultants is computing the determinant of the Dixon matrix [Minimair, 2017]. To the best of our knowledge, the first and only interpolation method that has ever been applied to Dixon resultants was done by Kapur and Saxena [Kapur and Saxena, 1995] in 1995. They used Zippel's sparse interpolation from [Zippel, 1979] to interpolate the Dixon resultant R.

Zippel's sparse interpolation does $O(mdt)$ black box probes for the first image of R modulo a prime, where m is the number of parameters, d is the maximum partial degree of R, $t = \#R$ and p is chosen so that arithmetic in $\mathbb{Z}_p$ can be done in the hardware. But one has to recover the integer coefficients of R which may need more primes. Using the support of the first image of R modulo a prime (the list of monomials without their coefficients), the integer coefficients of the Dixon resultant R can be recovered using $O(t)$ probes to the black box for each subsequent prime [Zippel, 1979].

In 2015, Lewis developed the Dixon-EDF (Early Detection Factor) algorithm for computing the Dixon resultant because he could not solve many of the systems he listed in [Lewis, 2017] using Gröbner bases and Triangular sets in both Maple and Magma. The Dixon-EDF algorithm is a variant of the Gaussian elimination algorithm. It is a modified row reduction of the Dixon matrix that factors out the greatest common divisor of each pivot row at each step. The Dixon-EDF algorithm is able to detect some factors of the Dixon resultant early. One can interrupt it part way to switch to another method. Lewis often switches to the Gentleman & Johnson minor expansion algorithm [Gentleman and Johnson, 1974] to finish the computation. The drawback of the Dixon-EDF method is that it is not automatic, and expression swell may occur when computing in $\mathbb{Q}[Y, x_1]$. The implementation of the Dixon-EDF method was done in Fermat; a computer algebra system designed and implemented by Lewis whose built-in multivariate gcd algorithm uses Zippel's gcd algorithm [Lewis, 2004]. Another variation of the Dixon-EDF method called PRDF(Pivot Row Detection of Factors) was designed and implemented by Minimair in Maple [Minimair, 2017]. This method requires fewer gcd computations than Lewis' Dixon-EDF method.

A new interpolation algorithm for computing Dixon resultants

We begin with a description of our new Dixon resultant algorithm. Let the Dixon resultant R in x_1 of a parametric polynomial system $\mathcal{F}$ over $\mathbb{Q}$ be written as

$$R = \sum_{k=0}^{\hat{d}} \bar{r}_k(y_1, \ldots, y_m) x_1^k \in \mathbb{Q}[y_1, y_2, \ldots, y_m][x_1]$$

where $\hat{d} = \deg(R, x_1)$. If $\hat{d} = 0$, this means that the given parametric polynomial system $\mathcal{F}$ does not have a solution. Now let $\hat{d} > 0$ and let $C = \gcd(\bar{r}_0, \bar{r}_1 \cdots, \bar{r}_{\hat{d}})$ be the polynomial content of the Dixon resultant R.

In practice, when R factors over $\mathbb{Q}$, it often has many repeated factors with large degrees and a large unwanted polynomial content C. To avoid unwanted factors in R, we will compute the monic square-free factors R_j of R and not the Dixon resultant R in expanded form. The monic square-free factorization of the Dixon resultant R is a factorization of the form

$$\hat{r} \prod_{j=1}^{l} R_j^j$$

where each R_j can be written as

$$R_j = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \frac{f_{jk}(y_1, y_2, \ldots, y_m)}{g_{jk}(y_1, y_2, \ldots, y_m)} x_1^{d_{jk}} \in \mathbb{Q}(y_1, y_2, \ldots, y_m)[x_1]$$

for non-zero $f_{jk}, g_{jk} \in \mathbb{Q}[y_1, y_2, \ldots, y_m]$ such that

(1) $\hat{r} = C/L$ for some $L \in \mathbb{Q}[Y]$,

(2) each R_j is monic and square-free in $\mathbb{Q}(Y)[x_1]$,

(3) $\gcd(R_i, R_j) = 1$ for $i \neq j$,

(4) $\gcd(f_{jk}, g_{jk}) = 1$ for all $0 \leq k \leq T_j - 1$, and

(5) $\mathrm{LC}(g_{jk}) = 1$.

This monic square-free factorization exists and it is unique [von zur Gathen and Gerhard, 2013, Section 14.6]. We remark that the monic square-free factors R_j of R are not necessarily irreducible over $\mathbb{Q}(y_1, y_2, \ldots, y_m)$. To give the reader an idea of what we are computing, we give the following real example from [Lewis, 2017, Section 8, Page 247].

Example 1.10 (Robot arms parametric system Appendix A.1)**.** Consider the parametric system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_7][x_1, x_2, x_3, x_4]$ listed in Appendix A.1. Let

$$C = -65536 \left(y_2^2 + 1\right)^8 \left(y_2^2 y_4^2 + 2y_2^2 y_4 y_5 + y_2^2 y_5^2 + y_4^2 - 2y_4 y_5 + y_5^2\right)^4 y_4^8$$

$$A_1 = x_1^2 + 1$$

$$\begin{aligned} A_2 = {} & (y_2^2 y_3^2 + 2y_2^2 y_3 y_6 - y_2^2 y_4^2 - 2y_2^2 y_4 y_5 - y_2^2 y_5^2 + y_2^2 y_6^2 + y_2^2 y_7^2 + y_3^2 + 2y_3 y_6 - y_4^2 + 2y_4 y_5 - y_5^2 \\ & + y_6^2 + y_7^2) x_1^2 + (-4y_2^2 y_3 y_7 - 4y_3 y_7) x_1 + y_2^2 y_3^2 - 2y_2^2 y_3 y_6 - y_2^2 y_4^2 - 2y_2^2 y_4 y_5 - y_2^2 y_5^2 + y_2^2 y_6^2 \\ & + y_2^2 y_7^2 + y_3^2 - 2y_3 y_6 + 2y_4 y_5 - y_5^2 + y_6^2 + y_7^2 - y_4^2 \end{aligned}$$

$$A_3 = \left(y_1^2 + 2y_1 y_4\right) x_1^2 + y_1^2 - 4y_1 y_3 + 2y_1 y_4 + 4y_3^2 - 4y_3 y_4$$

$$A_4 = \left(y_1^2 - 2y_1 y_4\right) x_1^2 + y_1^2 - 4y_1 y_3 - 2y_1 y_4 + 4y_3^2 + 4y_3 y_4.$$

By eliminating variables $\{x_2, x_3, x_4\}$ from $\mathcal{F}$, we determined that the Dixon resultant R of the robot arms system in x_1 has **6,924,715** terms in expanded form and it factors as

$$CA_1^{\mathbf{24}}A_2^{\mathbf{4}}A_3^{\mathbf{2}}A_4^{\mathbf{2}}.$$

Our new Dixon resultant algorithm computes R_1, R_2 and R_3 only where

- $R_1 = A_1$,
- $R_2 = \text{monic}(A_2, x_1)$ and
- $R_3 = \text{monic}(A_3A_4, x_1)$.

Note that the largest polynomial coefficient of R_1, R_2 and R_3 to be interpolated by our new Dixon resultant algorithm is the leading coefficient of A_2 which has only 14 terms, unlike Kapur and Saxena in [Kapur and Saxena, 1995] who would interpolate **6,924,715** terms of R using Zippel's algorithm. Also, observe that R_1 and R_2 are irreducible over $\mathbb{Q}(y_1, y_2, \ldots, y_m)$ but R_3 is not.

Let M be the Dixon matrix of polynomial entries in $x_1, y_1, \ldots, y_m$ and let $R = \det(M)$ be the Dixon resultant. Given a black box $\mathbf{BB} : \left(\mathbb{Z}_p^{m+1}, p\right) \to \mathbb{Z}_p$ for the Dixon resultant R, we develop a new Dixon resultant algorithm that probes the black box **BB** and interpolates the monic square-free factors R_j of R from monic univariate polynomial images of R in x_1 using sparse multivariate rational function interpolation to recover the coefficients of R_j in $\mathbb{Q}(Y)$ modulo primes, and uses Chinese remaindering and rational number reconstruction to recover the rational coefficients of R_j. Note that an implication of using a black box model for the Dixon resultant R is that many important properties of R such as the number of terms and the variable degrees of R are unknown, so we have to interpolate them probabilistically since the Dixon resultant R is unknown. Otherwise, we have to use bounds for the number of terms and the variable degrees which are often very high.

We interpolate the R_j's because it is relatively cheap to compute a square-free factorization of a monic polynomial image of R in x_1 and the square-free factorization factors will be consistent from one image to the next with high probability. By not interpolating the repeated factors or the polynomial content of R, the number of polynomial terms to be interpolated and the number of black box probes and the number of primes needed to recover the R_j's are often significantly reduced. Due to this reduction, our new interpolation algorithm performs favourably on real parametric systems when compared to other algorithms for computing R directly (See Subsection 5.4.3 for benchmarks).

First Attempt

Definition 1.11. The monic square-free part S of the Dixon resultant R is the product of the monic square-free factors R_j, that is, $S = \prod_{j=1}^{l} R_j$.

In a preliminary stage of this work (when my thesis proposal was defended), we first designed and implemented our Dixon resultant algorithm to interpolate the monic square-free part S from monic univariate images of R in x_1. But we discovered that when the number of the monic square-free factors $l > 1$, interpolating the R_j's instead of S often reduces the number of black box probes required. These savings are realized because there is a further reduction in the number of terms in the largest polynomial coefficient of R_j to be interpolated compared to the monic product S. Also, the same monic univariate polynomial images of R in x_1 that yield the first monic square-free factor R_1 can be re-used to recover subsequent monic square-free factors in R.

	x_1	x_2	x_3	x_4
# of black box probes required to interpolate S	$222,301$	$3,137,373$	$116,741$	$5,531,491$
# of black box probes required to interpolate the R_j's	$19,241$	$1,210,889$	$116,741$	$1,335,853$
Savings in # of black box probes	$203,060$	$1,926,484$	0	$4,195,638$
Number of R_j's	3	3	2	3
# of terms in the largest polynomial coefficient of R_j	14	691	85	624
# of terms in the largest polynomial coefficient of S	106	$2,200$	85	$2,388$

Table 1.4: Interpolating monic square-free part S versus interpolating the square-free factors R_j

Table 1.4 contains the number of black box probes required for interpolating the monic square-free part S versus interpolating the monic square-free factors R_j one at a time for the *robot arms* problem and it shows a significant reduction in the number of black box probes when the main variable is x_1, x_2 or x_4. Notice in column x_3 that both methods used the same number of black box probes. This is because the number of terms in the largest polynomial coefficient of R_j and S is the same. Thus, no gain is realized in terms of the number black box probes used, even though we save some time when we perform rational function interpolation while the number of the monic square-free factors for this case is more than 1.

Using a Kronecker substitution on the parameters

The main goal we had in mind during the design of our Dixon resultant algorithm was to ensure that our new algorithm uses the fewest number of black box probes to interpolate the monic square-free factors R_j of R. This is essential because a probe to the black box entails evaluating a Dixon matrix of polynomial entries over $\mathbb{Z}_p$ at random points and also computing determinant of integer matrices over $\mathbb{Z}_p$, which are not cheap computations as the Dixon matrix can be large.

To this end, we adapt the sparse multivariate rational function interpolation algorithm of Cuyt and Lee for our purposes [Cuyt and Lee, 2011]. As far as we are aware, it is the best sparse multivariate rational function algorithm because it uses fewer black box probes than the methods in [Kaltofen and Trager, 1990, Kaltofen and Yang, 2007, de Kleine et al., 2005]. The Cuyt and Lee algorithm must be combined with a sparse polynomial

interpolation algorithm to produce sets of auxiliary univariate rational functions that are densely interpolated in normalized form (there is a constant term 1 in their denominator coefficients) and sparse interpolation is performed using their coefficients to produce the desired sparse multivariate rational function. To use the fewest number of black box probes possible, we modify Cuyt and Lee's rational function interpolation algorithm [Cuyt and Lee, 2011] to use the Ben-Or/Tiwari algorithm [Ben-Or and Tiwari, 1988] as the main sparse polynomial algorithm in our Dixon resultant algorithm.

Let $f = \sum_{k=1}^{t} a_k M_k(y_1, \ldots, y_m) \in \mathbb{Z}[y_1, \ldots, y_m]$ with $a_k \neq 0$ be a sparse polynomial. The Ben-Or/Tiwari algorithm interpolates f using $2T$ evaluation points which are of the form $\{(2^j, 3^j, \ldots, p_m^j) : 0 \leq j \leq 2T-1\}$ where p_m is the m-th prime assuming a term bound $T \geq t$ is known. Let $\hat{m}_i = M_i(2, 3, \ldots, p_m)$ be the monomial evaluation. To avoid intermediate expression swell, the Ben-Or/Tiwari algorithm must be done modulo a prime p satisfying $p > \max_{i=1}^{t} \hat{m}_i \leq p_m^d$ where $d = \deg(f)$. However, such a prime p may be too large to use machine arithmetic, thus limiting the effectiveness of our proposed Dixon resultant algorithm to solve many parametric polynomial systems. Also, one has to deal with the unlucky evaluation points problem posed by using the evaluation points $(2^j, 3^j, \ldots, p_n^j)$ (See Examples 4.1 and 4.2).

To address these problems, we develop a new sparse multivariate rational function interpolation method which avoids unlucky evaluation points with high probability and needs smaller primes. Our new sparse rational function interpolation method modifies the Cuyt and Lee's algorithm and the Ben-Or/Tiwari algorithm to use a Kronecker substitution to reduce the size of the prime and we evaluate at powers of a generator of $\mathbb{Z}_p^*$ instead of powers of primes $(2^j, 3^j, \ldots, p_n^j)$ to avoid unlucky evaluation points with high probability. Thus, instead of interpolating the R_j's directly, we interpolate

$$K_r(R_j) = x_1^{d_{T_j}} + \sum_{k=0}^{T_j-1} \frac{f_{jk}(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}{g_{jk}(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})} x_1^{d_{j_k}} \in \mathbb{Q}(y)[x_1]$$

where the map $K_r : \mathbb{Q}(y_1, \ldots, y_m)[x_1] \to \mathbb{Q}(y)[x_1]$ is a Kronecker substitution and each $r_i > \max_{j=1}^{l} \left(\max_{k=0}^{T_j-1} (\deg(f_{jk}, y_i), \deg(g_{jk}, y_i)) \right)$ for $1 \leq i \leq m$. Inverting the Kronecker map yields the R_j. With this modification, we remark that the number of terms and the number of black box probes needed to interpolate R_j does not change.

Identifying the extraneous factors of the monic square-free factors

We recall from Example 1.10 that

$$A_3 = \left(y_1^2 + 2y_1y_4\right) x_1^2 + y_1^2 - 4y_1y_3 + 2y_1y_4 + 4y_3^2 - 4y_3y_4$$
$$A_4 = \left(y_1^2 - 2y_1y_4\right) x_1^2 + y_1^2 - 4y_1y_3 - 2y_1y_4 + 4y_3^2 + 4y_3y_4$$

where A_3 and A_4 are factors of the Dixon resultant R in x_1 for the robot arms system. The monic square-free factor $R_3 = \text{monic}(A_3 A_4, x_1)$ of R is an extraneous factor.

A natural question that follows is how do we identify the extraneous factors from the interpolated monic square-free factors since the Dixon resultant R is a polynomial in the elimination ideal $J = I \cap \mathbb{Q}(Y)[x_1]$. Surely, some of the extraneous factors present in the Dixon resultant R may have disappeared when the polynomial content was removed. Regardless, the presence of any extraneous factor in the monic square-free factors must be identified. We have designed a new probabilistic algorithm based on Gröbner basis to identify the extraneous factor in the monic square-free factors. This algorithm involves specializing the set of parameters Y at random values selected from $\mathbb{Z}_p$, where p is a random 62 bit prime.

For an evaluation point $\beta \in \mathbb{Z}_p^m$ selected uniformly at random for the parameters, we create a new polynomial system $G = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_n\}$ by computing $\hat{g}_i = \hat{f}_i(X, \beta) \in \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$ satisfying $\deg(\hat{g}_i, x_1) = \deg(\hat{f}_i, x_1)$ for all i. Then we compute the monic univariate polynomial in $J^* = I^* \cap \mathbb{Z}_p[x_1]$ where $I^* = \langle \hat{g}_1, \hat{g}_2, \ldots, \hat{g}_n \rangle$ using a Gröbner basis, and we use it to identify the extraneous factors in our monic-square factors R_j with high probability (See Subsection 5.3.2 for more details).

Implementation and Experimental Comparison

We have implemented our Dixon resultant algorithm in Maple with several parts coded in C to improve its overall efficiency. Our hybrid Maple + C code (with instructions and test files) can be downloaded freely for use from the web at:

http://www.cecm.sfu.ca/personal/monaganm/code/DixonRes/.

We compare our new Dixon resultant algorithm with a hybrid Maple + C implementation of Zippel's interpolation algorithm [Zippel, 1979], a Maple implementation of Gentleman and Johnson minor expansion algorithm [Gentleman and Johnson, 1974] and a Maple implementation of Lewis' Dixon-EDF algorithm for computing R. Experimental results (See Section 5.4.3 for benchmarks) show that our new Dixon resultant algorithm outperforms Zippel's algorithm for interpolating the Dixon resultant R and is able to solve many parametric polynomial systems that other methods cannot solve.

Failure Probability Analysis and Complexity Analysis

Our Dixon resultant algorithm is probabilistic of Monte Carlo type. That is, our Dixon resultant algorithm may fail, and, even with failure detection checks in place, the returned monic square-free factors R_j might be incorrect. Thus, our next contribution is the failure probability analysis of our Dixon resultant algorithm and the complexity analysis of our Dixon resultant algorithm in terms of the number of black box probes required to interpolate the R_j's (See Chapter 6). We identify several causes of failure in our Dixon resultant

algorithm, and we obtain new failure probability bounds which depend on the input parametric polynomial systems. New bounds for the Dixon resultant R and bounds for its monic square-free factors R_j are also obtained which are potentially useful in other applications.

1.2.2 Solving $Ax = b$ using sparse rational function interpolation

Consider the parametric linear system $Ax = b$ where $A \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^{n \times n}$ is the coefficient matrix of full rank n and $b \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^n$ is the right-hand side column vector such that the number of terms in the entries of A and b denoted by $\#A_{ij}, \#b_i \leq t$ and $\deg(A_{ij}), \deg(b_i) \leq d$. Elementary linear algebra tells us that the solution vector x is unique since the coefficient matrix A is of full rank n. In this thesis, we aim to interpolate the solution vector of rational functions

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T = \begin{bmatrix} \frac{f_1}{g_1} & \frac{f_2}{g_2} & \cdots & \frac{f_n}{g_n} \end{bmatrix}^T \tag{1.1}$$

such that for $f_k, g_k \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$, $g_k \neq 0$, $g_k | \det(A)$ and $\gcd(f_k, g_k) = 1$ for $1 \leq k \leq n$.

Using Cramer's rule, the solutions of $Ax = b$ are given by

$$x_i = \frac{\det(A^i)}{\det(A)} \in \mathbb{Z}(y_1, \ldots, y_m) \tag{1.2}$$

where A^i is the matrix obtained by replacing the i-th column of A with b, and $\det(A)$ is a polynomial in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$. Let

$$\tilde{x}_i := \det(A^i) = x_i \det(A) \in \mathbb{Z}[y_1, y_2, \ldots, y_m].$$

Maple and other computer algebra systems such as Magma have an implementation of the Bareiss/Edmonds one-step fraction free Gaussian elimination algorithm [Bareiss, 1968, Edmonds, 1967] which triangularizes an augmented matrix $B = [A|b]$ to obtain $\det(A)$ as a polynomial in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$, and then solves for the polynomials $\tilde{x}_i$ via back substitution using Lipson's fraction free back formula [Lipson, 1969]. Ignoring pivoting, the following pseudocode (Algorithm 1) of the Bareiss/Edmonds algorithm and Lipson's fraction free back substitution formula solves $Ax = b$:

Algorithm 1: BareissPseudocode

Input: The coefficient matrix A and the column vector b with $n \geq 1$ and $m \geq 1$.

Output: The unique vector $x \in \mathbb{Z}(y_1, y_2, \ldots, y_m)^n$ such that $Ax = b$.

$B := [A|b\,]$; $\quad B_{0,0} := 1$;
// fraction free triangularization begins
for $k = 1, 2, \ldots, n-1$ **do**
  for $i = k+1, k+2, \ldots, n$ **do**
    for $j = k+1, k+2, \ldots, n+1$ **do**

$$B_{i,j} := \frac{B_{k,k}B_{i,j} - B_{i,k}B_{k,j}}{B_{k-1,k-1}} \tag{1.3}$$

    end do
    $B_{i,k} := 0$;
  end do
end do
// fraction free back substitution begins
$\tilde{x}_n := B[n, n+1]$;
for $i = n-1, n-2, \ldots, 2, 1$ **do**
  $N_i := B_{i,n+1}B_{n,n} - \sum_{j=i+1}^{n} B_{i,j}\tilde{x}_j$;
  $D_i := B_{i,i}$;

$$\tilde{x}_i := \frac{N_i}{D_i}; \tag{1.4}$$

end do
// simplification begins
for $i = 1, 2, \ldots, n$ **do**
  $h_i = \gcd(\tilde{x}_i, B_{n,n}) \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$;
  $f_i := \frac{\tilde{x}_i}{h_i}$;
  $g_i := \frac{B_{n,n}}{h_i}$;
  $x_i := f_i / g_i$;
end do

Note that the divisions by $B_{k,k}$ and by D_i in Algorithm 1 are exact in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$ and $B_{k,k}$ is the determinant of the principal $k \times k$ sub-matrix of A. However, there is an expression swell. At the last major step of triangularizing B when $k = n-1$ where it computes

$$B_{n,n} = \frac{B_{n-1,n-1}B_{n,n} - B_{n,n-1}B_{n-1,n}}{B_{n-2,n-2}} = \det(A), \tag{1.5}$$

the numerator polynomial in (1.5) is the product of determinants $B_{n,n}$ and $B_{n-2,n-2}$ which are polynomials in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$. If the original entries $B_{i,j}$ from B are sparse polynomials in many parameters then the numerator polynomial in (1.5) may be 100 times or more larger than $\det(A)$. The same situation also holds for the polynomials $\tilde{x}_i$.

One approach to avoid this expression swell tried by Monagan and Vrbik in [Vrbik and Monagan, 2009]. They compute the quotients of (1.3) and (1.4) directly using lazy polynomial arithmetic without constructing the numerator polynomial in (1.5) in expanded form.

Another approach is to interpolate the polynomials $\tilde{x}_i$ and $\det(A)$ directly from points using sparse polynomial interpolation algorithms [Roche, 2018, Giesbrecht et al., 2006, 2009, Ben-Or and Tiwari, 1988, Zippel, 1990], and Chinese remaindering when needed. This approach is described briefly as follows.

- Pick an evaluation point $\alpha \in \mathbb{Z}_p^m$ and solve

$$A(\alpha)x(\alpha) = b(\alpha) \bmod p$$

  for $\tilde{x}(\alpha)$ using Gaussian elimination over $\mathbb{Z}_p$ and also compute $\det(A(\alpha))$ at the same time.

- Provided $\det(A(\alpha)) \neq 0$, then

$$\tilde{x}_i(\alpha) = x_i(\alpha) \times \det(A(\alpha)).$$

  Thus, we have images of $\tilde{x}_i$ and $\det(A)$ so we can interpolate them.

To compute the solution vector x in the simplest terms, we compute

$$h_i = \gcd(\tilde{x}_i, \det(A))$$

for $1 \leq i \leq n$ and cancel them from $\dfrac{\tilde{x}_i}{\det(A)}$ to simplify the solutions. However, in practice there may be a large cancellation in $\dfrac{\tilde{x}_i}{\det(A)}$. That is, h_i may be a large factor so that the final solution

$$x_i = \frac{\tilde{x}_i/h_i}{\det(A)/h_i}$$

may be small. Our new algorithm will interpolate x_i directly, thus avoiding any gcd computations which may be expensive. To illustrate the gain realized by our new algorithm for solving $Ax = b$, we give the following real example.

Example 1.12. Consider the following linear system of 21 equations in variables $x_1, x_2, \cdots, x_{21}$ and parameters $y_1, y_2, \ldots, y_5$:

$$\begin{cases} x_7 + x_{12} = & 1 \\ x_8 + x_{13} = & 1 \\ x_{21} + x_6 + x_{11} = & 1 \\ x_1y_1 + x_1 - x_2 = & 0 \\ x_3y_2 + x_3 - x_4 = & 0 \\ x_{11}y_3 + x_{11} - x_{12} = & 0 \\ x_{16}y_5 - x_{17}y_5 - x_{17} = & 0 \\ -x_{20}y_3 + x_{21}y_3 + x_{21} = & 0 \\ -x_5y_3 + x_6y_3 + x_6 - x_7 = & 0 \\ x_8y_4 + x_9y_3 + x_9 = & 0 \\ -x_{10}y_2 + x_{18}y_2 + x_{18} - x_{19} = & 0 \\ y_4(x_{14} - x_{13}) + x_{14} - x_{15} = & 0 \\ 2x_3(y_2^2 - 1) + 4x_4 - 2x_5 = & 0 \\ 2y_1^2(x_1 - 1) - 2x_{10} + 4x_2 = & 0 \\ 2y_3^2(x_{19} - 2x_{20} + x_{21}) - 2x_{21} = & 0 \\ 2y_4^2(x_7 - 2x_8 + x_9) - 2x_9 = & 0 \\ 2x_{11}(y_3^2 - 1) + 4x_{12} - 2x_{13} = & 0 \\ 2y_4^2(x_{12} - 2x_{13} + x_{14}) - 2x_{14} + 4x_{15} - 2x_{16} = & 0 \\ 2y_3^2(x_4 - 2x_5 + x_6) - 2x_6 + 4x_7 - 2x_8 = & 0 \\ 2y_5^2(x_{15} - 2x_{16} + x_{17}) - 2x_{17} = & 0 \\ -2y_2^2(2x_{10} + x_{18} + x_2) - 2x_{18} + 4x_{19} - 2x_{20} = & 0 \end{cases}$$

where the solution of the above system defines a general cubic Beta-Spline in the study of modelling curves in Computer Graphics.

Using the Bareiss/Edmonds/Lipson algorithm on page 16, we find that

- $\#B[n, n] = \#\det(A) = 1033$,
- $\#B[n-2, n-2] = 672$ and $\#B[n, n] \times B[n-2, n-2] = 14348$, so an expression swell factor of $14348/1033 = 14$.

Furthermore, we obtain the number of terms in the numerator and denominator polynomials of $\tilde{x}_i = N_i/D_i$ and $x_i = f_i/g_i$, and the expression swell factor labelled **swell** for computing $\tilde{x}_i$ in Table 1.5.

	1	2	3	4	5	6	7	8	9	10	11
$\#N_i$	586	1,172	1,197	1,827	2,142	1,666	2,072	1,320	1,320	2,650	2,543
$\#D_i$	2	3	6	9	9	9	9	9	18	18	27
$\#\tilde{x}_i$	293	586	504	693	882	686	840	536	424	879	638
swell	2	2	3	3	3	3	3	3	3	3	4
$\#\ f_i$	1	2	4	4	4	19	16	8	8	8	2
$\#\ g_i$	5	3	10	7	4	22	16	16	26	12	3

	12	13	14	15	16	17	18	19	20	21
$\#N_i$	3,490	3,971	5,675	7,410	4,940	7,072	11,793	12,802	11,211	9,620
$\#D_i$	36	36	117	153	153	432	672	672	672	672
$\#\tilde{x}_i$	834	1,033	871	1044	696	348	690	836	693	528
swell	4	4	7	7	7	20	17	15	16	18
$\#\ f_i$	1	1	1	1	1	2	14	4	1	1
$\#\ g_i$	3	3	5	5	3	3	23	7	4	7

Table 1.5: Number of terms in the numerator and denominator polynomials of $\tilde{x}_i = N_i/D_i$ and $x_i = f_i/g_i$, and expression swell factor for computing $\tilde{x}_i$

The reader should note that the largest polynomial to be interpolated in x_i by our new algorithm for solving $Ax = b$ is g_9 in $x_9 = \frac{f_9}{g_9}$ where $\#g_9 = 26$.

The Gentleman & Johnson minor expansion algorithm [Gentleman and Johnson, 1974] can also be used to compute the solutions x_i by computing $n+1$ determinants, namely, the numerators $\det(A^i)$ for $1 \leq i \leq n$ (A^i is as defined in (1.2)), and the denominator $\det(A)$ only once. But then, we still have to compute $g_i = \gcd(\det(A^i), \det(A))$ to simplify the solutions x_i which is not cheap.

In this thesis, we avoid any gcd computations by interpolating the simplified solutions $x_i = f_i/g_i$ directly using our new sparse rational function interpolation (See Chapter 4). Similar to our approach for computing Dixon resultants, we use a black box representation to denote any given parametric linear system. That is, a black box **BB** representing $Ax = b$ denoted by $\mathbf{BB} : \left(\mathbb{Z}_p^m, p\right) \rightarrow \mathbb{Z}_p^n$ is a computer program that takes an evaluation point $\alpha \in \mathbb{Z}_p^m$ and a prime p as two inputs and outputs $x(\alpha) = A^{-1}(\alpha)b(\alpha) \in \mathbb{Z}_p^n$. The implication of the black box representation of $Ax = b$ is that important properties of x such as the number of terms in the polynomials f_k and g_k, and their variable degrees are unknown so we have to find them by interpolation.

Our new algorithm probes a given black box **BB** and uses our new sparse multivariate rational function interpolation to interpolate the rational function entries in x modulo primes, and then uses Chinese remaindering and rational number reconstruction to recover their rational coefficients. We have made a hybrid Maple + C implementation of our new algorithm for solving $Ax = b$ which can be downloaded for use from the web at:

http://www.cecm.sfu.ca/personal/monaganm/code/ParamLinSolve/.

We present timing results comparing our new algorithm for solving parametric linear systems with a Maple implementation of the Bareiss/Edmonds/Lipson fraction free Gaussian elimination algorithm with three other algorithms for solving $Ax = b$. Finally, we give a detailed failure probability analysis of our new algorithm for solving $Ax = b$ and the complexity analysis of our algorithm in terms of the number of black box probes required to interpolate x.

1.3 Some Elimination Techniques

In this section, we review some classical elimination techniques, namely, the Sylvester and Macaulay resultants, and Gröbner bases. In Chapter 2, we will provide a detailed description of the Dixon resultant.

1.3.1 Sylvester Resultant

The Sylvester resultant is commonly used to determine the existence of a common root of two univariate polynomials. It has been studied extensively, and many computer algebra systems have an efficient implementation of the algorithm. Maple has a dense modular algorithm [Brown, 1971] that was implemented by Wittkopf. Before presenting the derivation of the Sylvester resultant, we first give the following important lemma.

Lemma 1.13. [von zur Gathen and Gerhard, 2013, Lemma 6.13] Let F be a field and let $f, g \in F[x]$ be non-zero polynomials. Then $\gcd(f, g) \neq 1$ if and only if there exists non-zero polynomials $s, t \in F[x]$ such that $sf + tg = 0$ with $\deg(s) < \deg(g)$ and $\deg(t) < \deg(f)$.

Consider the problem of determining if two polynomials $f(x), g(x) \in F[x]$ of positive degrees d_f and d_g in variable x have a common root. Let

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_{d_f} x^{d_f}$$

and

$$g(x) = b_0 + b_1 x + b_2 x^2 + \cdots b_{d_g} x^{d_g}$$

where the coefficients $a_i, b_j \in F$, for $0 \leq i \leq d_f$ and $0 \leq j \leq d_g$. Suppose

$$t = t_0 + t_1 x + t_2 x^2 + \cdots t_{d_f - 1} x^{d_f - 1}$$

and

$$s = s_0 + s_1 x + s_2 x^2 + \cdots s_{d_g - 1} x^{d_g - 1}$$

where the coefficients $t_i, s_j \in F$, $\forall i, j$. Using Lemma 1.13, we can write

$$sf + tg = 0,$$

which is equivalent to creating the following $(d_g + d_f)$ homogeneous linear system in terms of the s_i's and the t_i's:

$$\begin{aligned}
s_0a_0 + t_0b_0 &= 0 \\
s_1a_0 + s_0a_1 + t_1b_0 + t_0b_1 &= 0 \\
s_2a_0 + s_1a_1 + s_0a_2 + t_2b_0 + t_1b_1 + t_0b_2 &= 0 \\
&\vdots \\
s_{d_g-1}a_{d_f-1} + s_{d_g-2}a_{d_f} + t_{d_f-1}b_{d_g-1} + t_{d_f-2}b_{d_g} &= 0 \\
s_{d_g-1}a_{d_f} + t_{d_f-1}b_{d_g} &= 0
\end{aligned}$$

The above system can be written in matrix form as

$$S(f,g)^T V^T = 0$$

where $S(f,g)$ is the following $(d_g + d_f) \times (d_g + d_f)$ matrix:

$$S(f,g) = \begin{pmatrix}
a_{d_f} & a_{d_f-1} & \dots & a_0 & & & & \\
 & a_{d_f} & a_{d_f-1} & \cdots & a_0 & & & \\
 & & a_{d_f} & a_{d_f-1} & \cdots & a_0 & & \\
 & & & \ddots & \ddots & \ddots & \ddots & \\
 & & & & a_{d_f} & a_{d_f-1} & \cdots & a_0 \\
b_{d_g} & b_{d_g-1} & \dots & b_0 & & & & \\
 & b_{d_g} & b_{d_g-1} & \dots & b_0 & & & \\
 & & b_{d_g} & b_{d_g-1} & \dots & b_0 & & \\
 & & & \ddots & \ddots & \ddots & \ddots & \\
 & & & & b_{d_g} & b_{d_g-1} & \dots & b_0
\end{pmatrix}$$

and

$$V = \left[s_{d_g-1}, s_{d_g-2}, \cdots s_0, t_{d_f-1}, t_{d_f-2}, \cdots t_0 \right].$$

The matrix $S(f,g)$ is known as the *Sylvester matrix* and its determinant denoted by $\text{res}_x(f,g)$ is the Sylvester resultant of f and g with respect to x. The Sylvester matrix $S(f,g)$ has d_g rows of a_i's, d_f rows of b_i's, and the blank spaces are filled with zeros. We know from linear algebra that the system $S(f,g)^T V^T = 0$ has a non-trivial solution if and only if $\det(S(f,g)) = 0$.

Example 1.14. Let $f(x) = x^2 + 5x + 6$ and $g(x) = -x^2 - 4x - 3$. The matrix

$$S(f,g) = \begin{pmatrix} 1 & 5 & 6 & 0 \\ 0 & 1 & 5 & 6 \\ -1 & -4 & -3 & 0 \\ 0 & -1 & -4 & -3 \end{pmatrix}$$

is the Sylvester matrix of f and g.

The following result describes how the Sylvester resultant can be used under certain conditions to solve systems of polynomial equations.

Theorem 1.15. [Geddes et al., 1992, Theorem 9.5] Let k be an algebraically closed field and let

$$\hat{f} = \sum_{i=0}^{\deg(\hat{f})} \hat{f}_i(x_2, x_3, \ldots, x_r)x_1^i, \quad \hat{g} = \sum_{i=0}^{\deg(\hat{g})} \hat{g}_i(x_2, x_3, \ldots, x_r)x_1^i$$

be elements of $k[x_1, x_2, \ldots, x_r]$ of positive degrees in x_1. If $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_r)$ is a common root of $\hat{f}$ and $\hat{g}$, then

$$\mathrm{res}_{x_1}(\hat{f}, \hat{g})(\alpha_2, \alpha_3, \ldots, \alpha_r) = 0.$$

Conversely, if $\mathrm{res}_{x_1}(\hat{f}, \hat{g})$ vanishes at $(\alpha_2, \alpha_3, \ldots, \alpha_r)$, then at least one of the following holds:

(a) $\hat{f}_{\deg(\hat{f})}(\alpha_2, \alpha_3, \ldots, \alpha_r) = \cdots = \hat{f}_0(\alpha_2, \alpha_3, \ldots, \alpha_r) = 0$,

(b) $\hat{g}_{\deg(\hat{g})}(\alpha_2, \alpha_3, \ldots, \alpha_r) = \cdots = \hat{g}_0(\alpha_2, \alpha_3, \ldots, \alpha_r) = 0$,

(c) $\hat{f}_{\deg(\hat{f})}(\alpha_2, \alpha_3, \ldots, \alpha_r) = \hat{g}_{\deg(\hat{g})}(\alpha_2, \alpha_3, \ldots, \alpha_r) = 0$,

(d) There exists $\alpha_1 \in k$ such that $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_r)$ is a common root of $\hat{f}$ and $\hat{g}$.

The above theorem provides us a way to use Sylvester resultants to extend partial solutions of systems of polynomial equations to full solutions under certain conditions. In particular, if conditions $(a), (b), (c)$ are not satisfied, then (d) assures us that the partial solutions can be extended by one more coordinate. Also, note that Theorem 1.15 can be applied inductively and modified to handle more than two polynomials.

The following example demonstrates how to use the Sylvester resultant to eliminate a variable from two bivariate polynomials.

Example 1.16. Eliminating x_2 from the two bivariate polynomials

$$\hat{f}_1 = x_2x_1^2 - x_1 - 1$$
$$\hat{f}_2 = x_2x_1^2 + x_2x_1 - 2,$$

we get that

$$\text{res}_{x_2}(\hat{f}_1, \hat{f}_2) = x_1^3 + x_1.$$

However, the resultant of $\hat{f}_1$ and $\hat{f}_2$ is x_1^2+1, so the factor x_1 of $\text{res}_{x_2}(\hat{f}_1, \hat{f}_2)$ is an extraneous factor.

The reader should note that the Sylvester resultant is not efficient when used to successively eliminate $n-1$ variables from a system of polynomial equations in n variables [Lewis, 1996]. For future use in Chapter 6, we state the following facts about the Sylvester resultant now.

Lemma 1.17. [von zur Gathen and Gerhard, 2013, Corollary 6.21] Let F be an integral domain and let $f, g \in F[x]$ be non-zero polynomials with $\deg(f) + \deg(g) \geq 1$. Then there exist non-zero polynomials $s, t \in F[x]$ such that $sf + tg = \text{res}_x(f, g)$ where $\deg(s) < \deg(g)$ and $\deg(t) < \deg(f)$.

Lemma 1.18. [Hu and Monagan, 2021, Lemma 4] Let F be an integral domain. Let f and g be polynomials in $F[y_1, y_2, \ldots, y_m]$ with $d_f = \deg(f, y_1) > 0$ and $d_g = \deg(g, y_1) > 0$. Let a_{d_f} and b_{d_g} be the leading coefficients of f and g with respect to the main variable y_1. Let $R_s = \text{res}_{y_1}(f, g)$ be the Sylvester resultant of f and g with respect to y_1. Let $\alpha \in F^{m-1}$ be an evaluation point. For any prime p, let ϕ_p be the modular mapping $\phi_p(f) = f \bmod p$. Then the following results hold:

(i) R_s is a polynomial in $F[y_2, \ldots, y_m]$,

(ii) $\deg(R_s) \leq \deg(f)\deg(g)$ (Bezout bound) and

(iii) $\deg(R_s, y_i) \leq d_g \deg(f, y_i) + d_f \deg(g, y_i)$ for $2 \leq i \leq m$.

If F is a field, and $a_{d_f}(\alpha) \neq 0$ and $b_{d_g}(\alpha) \neq 0$ then

(iv) $\text{res}_{y_1}(f(y_1, \alpha), g(y_1, \alpha)) = R_s(\alpha)$ and

(v) $\deg(\gcd(f(y_1, \alpha), g(y_1, \alpha)), y_1) > 0 \iff R_s(\alpha) = 0$.

If $F = \mathbb{Z}$, and $\phi_p(a_{d_f}) \neq 0$ and $\phi_p(b_{d_g}) \neq 0$ then

(vi) $\text{res}_{y_1}(\phi_p(f), \phi_p(g)) = \phi_p(R_s)$ and

(vii) $\deg(\gcd(\phi_p(f), \phi_p(g)), y_1) > 0 \iff \phi_p(R_s) = 0$.

1.3.2 Macaulay Resultant

The material in this section follows [Cox et al., 2005, Kapur and Yagati, 1992]. Given a system of homogeneous polynomial equations $\mathcal{F} = \{\hat{f}_1, \ldots, \hat{f}_n\}$ in n variables, we describe the classical formulation of the Macaulay resultant $\overline{R}_M$. Let $d_i = \deg(\hat{f}_i)$ be the total degree of the polynomial $\hat{f}_i$ for $1 \leq i \leq n$. Let

$$D = 1 + \sum_{i=1}^{n}(d_i - 1) = 1 - n + \sum_{i=1}^{n} d_i$$

where $n = |\mathcal{F}|$. The quantity D is often called the Macaulay degree.

Consider the monomial set of all terms of degree D in all the n variables $x_1, x_2, \ldots, x_n$ defined by

$$T = \left\{ x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \sum_{i=1}^{n} \alpha_i = D \right\}$$

where the cardinality of T is

$$|T| = \binom{D+n-1}{n-1}.$$

Note that $|T|$ counts the number of non-negative solutions to $\sum_{i=1}^{n} \alpha_i = D$. Let

$$\begin{aligned}
T_1 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \sum_{i=1}^{n} \alpha_i = D - d_1 \right\}, \\
T_2 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \sum_{i=1}^{n} \alpha_i = D - d_2 \text{ and } \alpha_1 < d_1 \right\}, \\
T_3 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \sum_{i=1}^{n} \alpha_i = D - d_3 \text{ and } \alpha_2 < d_2 \text{ and } \alpha_1 < d_1 \right\}, \\
&\vdots \\
T_n &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \sum_{i=1}^{n} = D - d_n \text{ and } \alpha_i < d_i \text{ for } 1 \leq i \leq n-1 \right\}.
\end{aligned}$$

Macaulay referred to the set T_i as the reduced set of terms with respect to $x_1, x_2, \ldots, x_i$, and he proved that

$$\sum_{i=1}^{n} |T_i| = |T|.$$

One can also view the terms of set T_i as terms of degree $D - d_i$ which are not divisible by $x_1^{d_1}$ or $x_2^{d_2}$ or $\cdots$ or $x_{i-1}^{d_{i-1}}$. Thus, a term is reduced with respect to $x_1, x_2, \ldots, x_i$ if it is not divisible by any of $x_1^{d_1}, x_2^{d_2}, \ldots, x_i^{d_i}$. Using the monomials from the sets T_i as multipliers against the old polynomial system $\mathcal{F}$, one can form a new square system of $|T|$ linear equations $\mathcal{G} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_{|T|}\}$ in T unknowns which are power products of D, namely:

$$
\begin{aligned}
\hat{g}_1 &= T_{1,1}\hat{f}_1 \\
\hat{g}_2 &= T_{1,2}\hat{f}_1 \\
\hat{g}_3 &= T_{1,3}\hat{f}_1 \\
&\vdots \\
\hat{g}_{|T_1|} &= T_{1,|T_1|}\hat{f}_1 \\
\hat{g}_{|T_1|+1} &= T_{2,1}\hat{f}_2 \\
\hat{g}_{|T_1|+2} &= T_{2,2}\hat{f}_2 \\
&\vdots \\
\hat{g}_{|T|} &= T_{n,|T_n|}\hat{f}_n
\end{aligned}
$$

The order in which the polynomials $\hat{f}_i$ are considered for selection against the multipliers from the monomial sets yields different (but equivalent) systems of linear equations $\mathcal{G}$.

A resultant matrix (Macaulay matrix) A with $|T|$ columns and $\sum_{i=1}^{n}|T_i|= |T|$ rows is constructed from the system of linear equations $\mathcal{G}$ such that the columns of A are labelled by the terms in T in some order. The first $|T_1|$ rows of A are labelled by elements of T_1, the next $|T_2|$ rows are labelled by T_2, and one continues in the same fashion way up to last row of A labelled by the last element in T_n. In a row labelled by the term $t \in T_i$, one must arrange the coefficients of tf_{i+1} with the coefficient of a term t^* in tf_{i+1} appearing under the column labelled by t^*. As a reminder, the total degree of the polynomials tf_{i+1} is D. We give the following example to illustrate how to construct a Macaulay resultant matrix.

Example 1.19. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_6\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_{15}][x_1, x_2, x_3]$ where

$$
\begin{aligned}
\hat{f}_1 &= y_1x_1^2 + y_2x_1x_2 + y_3x_1x_3 + y_4x_2^2 + y_5x_2x_3 + y_6x_3^2 \\
\hat{f}_2 &= y_7x_1^2 + y_8x_1x_2 + y_9x_1x_3 + y_{10}x_2^2 + y_{11}x_2x_3 + y_{12}x_3^2 \\
\hat{f}_3 &= y_{13}x_1 + y_{14}x_2 + y_{15}x_3
\end{aligned}
$$

Clearly, $(d_1, d_2, d_3) = (2, 2, 1)$, and the Macaulay degree

$$D = 1 - 3 + (2 + 2 + 1) = 3.$$

We have that

$$|T|= \binom{3+3-1}{3-1} = \binom{5}{2} = 10,$$

and

$$T = \left\{ x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_n} : \sum_{i=1}^{3} \alpha_i = 3 \right\} = \left\{ x_1^3, x_1^2 x_2, x_1^2 x_3, x_1 x_2^2, x_1 x_2 x_3, x_1 x_3^2, x_2^3, x_2^2 x_3, x_2 x_3^2, x_3^3 \right\}.$$

Furthermore,

$$\begin{aligned}
T_1 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3} : \sum_{i=1}^{3} \alpha_i = 1 \right\} = \{x_1, x_2, x_3\}, \\
T_2 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3} : \sum_{i=1}^{3} \alpha_i = 1 \text{ and } \alpha_1 < 2 \right\} = \{x_1, x_2, x_3\} \text{ and} \\
T_3 &= \left\{ x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3} : \sum_{i=1}^{3} \alpha_i = 2 \text{ and } \alpha_2 < 2 \text{ and } \alpha_1 < 2 \right\} = \left\{ x_3^2, x_1 x_3, x_1 x_2, x_2 x_3 \right\},
\end{aligned}$$

so

$$\sum_{i=1}^{3} |T_i| = |T| = 10.$$

Thus, the Macaulay matrix A is

$$\begin{array}{c} \\ x_1\hat{f}_1 \\ x_2\hat{f}_1 \\ x_3\hat{f}_1 \\ x_1\hat{f}_2 \\ x_2\hat{f}_2 \\ x_3\hat{f}_2 \\ x_1x_2\hat{f}_3 \\ x_1x_3\hat{f}_3 \\ x_2x_3\hat{f}_1 \\ x_3^2\hat{f}_3 \end{array} \begin{array}{c} \begin{array}{cccccccccc} x_1^3 & x_1^2x_2 & x_1^2x_3 & x_1x_2^2 & x_1x_2x_3 & x_1x_3^2 & x_2^3 & x_2^2x_3 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & 0 & 0 & 0 & 0 \\ 0 & y_1 & 0 & y_2 & y_3 & 0 & y_4 & y_5 & y_6 & 0 \\ 0 & 0 & y_1 & 0 & y_2 & y_3 & 0 & y_4 & y_5 & y_6 \\ y_7 & y_8 & y_9 & y_{10} & y_{11} & y_{12} & 0 & 0 & 0 & 0 \\ 0 & y_7 & 0 & y_8 & y_9 & 0 & y_{10} & y_{11} & y_{12} & 0 \\ 0 & 0 & y_7 & 0 & y_8 & y_9 & 0 & y_{10} & y_{11} & y_{12} \\ 0 & y_{13} & 0 & y_{14} & y_{15} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & y_{13} & 0 & y_{14} & y_{15} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & y_{13} & 0 & 0 & y_{14} & y_{15} & 0 \\ 0 & 0 & 0 & 0 & 0 & y_{13} & 0 & 0 & y_{14} & y_{15} \end{pmatrix} \end{array}.$$

Let $\det(A)$ denote the determinant of the Macaulay matrix A formed from the system of linear equations $\mathcal{G}$. The construction of A solely depends on the orderings of the homogeneous polynomial system $\mathcal{F}$. That is, a different resultant matrix is produced each time a different ordering is chosen for the polynomial system $\mathcal{F}$. Let $\det(A_\sigma)$ be the determinant of a matrix constructed using a permutation σ of the polynomials in $\mathcal{F}$. Let S_n be the symmetric group on n letters. Then $|S_n| = n!$, which means we have $n!$ possible determinants.

Macaulay discussed two ways to obtain the Macaulay resultant denoted by $\overline{R}_M$ from all the possible determinants $\det(A_\sigma)$. First, the Macaulay resultant $\overline{R}_M$ can be computed

by computing the gcd of all possible determinants $\det(A_\sigma)$. However, this is an intractable way of computing the resultant of a polynomial system. As a result, Macaulay derived a formula relating the Macaulay resultant $\overline{R}_M$ and $\det(A_\sigma)$, namely,

$$\overline{R}_M = \frac{\det(A_\sigma)}{\det(B_\sigma)}, \tag{1.6}$$

where B_σ is the determinant of a sub-matrix of A_σ obtained by deleting all columns labelled by terms not divisible by any $n-1$ elements of $\{x_1^{d_1}, x_2^{d_2}, \ldots, x_n^{d_n}\}$, and deleting the rows which contain at least one non-zero entry a_i, the coefficients of $x_i^{d_i}$ in $\hat{f}_i$, for $1 \leq i \leq n$ in the deleted columns. We remark that computing $\overline{R}_M$ using (1.6) could result in expression swell because

$$\det(A_\sigma) = \overline{R}_M \det(B_\sigma)$$

is much bigger than $\overline{R}_M$. This is another reason why we prefer Dixon resultants.

Example 1.20 (Continuation of Example 1.19)**.** Recall $(d_1, d_2, d_3) = (2, 2, 1)$ and

$$T = \left\{x_1^3,\, x_1^2x_2,\, x_1^2x_3,\, x_1x_2^2,\, x_1x_2x_3,\, x_1x_3^2,\, x_2^3\,, x_2^2x_3,\, x_2x_3^2,\, x_3^3\right\}.$$

To produce a sub-matrix B_σ, we first need to select which columns to delete from matrix $A_\sigma = A$ which was constructed in Example 1.19. We do this by checking the terms in T that are not divisible by any 2 of $\{x_1^2, x_2^2, x_3\}$. By inspection, one can easily see that the elements of the set $\{x_1^2x_3, x_2^2x_3\}$ are the only terms that are divisible by two of the elements from $\{x_1^2, x_2^2, x_3\}$. Thus, only columns 3 and 8 would remain after deletion.

To delete the rows, recall that the coefficients of $\{x_1^2, x_2^2, x_3\}$ in the original polynomial system $\mathcal{F}$ are $\{y_1, y_{10}, y_{15}\}$. Thus,

$$B_\sigma = \begin{array}{c} \\ x_3\hat{f}_1 \\ x_2\hat{f}_2 \end{array} \begin{array}{c} \begin{array}{cc} x_1^2x_3 & x_2^2x_3 \end{array} \\ \begin{pmatrix} y_1 & y_4 \\ y_7 & y_{10} \end{pmatrix} \end{array}.$$

We found out that $\#\overline{R}_M = 234, \#\det(A_\sigma) = 432$ and $\#\det(B_\sigma) = 2$, so an expression swell of about a factor of 2.

Example 1.21 (Heron 3d system)**.** Consider the parametric system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots \hat{f}_6\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_6][x_1, x_2, \cdots, x_6]$ listed in Appendix section A.4. Using Macaulay resultant to eliminate variables $x_2, x_3, \ldots, x_6$, we found out that A_σ is a 792×792 matrix and its sub-matrix B_σ is a 600×600 matrix.

Our attempt to quantify the expression swell factor was unsuccessful because Maple ran out of memory when computing $\det(A_\sigma)$ and also $\det(B_\sigma)$. Using Dixon resultants, we were able to determine that the number of terms in the Dixon resultant of $\mathcal{F}$ is only 23.

A drawback of the Macaulay resultant formulation is that it is possible that the matrices A_σ and B_σ are singular, thus providing no information about the desired Macaulay resultant. Another drawback of the Macaulay resultant formulation is that the sizes of the resultant matrices are often large leading to an extraneous factor of high degree. Details on how to address these hurdles are provided in [Kapur and Yagati, 1992].

1.3.3 Gröbner Bases

The first Gröbner basis algorithm [Buchberger, 2006] was given by Bruno Buchberger in 1965. The algorithm was named after his PhD advisor Wolfgang Gröbner. Since then, many variations of the algorithm have been developed and implemented in computer algebra systems such as Maple and Magma. The theory of Gröbner bases is much more extensive and sophisticated than what we can delve into here. In this thesis, our focus is to only review the parts of Gröbner bases that are needed to perform variable elimination from polynomial systems. We note that the material in this section follows [Cox et al., 2015].

Monomial Orderings

Definition 1.22. Let k be a field and let M be a set of monomials in $k[x_1, x_2, \ldots, x_n]$, i.e.,

$$M = \{x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots, x_n^{\alpha_n} : \text{for all}\, \alpha \in \mathbb{Z}_{\geq 0}^n\}.$$

The vector α is called an exponent vector.

Definition 1.23. Let M be a set of monomials in $k[x_1, x_2, \ldots, x_n]$. An order relation $<$ on M is a total ordering if $\forall x^\alpha, x^\beta, x^\gamma$

(i) Either $x^\alpha < x^\beta$ or $x^\alpha > x^\beta$ or $x^\alpha = x^\beta$ and

(ii) $x^\alpha > x^\beta$ and $x^\beta > x^\gamma \implies x^\alpha > x^\gamma$.

Definition 1.24. A monomial ordering on $k[x_1, \ldots, x_n]$ is a relation $>$ on $\mathbb{Z}_{\geq 0}^n$ satisfying

(i) $>$ is a total ordering,

(ii) $\forall \alpha, \beta, \gamma \in \mathbb{Z}_{\geq 0}^n$, we have $\alpha > \beta \implies \gamma + \alpha > \gamma + \beta$ and

(iii) Every non-empty subset $S \subset \mathbb{Z}_{\geq 0}^n$ has a least element under $>$.

Definition 1.25. Let $\alpha, \beta \in \mathbb{Z}_{\geq 0}^n$ with $\alpha \neq \beta$. Then $\alpha > \beta$ in lexicographical order written as $\alpha >_{\text{lex}} \beta$, if the left-most non-zero element in $\alpha - \beta$ is positive. That is, the monomials are compared first by their degree in the first variable, with ties broken by degree in the second variable and so on.

Definition 1.26. Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$ with $\alpha \neq \beta$. Let $\deg(\alpha) = \sum_{i=1}^n \alpha_i = \deg(x^\alpha)$. Then $\alpha > \beta$ in graded lexicographical order written as $\alpha >_{\text{grlex}} \beta$, if $\deg(\alpha) > \deg(\beta)$ or $\deg(\alpha) = \deg(\beta)$ and $\alpha >_{\text{lex}} \beta$. That is, monomials are compared first by their total degree, with ties broken by lexicographic order.

Definition 1.27. For graded reverse lexicographic order with $x_1 > x_2 > \cdots > x_n$, monomials are compared first by their total degree, with ties broken by reverse lexicographic order, that is, by the smallest degree in $x_n, x_{n-1}, \cdots, x_1$. This order is commonly used because it typically provides for the fastest Gröbner basis computations.

Definition 1.28. The leading term of f denoted by $\text{LT}(f)$ is the term whose monomial is greatest with respect to the fixed monomial order. The coefficient and monomial of this term denoted by $\text{LC}(f)$ and $\text{LM}(f)$ are called the leading coefficient and the leading monomial respectively.

Example 1.29. Consider the polynomial $f = x_1x_2^2x_3 + x_1x_3^4$. The $\text{LM}(f) = x_1x_2^2x_3$ with respect to lexicographic order with $x_1 > x_2 > x_3$ and the $\text{LM}(f) = x_1x_3^4$ with respect to graded lexicographic order with $x_1 > x_2 > x_3$.

Next, we would like to discuss testing for membership in an ideal with respect to a monomial order using the division algorithm.

Division algorithm in $k[x_1, x_2, \ldots, x_n]$

Given $f_1, f_2, \ldots, f_s \in k[x_1, x_2, \ldots, x_n] \setminus \{0\}$, to compute $f \div \{f_1, f_2, \ldots, f_s\}$ with respect to a monomial ordering means we seek quotients $q_1, q_2, \ldots, q_s \in k[x_1, x_2, \ldots, x_n]$ and a remainder $r \in k[x_1, x_2, \ldots, x_n]$, which implies that f is expressible as $f = \sum_{i=1}^s q_i f_i + r$.

Algorithm 2: Division algorithm

Input: A monomial ordering on $\mathbb{Z}^n_{\geq 0}$, a polynomial f and non-zero divisors $f_1, \ldots, f_s \in k[x_1, x_2, \ldots, x_n] \setminus \{0\}$.

Output: a polynomial r where no term of r is divisible by an $\text{LT}(f_i)$, and polynomials $q_1, q_2, \ldots, q_s$ such that $f = \sum_{i=1}^s q_i f_i + r$.

$(q_1, q_2, \ldots, q_s) \leftarrow (0, 0, \ldots, 0)$; $(r, p) \leftarrow (0, f)$
while $p \neq 0$ **do**
    Select the first f_i such that $\text{LT}(f_i) | \text{LT}(p)$.
    if no such i **exists then**
        $(r, p) \leftarrow (r + \text{LT}(p), p - \text{LT}(p))$
    else $t \leftarrow \dfrac{\text{LT}(p)}{\text{LT}(f_i)}$
        $(q_i, p) \leftarrow (q_i + t, p - tf_i)$
    end if
end while
return $(q_1, q_2, \ldots, q_s, r)$

We remark that the output and the number of steps of the division algorithm depends on the order of the f_i's. We give the following example to illustrate this.

Example 1.30. Suppose $f_1 = x_1x_2 + 1, f_2 = x_2 + 1$ and $f = x_1x_2^2 - x_1$. Computing $f \div \{f_1, f_2\}$ with respect to $<_{lex}$ with $x_1 > x_2$ yields $q_1 = x_2$, $q_2 = -1$, and a remainder $r = -x_1 + 1$. But computing $f \div \{f_2, f_1\}$ with respect to $<_{lex}$ with $x_2 > x_1$ yields $q_1 = x_1x_2 - x_1$, $q_2 = 0$, and a remainder $r = 0$, which implies that $f \in \langle f_1, f_2 \rangle$.

Now consider the following example.

Example 1.31. Let $\{f_1, f_2\} = \{x_1^2 + 1, x_1x_2 + 1\}$. Let $I = \langle f_1, f_2 \rangle$. Notice that

$$g = x_2 - x_1 = x_2f_1 - x_1f_2 \Longrightarrow g \in I.$$

However, one cannot use the division algorithm in the above example with respect to any monomial ordering to confirm that g is indeed an element of I. This problem will be remedied by Gröbner bases.

Definition 1.32. Let $I \subseteq k[x_1, x_2, \ldots, x_n] \setminus \{0\}$ be an ideal and let $<$ be a monomial ordering. A set G is said to be a Gröbner basis for I with respect to $<$ if for every $f \in I$, $\mathrm{LT}(f)$ is divisible by $\mathrm{LT}(g)$ for some $g \in G$.

Theorem 1.33. [Cox et al., 2015, Corollary 6, page 78] Fix a monomial order. Then every ideal $I \subseteq k[x_1, x_2, \ldots, x_n]$ has a Gröbner basis. Furthermore, any Gröbner basis for an ideal I is a basis of I.

Example 1.34 (Example 1.31 revisited)**.** Let $\{f_1, f_2\} = \{x_1^2+1, x_1x_2+1\}$. Let $I = \langle f_1, f_2 \rangle$. Recall

$$g = x_2 - x_1 = x_2f_1 - x_1f_2 \Longrightarrow g \in I.$$

Thus $\{f_1, f_2, g\}$ is a basis for I. The set $G = \{f_1, g\}$ is also a basis since $f_2 = x_1g + f_1$. Therefore, $I = \langle f_1, g \rangle$. Furthermore, notice that $\langle \mathrm{LT}(I) \rangle = \langle x_2, x_1^2 \rangle$ with respect to lexicographical order $x_2 > x_1$. Since $x_1 \notin \langle \mathrm{LT}(I) \rangle$, it follows that G is a Gröbner basis for I.

Proposition 1.35. [Cox et al., 2015, Proposition 1 and Corollary 2, page 83] Let $G = \{g_1, g_2, \ldots, g_t\}$ be a Gröbner basis for an ideal $I \subset k[x_1, x_2, \ldots, x_n]$ with respect to the monomial ordering $<$. Let $f \in k[x_1, x_2, \ldots, x_n]$. Then the remainder r of $f \div G$ is unique and satisfies

(i) $r = 0$ or no term in r is divisible by $\mathrm{LT}(g_i)$.

(ii) There exists $g \in I$ such that $f = g + r$.

(iii) $f \in I \iff r = 0$.

Notice that condition (iii) in the above proposition solves the ideal membership problem.

Computing a Gröbner basis

Definition 1.36. Let $f, g \in k[x_1, x_2, \ldots, x_n] \setminus \{0\}, \mathrm{LM}(f) = x^\alpha, \mathrm{LM}(g) = x^\beta$. Let $x^\gamma = \mathrm{LCM}(x^\alpha, x^\beta)$. The S-polynomial of f and g denoted by $S(f, g)$ is defined as

$$S(f, g) = \frac{x^\gamma}{\mathrm{LT}(f)} f - \frac{x^\gamma}{\mathrm{LT}(g)} g.$$

Definition 1.37. Let $f, g \in k[x_1, x_2, \ldots, x_n]$. We define $f \bmod g$ to be the remainder of the division $f \div g$.

Using $S-$polynomials, we now state Buchberger's criterion for determining when a basis of an ideal is a Gröbner basis.

Theorem 1.38 (Buchberger's $S-$polynomial Criterion)**.** [Cox et al., 2015, Theorem 6, page 86] Let I be an ideal with the generating set $G = \{g_1, g_2, \ldots, g_n\}$. Then G is a Gröbner basis for an ideal $I = \langle g_1, g_2, \ldots, g_n \rangle$ with respect to a monomial ordering $<$ if and only if

$$S(g_i, g_j) \bmod G = 0 \text{ for all } i \neq j.$$

Example 1.39 (Example 1.34 revisited)**.** Let

$$I = \left\langle x_1^2 + 1, x_1 x_2 + 1 \right\rangle = \left\langle x_1^2 + 1, x_2 - x_1 \right\rangle.$$

Let $G = \{F_1, F_2\}$ where $F_1 = x_1^2 + 1$ and $F_2 = x_2 - x_1$. Observe that

$$F_3 = S(F_1, F_2) = x_2 F_1 - x_1^2 F_2 = x_2 + x_1^3 \bmod G_1 = 0.$$

So, G is a Gröbner basis for I.

Algorithm 3: Buchberger's algorithm for computing a Gröbner basis

Input: A set of generators $F = \{f_1, f_2, \ldots, f_s\} \subset k[x_1, x_2, \ldots, x_n] \setminus \{0\}$ and a monomial ordering $<$.

Output: A Gröbner basis $G = \{g_1, g_2, \cdots g_t\}$ for $I = \langle f_1, f_2, \ldots, f_s \rangle$ with respect to $<$.

```
1  G_1 := F;   k := 1;
2  repeat
3        k := k + 1;
4        G_k := G_{k-1};
5        for each pair {f, g} ⊂ G_{k-1} do
6            r := S(f, g) mod G_{k-1};
7            if r ≠ 0 then G_k := G_k ∪ {r};
8        end do
9  until G_k = G_{k-1}
10 return G_k
```

Buchberger's algorithm for computing a Gröbner basis terminates when

$$S(g_i, g_j) \bmod G = 0$$

for all $g_i, g_j \in G$ for $i \neq j$. How can we be certain that this algorithm will always terminate? We know this because the ideals generated by the leading term of the G_i's will stabilize by the ascending chain condition, which states that every strictly increasing sequence of ideals in $k[x_1, \cdots, x_n]$ is finite. The ascending chain condition also implies that every ideal of $k[x_1, \cdots, x_n]$ has a finite generating set (the Hilbert basis theorem), so we know that the output of the Buchberger's algorithm is a Gröbner basis. In general, if G is a Gröbner basis for an ideal I, and $\mathrm{LT}(g_i)|\mathrm{LT}(g_j)$ for $i \neq j$, then $G \setminus \{g_j\}$ is a Gröbner basis for I.

Definition 1.40. Let $G = \{g_1, g_2, \cdots g_t\}$ be a Gröbner basis for $I = \langle f_1, f_2, \ldots, f_s \rangle$ with respect to a monomial order $>$. Then G is minimal if

(i) $\mathrm{LC}(g_i) = 1$ for all i,

(ii) $\mathrm{LT}(g_i) \nmid \mathrm{LT}(g_j)$ $\forall i \neq j$.

Furthermore, G is reduced if

(i) $\mathrm{LC}(g_i) = 1$ for all i,

(ii) $\mathrm{LT}(g_i)$ does not divide any term in g_j for $i \neq j$.

We now state the most important result about a Gröbner basis that concerns us which is how to use a Gröbner basis to solve a polynomial system.

Theorem 1.41 (The Elimination Theorem)**.** [Cox et al., 2015, Theorem 2, page 122] Let $I \subset k[x_1, x_2, \ldots, x_n]$ be an ideal and let G be a Gröbner basis of I with respect to lexicographical order where $x_1 > x_2 > \cdots, x_n$. Then $G \cap k[x_i, x_{i+1}, \ldots, x_n]$ is a Gröbner basis for the elimination ideal $I \cap k[x_i, x_{i+1}, \ldots, x_n]$ with respect to $>$ for $1 \leq i \leq n$.

We use the following example to demonstrate how to solve a parametric polynomial system using Gröbner basis by computing its resultant $\overline{R}$ as defined in Definition 1.6.

Example 1.42 (Heron2d system)**.** Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3\} \subset \mathbb{Q}[y_1, y_2, y_3][x_1, x_2, x_3]$ where

$$\begin{aligned}
\hat{f}_1 &= x_2^2 + x_3^2 - y_3^2 \\
\hat{f}_2 &= (x_2 - y_1)^2 + x_3^2 - y_2^2 \\
\hat{f}_3 &= -x_3 y_1 + 2x_1.
\end{aligned}$$

Using lexicographical order with $x_2 > x_3 > x_1$ such that the coefficient field is $\mathbb{Q}(Y)$, we obtain the Gröbner basis $G = \{\hat{g}_1, \hat{g}_2, \hat{g}_3\}$ where

$$\hat{g}_1 = x_1^2 + \frac{1}{16}y_1^4 - \frac{1}{8}y_1^2y_2^2 - \frac{1}{8}y_1^2y_3^2 + \frac{1}{16}y_2^4 - \frac{1}{8}y_2^2y_3^2 + \frac{1}{16}y_3^4,$$
$$\hat{g}_2 = x_3 - \frac{2x_1}{y_1} \text{ and}$$
$$\hat{g}_3 = x_2 + \frac{-y_1^2 + y_2^2 - y_3^2}{2y_1}.$$

If we seek to eliminate variables x_2, x_3 from $\mathcal{F}$ then the resultant $\overline{R} = \hat{g}_1$.

1.4 Thesis Outline

The generalized formulation of Dixon resultants is described in Chapter 2. A new probabilistic algorithm for extracting a sub-matrix of maximal rank from a given Dixon matrix D with its failure probability analysis is also presented in Chapter 2. Chapter 3 covers the presentation of all the background materials and the sparse interpolation tools that were necessary for designing our new algorithms. In Chapter 4, a new sparse multivariate rational function interpolation algorithm is developed. Our new Dixon resultant algorithm which uses our new sparse multivariate rational function interpolation method with benchmarks is presented in Chapter 5. The detailed failure probability analysis and the complexity analysis (in terms of the number of black box probes used) of our Dixon resultant algorithm are given in Chapter 6. Lastly, in Chapter 7, we present a new black box algorithm for solving a parametric linear system using our new sparse rational function interpolation method.

1.5 Published Work

Some parts of this thesis on Dixon resultants have been published in the proceedings of CASC' 2022 [Jinadu and Monagan, 2022a]. We have also presented some of our work on Dixon resultants at the Maple 2022 conference, and at the ISSAC 2022 poster session (a 4-page extended abstract was published [Jinadu and Monagan, 2022b] in ACM communications). Our results on solving parametric linear systems have been published in the proceedings of CASC' 2023 [Jinadu and Monagan, 2023]. A journal version of our results on Dixon resultants is currently being prepared.

1.6 Demo of Software

We demonstrate how our new two software programs work by solving Example 1.10 (the robot arms system) and interpolating the unique solution of a simple parametric linear system.

```
[ajinadu@cecm-maple ~]$ maple robotarms
    |\^/|     Maple 2022 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2022
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(CodeTools):

# Solving the robotarms system using our Dixon resultant code

> sys := [

> aa * t2^2 * t1^2 * b2^2 * b1^2 + 2 * l1 * t1^2 * b2^2 * b1^2 + aa *
> t1^2 * b2^2 * b1^2 - 2 * l1 * t2^2 * b2^2 * b1^2 + aa * t2^2 * b2^2
> * b1^2 + aa * b2^2 * b1^2
>  + 2 * l2 * t2^2 * t1^2 * b1^2 + aa * t2^2 * t1^2 * b1^2 + 2 * l2 *
> t1^2 * b1^2 + 2 * l1 * t1^2 * b1^2 + aa * t1^2 * b1^2 + 2 * l2 *
> t2^2 * b1^2 - 2 * l1 * t2^2 * b1^2
>  + aa * t2^2 * b1^2 + 2 * l2 * b1^2 + aa * b1^2 - 2 * l2 * t2^2 *
> t1^2 * b2^2 + aa * t2^2 * t1^2 * b2^2 - 2 * l2 * t1^2 * b2^2 + 2 *
> l1 * t1^2 * b2^2 + aa * t1^2 * b2^2
>  - 2 * l2 * t2^2 * b2^2 - 2 * l1 * t2^2 * b2^2 + aa * t2^2 * b2^2 -
> 2 * l2 * b2^2 + aa * b2^2 + aa * t2^2 * t1^2 + 2 * l1 * t1^2 + aa *
> t1^2 - 2 * l1 * t2^2 + aa * t2^2 + aa,


> 2 * l1 * t2 * t1^2 * b2^2 * b1^2 - 2 * l1 * t2^2 * t1 * b2^2 * b1^
> 2 - 2 * l1 * t1 * b2^2 * b1^2 + 2 * l1 * t2 * b2^2 * b1^2 + 2 * l2
> * t2^2 * t1^2 * b2 * b1^2
>  + 2 * l2 * t1^2 * b2 * b1^2 + 2 * l2 * t2^2 * b2 * b1^2 + 2 * l2
> * b2 * b1^2 + 2 * l1 * t2 * t1^2 * b1^2 - 2 * l1 * t2^2 * t1 * b1^
> 2 - 2 * l1 * t1 * b1^2 + 2 * l1 * t2 * b1^2
>  - 2 * l2 * t2^2 * t1^2 * b2^2 * b1 - 2 * l2 * t1^2 * b2^2 * b1 - 2
> * l2 * t2^2 * b2^2 * b1 - 2 * l2 * b2^2 * b1 - 2 * l2 * t2^2 * t1^
> 2 * b1 - 2 * l2 * t1^2 * b1 - 2 * l2 * t2^2 *
> b1 - 2 * l2 * b1 + 2 * l1 * t2 * t1^2 * b2^2 - 2 * l1 * t2^2 * t1
> * b2^2 - 2 * l1 * t1 * b2^2 + 2 * l1 * t2 * b2^2 + 2 * l2 * t2^2 *
> t1^2 * b2 + 2 * l2 * t1^2 * b2
>  + 2 * l2 * t2^2 * b2 + 2 * l2 * b2 + 2 * l1 * t2 * t1^2 - 2 * l1
> * t2^2 * t1 - 2 * l1 * t1 + 2 * l1 * t2,


>  - al^2 * x * t1^2 * b1^2 - x * t1^2 * b1^2 - l3 * al^2 * t1^2 * b1^2
> - l2 * al^2 * t1^2 * b1^2 - l1 * al^2 * t1^2 * b1^2 + l3 * t1^2 *
> b1^2 - l2 * t1^2 * b1^2
>  - l1 * t1^2 * b1^2 - al^2 * x * b1^2 - x * b1^2 - l3 * al^2 * b1^2
> - l2 * al^2 * b1^2 + l1 * al^2 * b1^2 + l3 * b1^2 - l2 * b1^2 + l1
> * b1^2 - 4 * l3 * al * t1^2 *
> b1 - 4 * l3 * al * b1 - al^2 * x * t1^2 - x * t1^2 + l3 * al^2 * t1^2
```

```
> + l2 * al^2 * t1^2 - l1 * al^2 * t1^2 - l3 * t1^2 + l2 * t1^2 - l1
> * t1^2 - al^2 * x - x
>  + l3 * al^2 + l2 * al^2 + l1 * al^2 - l3 + l2 + l1,

>  - al^2 * y * t1^2 * b1^2 - y * t1^2 * b1^2 - 2 * l3 * al * t1^2 *
> b1^2 + 2 * l1 * al^2 * t1 * b1^2 + 2 * l1 * t1 * b1^2 - al^2 * y *
> b1^2 - y * b1^2 - 2 * l3 * al * b1^2  + 2 * l3 * al^2 * t1^2 * b1 + 2
> * l2 * al^2 * t1^2 * b1 - 2 * l3 * t1^2 * b1 + 2 * l2 * t1^2 * b1
> + 2 * l3 * al^2 * b1 + 2 * l2 * al^2 * b1 - 2 * l3 * b1 + 2 * l2 *
> b1 - al^2 * y * t1^2 - y * t1^2 + 2 * l3 * al * t1^2 + 2 * l1 * al^2 *
> t1 + 2 * l1 * t1 - al^2 * y - y + 2 * l3 * al]:
> Y := indets(sys);
                    Y := {aa, al, b1, b2, l1, l2, l3, t1, t2, x, y}

> Z := [t1=x1, aa=y1, al=y2,l1=y3, l2=y4, l3=y5, x=y6, y=y7]:
> Sys := subs(Z,sys):
> elim := [t2,b1,b2];
                                  elim := [t2, b1, b2]

> X := [x1,y1,y2,y3,y4,y5,y6,y7];
                         X := [x1, y1, y2, y3, y4, y5, y6, y7]




>  read det; read newdeg; read dixon;read minor; read dixres; read nextprime; read Bmbot;

memory used=2.7MB, alloc=40.3MB, time=0.05
>      printf(" The number of polynomial equations = %d \n", nops(Sys) );
 The number of polynomial equations = 4
>      printf(" The number of variables  = %d \n", nops(elim)+1 );
 The number of variables  = 4
>      printf(" The number of parameters  = %d \n", nops(X)-1 );
 The number of parameters  = 7
>      M := dixonmatrix( Sys, elim ): # Construction of the Dixon matrix
Dixon: n=3
Dixon: matrix done
Dixon: #minors=2
Dixon: #delta=104256
Dixon: 32 x 32
>      rank,rows,cols := minor(M): # Extracting a maximal minor
minor: 32 x 32
minor: #nonzero=580
memory used=68.1MB, alloc=47.0MB, time=0.33
minor: starting elimination
>      printf("The rank is %d \n", rank);
The rank is 16
>      Bi := M[rows,cols]: # extract minor
>      L := StronglyConnectedBlocks(Bi):
>      BlockStructure := map(RowDimension,L);
                              BlockStructure := [8, 8]

>      member( min(BlockStructure),BlockStructure,'kat'):
>      if min(BlockStructure) = 1 then member( max(BlockStructure),BlockStructure,'kat'):
 fi:
```

```
>      n := BlockStructure[kat]:
>      R := convert(L[kat],listlist):
>      deg_M := [seq( max(seq(seq(degree(R[ii][kj],var),ii=1..n),kj=1..n)), var in X )]:
>      rt := [R,X,n ]:
>      GlobalCArray := Array(0..n-1,0..n-1,order=C_order,datatype=integer[8]):
>      E := CodeTools[Usage](DixonRes(BB,X)); # My Dixon resultant code
DO BASIS SHIFT
Number of Monic Square Factors = 3
Number of probes used to obtain the degree bounds is = 4096
memory used=124.0MB, alloc=79.0MB, time=2.88
Number of probes for the first prime = 13000
 The returned answer(s) is correct with w.h.p
memory used=67.43MiB, alloc change=0 bytes, cpu time=3.52s, real time=3.51s, gc time=24.0
4ms
        2   2   2       2   2           2   2   2       2   2           2   2   2
E := (x1  y2  y3  + 2 x1  y2  y3 y6 - x1  y2  y4  - 2 x1  y2  y4 y5 - x1  y2  y5

         2   2   2     2   2   2          2           2   2       2           2   2
     + x1  y2  y6  + x1  y2  y7  - 4 x1 y2  y3 y7 + x1  y3  + 2 x1  y3 y6 - x1  y4

           2           2   2     2   2     2   2     2   2       2           2   2
     + 2 x1  y4 y5 - x1  y5  + x1  y6  + x1  y7  + y2  y3  - 2 y2  y3 y6 - y2  y4

           2           2   2     2   2     2   2                   2               2
     - 2 y2  y4 y5 - y2  y5  + y2  y6  + y2  y7  - 4 x1 y3 y7 + y3  - 2 y3 y6 - y4

                   2     2     2     4   4       4   2   2       2   4       2   3
     + 2 y4 y5 - y5  + y6  + y7 ) (x1  y1  - 4 x1  y1  y4  + 2 x1  y1  - 8 x1  y1  y3

           2   2   2       2   2   2        2         2     4       3           2   2
     + 8 x1  y1  y3  - 8 x1  y1  y4  + 16 x1  y1 y3 y4  + y1  - 8 y1  y3 + 24 y1  y3

           2   2           3               2         4         2   2     2
     - 4 y1  y4  - 32 y1 y3  + 16 y1 y3 y4  + 16 y3  - 16 y3  y4 ) (x1  + 1)

>      printf( " The number of terms in E when fractions are cleared is %d \n",nops(expan
d(E)));
 The number of terms in E when fractions are cleared is 450
>      read xtrafac;
>     Ident_Factors( Sys,X,E); # This identifies the extraneous and good factors of E
 The number of terms in the largest polynomial to be interpolated is =  14
memory used=166.6MB, alloc=79.0MB, time=3.98
7, 5, JUNK(x1^2*y1^2+2*x1^2*y1*y4+y1^2-4*y1*y3+2*y1*y4+4*y3^2-4*y3*y4)
30, 14, GOOD-FACTOR(x1^2*y2^2*y3^2+2*x1^2*y2^2*y3*y6-x1^2*y2^2*y4^2-2*x1^2*y2^2*y4*y5-
x1^2*y2^2*y5^2+x1^2*y2^2*y6^2+x1^2*y2^2*y7^2-4*x1*y2^2*y3*y7+x1^2*y3^2+2*x1^2*y3*y6-x1^
2*y4^2+2*x1^2*y4*y5-x1^2*y5^2+x1^2*y6^2+x1^2*y7^2+y2^2*y3^2-2*y2^2*y3*y6-y2^2*y4^2-2*y2
^2*y4*y5-y2^2*y5^2+y2^2*y6^2+y2^2*y7^2-4*x1*y3*y7+y3^2-2*y3*y6-y4^2+2*y4*y5-y5^2+y6^2+
y7^2)
7, 5, JUNK(x1^2*y1^2-2*x1^2*y1*y4+y1^2-4*y1*y3-2*y1*y4+4*y3^2+4*y3*y4)
2, 1, GOOD-FACTOR(x1^2+1)



> quit
memory used=182.2MB, alloc=111.0MB, time=4.06
```

```
[ajinadu@cecm-maple ~]$ maple test2
    |\^/|     Maple 2022 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2022
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.

> with(LinearAlgebra):

# Solving a parametric linear system using sparse rational function interpolation
> gh := Matrix( [[y2+2,y3+6,0], [0,y3,y2+4]] ); # 2 by 3 augmented matrix
                                [y2 + 2    y3 + 6      0   ]
                          gh := [                           ]
                                [  0         y3      y2 + 4]


> gg := convert(gh,listlist):
> X := convert(indets(gg),list); ## These are the parameters
                                     X := [y2, y3]

> rt := [gg,X, RowDimension(gg)]:
> OrderRow := rt[3]:
> n,m := rt[3],rt[3]+1:
> deg_M := [seq( max(seq(seq(degree(rt[1][ii][kj],var),ii=1..n),kj=1..m)), var in rt[2] )
]:
> GlobalCArray := Array(1..n,1..m,order=C_order,datatype=integer[8]):
> read linsolve:
memory used=3.1MB, alloc=40.3MB, time=0.05
#trace(BB);
#trace(ParaLinSolve);
> gk1 := ParaLinSolve(BB,X); # This is the output of my code
All the degree bounds have been obtained
                              [-y2 y3 - 6 y2 - 4 y3 - 24   y2 + 4]
                        gk1 := [-------------------------, ------]
                              [      y2 y3 + 2 y3            y3  ]


> gk2 := LinearSolve(gh); # Using Maple's builtin linear solver
                                [  y2 y3 + 6 y2 + 4 y3 + 24]
                                [- -------------------------]
                                [        y3 (y2 + 2)       ]
                           gk2 := [                           ]
                                [          y2 + 4           ]
                                [          ------           ]
                                [            y3             ]


> quit
memory used=7.6MB, alloc=41.3MB, time=0.07
[ajinadu@cecm-maple ~]$
```

Chapter 2

Dixon Resultants

2.1 Summary of Contributions

In this chapter, we describe the generalized formulation of Dixon resultants in detail. But first, we highlight our main contributions here. In Subsection 2.2.1, we obtain a new formula (2.6) for constructing the cancellation matrix $\hat{\mathcal{C}}$ which produces the Dixon polynomial Δ_{X_e}. This new formula for constructing $\hat{\mathcal{C}}$ effectively avoids the occurrence of intermediate expression swell when computing Δ_{X_e}. Thus, Theorem 2.3 is new. In Subsection 2.2.1, we show that the Dixon resultant R is a polynomial in the elimination ideal $J = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle \cap \mathbb{Q}(Y)[x_1]$ (Theorem 2.13). In Subsection 2.2.4, we present a new probabilistic algorithm for extracting a sub-matrix of maximal rank (Algorithm 5) from a given Dixon matrix. In Subsection 2.2.5, we give a failure probability bound for Algorithm 5 using the Schwartz-Zippel Lemma.

2.2 Generalized Formulation

We first begin with our usual notations. Let $X = \{x_1, \ldots, x_n\}$ be the set of variables and let $Y = \{y_1, \ldots, y_m\}$ be the set of parameters with $n \geq 2$ and $m \geq 1$. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Q}[Y][X]$ be a parametric polynomial system where each $\hat{f}_i$ is a polynomial in variables X with coefficients in the polynomial ring $\mathbb{Q}[Y]$ such that $|\mathcal{F}| = |X|$. Let $I = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle$ be the ideal generated by $\mathcal{F}$ and let $J = I \cap \mathbb{Q}(Y)[x_1]$ be the elimination ideal in $\mathbb{Q}(Y)[x_1]$. Let $\overline{X}_e = \{\bar{x}_2, \cdots, \bar{x}_n\}$ be a set of new variables corresponding to the set $X_e = \{x_2, \ldots, x_n\}$ respectively. Let

$$\mathbf{x}^{\alpha} = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}.$$

Let $\mathcal{I} = \{2, \ldots, n\}$. For each $i \in \mathcal{I}$, let

$$\pi_i(\mathbf{x}^{\alpha}) = x_1^{\alpha_1} \bar{x}_2^{\alpha_2} \cdots \bar{x}_i^{\alpha_i} x_{i+1}^{\alpha_{i+1}} x_{i+2}^{\alpha_{i+2}} \cdots x_n^{\alpha_n}$$

so that

$$\pi_1(\mathbf{x}^\alpha) = \mathbf{x}^\alpha.$$

The evaluation map π_i extended naturally to polynomials is defined as

$$\pi_i(\hat{f}(x_1, x_2, \cdots, x_n)) = \hat{f}(x_1, \bar{x}_2, \cdots, \bar{x}_i, x_{i+1}, x_{i+2} \cdots, x_n). \tag{2.1}$$

Notice that for $i \in \mathcal{I}$ the evaluation map π_i replaces the $(i-1)$ variables after x_1 in $\hat{f} \in \mathcal{F}$ with the $\bar{x}_i$'s, and the main variable x_1 is never replaced.

There are four major steps involved in computing the Dixon resultant of a given parametric polynomial system $\mathcal{F}$. The first major step is to construct a matrix called the cancellation matrix $\mathcal{C}$, whose entries involve the given input parametric polynomial system $\mathcal{F}$. Then the determinant of $\mathcal{C}$, which is called the Dixon polynomial, is computed.

2.2.1 Computing the Dixon Polynomial

Definition 2.1. Given a parametric polynomial system $\mathcal{F}$, let $X_e = \{x_2, \ldots, x_n\}$ be the set of variables to be eliminated and let x_1 be the remaining variable. Let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3 \cdots, \bar{x}_n\}$ be the set of new variables corresponding to X_e. We define the $n \times n$ cancellation matrix

$$\mathcal{C} = \begin{pmatrix} \pi_1(\hat{f}_1) & \pi_1(\hat{f}_2) \ldots & \pi_1(\hat{f}_n) \\ \pi_2(\hat{f}_1) & \pi_2(\hat{f}_2) \ldots & \pi_2(\hat{f}_n) \\ \vdots & \vdots & \vdots \\ \pi_n(\hat{f}_1) & \pi_n(\hat{f}_2) \ldots & \pi_n(\hat{f}_n) \end{pmatrix}. \tag{2.2}$$

Definition 2.2. Let

$$P = \prod_{i=1}^{n-1} (X_{e_i} - \overline{X}_{e_i}) = \prod_{i=2}^{n} (x_i - \bar{x}_i)$$

and let

$$\Delta_{X_e} = \frac{\det(\mathcal{C})}{P}. \tag{2.3}$$

We refer to $\Delta_{X_e} \in \mathbb{Q}[Y, x_1][X_e, \overline{X}_e]$ as the Dixon polynomial of $\mathcal{F}$ with respect to X_e.

Notice that the determinant of the cancellation matrix $\det(\mathcal{C})$ is a multiple of the Dixon polynomial Δ_{X_e} in (2.3). Thus, if n is large, and since there are 2^{n-1} terms in

$$P = \prod_{i=2}^{n} (x_i - \bar{x}_i)$$

when expanded, then computing determinant of the cancellation matrix may result in large intermediate expression swell. This intermediate expression swell can cause the computation of the Dixon polynomial to become the most expensive step of the Dixon resultant method. We address this as follows.

Let $\text{Row}_j(\mathcal{C})$ denote the j-th row of the cancellation matrix $\mathcal{C}$. To avoid intermediate expression swell which may occur when computing Δ_{X_e}, we will not use formula (2.3) to compute Δ_{X_e}. Instead, we use an idea communicated to us by Lewis [Lewis, 2018a]. We first use a bottom up approach to construct a new cancellation matrix $\hat{\mathcal{C}}$ using the identity

$$\text{Row}_j(\hat{\mathcal{C}}) = \frac{\text{Row}_j(\mathcal{C}) - \text{Row}_{j-1}(\mathcal{C})}{x_j - \bar{x}_j} \tag{2.4}$$

for $j = n, n-1, \cdots 2$ and

$$\text{Row}_1(\hat{\mathcal{C}}) = \text{Row}_1(\mathcal{C}),$$

then we compute

$$\Delta_{X_e} = \det(\hat{\mathcal{C}}).$$

Upon simplification of identity (2.4), a new formula for getting the entries of the new cancellation matrix $\hat{\mathcal{C}}_{ij}$ is derived in Theorem 2.3, which avoids doing polynomial divisions. This new formula will also be used to obtain a height bound for Δ_{X_e} in Theorem 6.5 of Chapter 6 instead of using (2.3).

Theorem 2.3. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$ and let $d_{\max,j} = \max_{\hat{f}\in\mathcal{F}} \deg(\hat{f}, x_j)$ denote the maximum partial degree in variable x_j of all the polynomials $\hat{f} \in \mathcal{F}$. For $j = 2, 3, \ldots, n$, and $k = 1, 2, \ldots, n$, we have that

(i) the entries $\hat{\mathcal{C}}_{j,k}$ of the new cancellation matrix $\hat{\mathcal{C}}$ are polynomials and

$$\det(\hat{\mathcal{C}}) = \Delta_{X_e} = \frac{\det(\mathcal{C})}{P}.$$

(ii) Furthermore, by expressing

$$\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) = \sum_{u=0}^{d_{\max,j}} \tilde{f}_{u,j,k} x_j^u, \tag{2.5}$$

where

$$\tilde{f}_{u,j,k} \in \mathbb{Z}[x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_{j+1}, \ldots, x_n]$$

for $u \neq 0$, and

$$\tilde{f}_{0,j,k} \in \mathbb{Z}[x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n],$$

we obtain

$$\hat{\mathcal{C}}_{j,k} = \frac{\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)}{x_j - \bar{x}_j} = \sum_{i=0}^{d_{\max,j}-1} \left(\sum_{u=i}^{d_{\max,j}-1} \tilde{f}_{u+1,j,k}\ x_j^{u-i} \right) \bar{x}_j{}^i. \tag{2.6}$$

Proof. For $2 \leq j \leq n$, observe that $\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) \equiv 0 \pmod{()x_j - \bar{x}_j}$ since

$$\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) = \hat{f}_k(x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n) - \hat{f}_k(x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_j \cdots, x_n).$$

Thus, the difference quotient

$$\frac{\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)}{x_j - \bar{x}_j}$$

has no remainder, which implies that the entries $\hat{\mathcal{C}}_{j,k}$ are polynomials. Now let

$$\mathcal{E} = \begin{pmatrix} \pi_1(\hat{f}_1) & \pi_1(\hat{f}_2) & \ldots & \pi_1(\hat{f}_n) \\ \pi_2(\hat{f}_1) - \pi_1(\hat{f}_1) & \pi_2(\hat{f}_2) - \pi_1(\hat{f}_2) & \ldots & \pi_2(\hat{f}_n) - \pi_1(\hat{f}_n) \\ \pi_3(\hat{f}_1) - \pi_2(\hat{f}_1) & \pi_3(\hat{f}_2) - \pi_2(\hat{f}_2) & \ldots & \pi_3(\hat{f}_n) - \pi_2(\hat{f}_n) \\ \vdots & \vdots & \vdots & \\ \pi_n(\hat{f}_1) - \pi_{n-1}(\hat{f}_1) & \pi_n(\hat{f}_2) - \pi_{n-1}(\hat{f}_2) & \ldots & \pi_n(\hat{f}_n) - \pi_{n-1}(\hat{f}_n) \end{pmatrix}$$

where

$$\mathrm{Row}_i(\mathcal{E}) = \mathrm{Row}_i(\mathcal{C}) - \mathrm{Row}_{i-1}(\mathcal{C}).$$

From elementary linear algebra, we know that adding a multiple of a row of a matrix to another row of the same matrix does not affect its determinant. Thus, we have

$$\det(\mathcal{E}) = \det(\mathcal{C}).$$

Next, since $(x_j - \bar{x}_j) | \left(\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)\right)$, we can write

$$\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) = (x_j - \bar{x}_j)\hat{\mathcal{C}}_{j,k}$$

for some polynomial $\hat{\mathcal{C}}_{j,k} \in \mathbb{Z}[x_1, \bar{x}_2, \ldots, \bar{x}_j, x_j, x_{j+1}, \ldots, x_n]$. So, $\mathcal{E}$ becomes

$$\mathcal{E} = \begin{pmatrix} \pi_1(\hat{f}_1) & \pi_1(\hat{f}_2) & \ldots & \pi_1(\hat{f}_n) \\ (x_2 - \bar{x}_2)\hat{\mathcal{C}}_{2,1} & (x_2 - \bar{x}_2)\hat{\mathcal{C}}_{2,2} & \ldots & (x_2 - \bar{x}_2)\hat{\mathcal{C}}_{2,n} \\ (x_3 - \bar{x}_3)\hat{\mathcal{C}}_{3,1} & (x_3 - \bar{x}_3)\hat{\mathcal{C}}_{3,2} & \ldots & (x_3 - \bar{x}_3)\hat{\mathcal{C}}_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ (x_n - \bar{x}_n)\hat{\mathcal{C}}_{n,1} & (x_n - \bar{x}_n)\hat{\mathcal{C}}_{n,2} & \ldots & (x_n - \bar{x}_n)\hat{\mathcal{C}}_{n,n} \end{pmatrix}.$$

Therefore,

$$\det(\mathcal{E}) = \det(\hat{\mathcal{C}}) \prod_{j=2}^{n} (x_j - \bar{x}_j).$$

Using (2.3), we have

$$\Delta_{X_e} = \frac{\det(\mathcal{C})}{\prod_{j=2}^{n}(x_j - \bar{x}_j)} = \frac{\det(\mathcal{E})}{\prod_{j=2}^{n}(x_j - \bar{x}_j)} = \frac{\det(\hat{\mathcal{C}}) \prod_{j=2}^{n}(x_j - \bar{x}_j)}{\prod_{j=2}^{n}(x_j - \bar{x}_j)} = \det(\hat{\mathcal{C}}).$$

This completes part (i). We proceed with the proof of part (ii) as follows. Recall that the formal power series representation of $\dfrac{1}{x_j - \bar{x}_j}$ when expanded about $\bar{x}_j$ can be written as

$$\frac{1}{x_j - \bar{x}_j} = \sum_{i=1}^{\infty} x_j^{-i} \bar{x}_j{}^{i-1}.$$

Using (2.5), we have

$$\begin{aligned}\hat{\mathcal{C}}_{j,k} = \frac{\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)}{x_j - \bar{x}_j} &= \sum_{u=0}^{d_{\max,j}} \tilde{f}_{u,j,k} x_j^u \left(\sum_{i=1}^{\infty} x_j^{-i} \bar{x}_j{}^{i-1} \right) \\ &= \sum_{u=0}^{d_{\max,j}} \tilde{f}_{u,j,k} x_j^u \left(\sum_{i=1}^{u} x_j^{-i} \bar{x}_j{}^{i-1} \right) + \underbrace{\sum_{u=0}^{d_{\max,j}} \tilde{f}_{u,j,k} x_j^u \left(\sum_{i=u+1}^{\infty} x_j^{-i} \bar{x}_j{}^{i-1} \right)}_{G}.\end{aligned}$$

Since the entries $\hat{\mathcal{C}}_{j,k}$ are polynomials, we have that $G = 0$. Therefore,

$$\hat{\mathcal{C}}_{j,k} = \sum_{u=0}^{d_{\max,j}} \tilde{f}_{u,j,k} x_j^u \left(\sum_{i=1}^{u} x_j^{-i} \bar{x}_j{}^{i-1} \right). \tag{2.7}$$

Observe that

$$\sum_{k=1}^{u} \frac{\bar{x}_j^{k-1}}{x_j^{k-1}} = \frac{1 - \left(\frac{\bar{x}_j}{x_j}\right)^u}{1 - \frac{\bar{x}_j}{x_j}} = \frac{x_j^u - \bar{x}_j^u}{x_j^{u-1}(x_j - \bar{x}_j)}.$$

So,

$$\begin{aligned} x_j^u \left(\sum_{i=1}^{u} x_j^{-i-1} \bar{x}_j{}^{i} \right) = \frac{x_j^u}{x_j} \left(\sum_{i=1}^{u} x_j^{-i} \bar{x}_j{}^{i} \right) &= \frac{x_j^{u-1}\left(x_j^u - \bar{x}_j^u\right)}{x_j^{u-1}(x_j - \bar{x}_j)} \\ &= \frac{x_j^u - \bar{x}_j^u}{x_j - \bar{x}_j} \\ &= \frac{\bar{x}_j^u \left(\left(\frac{x_j}{\bar{x}_j}\right)^u - 1 \right)}{\bar{x}_j \left(\frac{x_j}{\bar{x}_j} - 1 \right)} = \frac{\bar{x}_j^{u-1} \left(\left(\frac{x_j}{\bar{x}_j}\right)^u - 1 \right)}{\left(\frac{x_j}{\bar{x}_j} - 1 \right)} \\ &= \bar{x}_j^{u-1} \sum_{i=1}^{u} \frac{x_j^{i-1}}{\bar{x}_j^{i-1}}. \end{aligned}$$

Therefore, we get

$$x_j^u \left(\sum_{i=1}^{u} x_j^{-i-1} \bar{x}_j{}^{i} \right) = \bar{x}_j^{u-1} \sum_{i=1}^{u} x_j^{i-1} \bar{x}_j^{-i+1} = \sum_{i=1}^{u} x_j^{i-1} \bar{x}_j^{u-i} = \sum_{i=0}^{u-1} x_j^i \ \bar{x}_j^{u-i-1}.$$

Thus, (2.7) becomes

$$\hat{\mathcal{C}}_{j,k} = \sum_{u=1}^{d_{\max,j}} \tilde{f}_{u,j,k} \left(\sum_{i=0}^{u-1} x_j^i \ \bar{x}_j^{\,u-i-1} \right). \tag{2.8}$$

By expanding (2.8), we get

$$\begin{aligned}
\sum_{u=1}^{d_{\max,j}} \tilde{f}_{u,j,k} \left(\sum_{i=0}^{u-1} x_j^i \ \bar{x}_j^{\,u-i-1} \right) &= \tilde{f}_{1,j,k} + \tilde{f}_{2,j,k} \left(\bar{x}_j + x_j \right) + \tilde{f}_{3,j,k} \left(\bar{x}_j^{\,2} + x_j \bar{x}_j + x_j^2 \right) \\
&\quad + \tilde{f}_{4,j,k} \left(\bar{x}_j^{\,3} + x_j \bar{x}_j^{\,2} + x_j^2 \bar{x}_j + x_j^3 \right) + \cdots + \\
&\quad + \tilde{f}_{d_{\max,j}-1,j,k} \left(\sum_{i=0}^{d_{\max,j}-2} x_j^i \ \bar{x}_j^{\,d_{\max,j}-2-i} \right) \\
&\quad + \tilde{f}_{d_{\max,j},j,k} \left(\sum_{i=0}^{d_{\max,j}-1} x_j^i \ \bar{x}_j^{\,d_{\max,j}-1-i} \right).
\end{aligned}$$

Rearranging the terms on the right-hand side of the above sum in powers of $\bar{x}_j$ yields

$$\begin{aligned}
\hat{\mathcal{C}}_{j,k} &= \bar{x}_j^0 \left(\tilde{f}_{1,j,k} + \tilde{f}_{2,j,k} x_j + \tilde{f}_{3,j,k} x_j^{\,2} + \tilde{f}_{4,j,k} x_j^{\,3} + \cdots + \tilde{f}_{d_{\max,j},j,k} x^{d_{\max,j}-1} \right) \\
&\quad + \bar{x}_j \left(\tilde{f}_{2,j,k} + \tilde{f}_{3,j,k} x_j + \tilde{f}_{4,j,k} x_j^{\,2} + \cdots + \tilde{f}_{d_{\max,j},j,k} x^{d_{\max,j}-2} \right) \\
&\quad + \bar{x}_j^2 \left(\tilde{f}_{3,j,k} + \tilde{f}_{4,j,k} x_j + \tilde{f}_{5,j,k} x_j^2 + \cdots + \tilde{f}_{d_{\max,j},j,k} x^{d_{\max,j}-3} \right) \\
&\quad + \bar{x}_j^3 \left(\tilde{f}_{4,j,k} + \tilde{f}_{5,j,k} x_j + \tilde{f}_{6,j,k} x_j^2 + \cdots + \tilde{f}_{d_{\max,j},j,k} x^{d_{\max,j}-4} \right) + \\
&\quad \vdots \\
&\quad + \bar{x}_j^{d_{\max,j}-2} \left(\tilde{f}_{d_{\max,j}-1,j,k} + \tilde{f}_{d_{\max,j},j,k} \ x \right) + \bar{x}_j^{d_{\max,j}-1} \tilde{f}_{d_{\max,j},j,k}.
\end{aligned}$$

By examining the sum on the right-hand side of the above sum, we can write

$$\begin{aligned}
\hat{\mathcal{C}}_{j,k} &= \bar{x}_j^0 \left(\sum_{w=1}^{d_{\max,j}} \tilde{f}_{w,j,k} x^{w-1} \right) + \bar{x}_j^{\,1} \left(\sum_{w=2}^{d_{\max,j}} \tilde{f}_{w,j,k} x^{w-2} \right) + \bar{x}_j^2 \left(\sum_{w=3}^{d_{\max,j}} \tilde{f}_{w,j,k} x^{w-3} \right) \\
&\quad + \bar{x}_j^3 \left(\sum_{w=4}^{d_{\max,j}} \tilde{f}_{w,j,k} x^{w-4} \right) + \cdots + \bar{x}_j^{d_{\max,j}-2} \left(\sum_{w=d_{\max,j}-1}^{d_{\max,j}} \tilde{f}_{w,j,k} x^{w-(d_{\max,j}-1)} \right) \\
&\quad + \bar{x}_j^{d_{\max,j}-1} \tilde{f}_{d_{\max,j},j,k}.
\end{aligned}$$

Substituting $i = w - 1$ in the above sum yields

$$\begin{aligned}\hat{\mathcal{C}}_{j,k} &= \bar{x}_j^0 \left(\sum_{i=0}^{d_{\max,j}-1} \tilde{f}_{i+1,j,k} x^{i-0} \right) + \bar{x}_j{}^1 \left(\sum_{i=1}^{d_{\max,j}-1} \tilde{f}_{i+1,j,k} x^{i-1} \right) + \bar{x}_j^2 \left(\sum_{i=2}^{d_{\max,j}-1} \tilde{f}_{i+1,j,k} x^{i-2} \right) \\ &+ \bar{x}_j^3 \left(\sum_{i=3}^{d_{\max,j}-1} \tilde{f}_{i+1,j,k} x^{i-3} \right) + \cdots + \bar{x}_j^{d_{\max,j}-2} \left(\sum_{i=d_{\max,j}-2}^{d_{\max,j}-1} \tilde{f}_{i+1,j,k} x^{i-(d_{\max,j}-2)} \right) \\ &+ \bar{x}_j^{d_{\max,j}-1} \tilde{f}_{d_{\max,j},j,k} \left(x^{(d_{\max,j}-1-(d_{\max,j}-1)} \right) \\ &= \sum_{i=0}^{d_{\max,j}-1} \left(\sum_{u=i}^{d_{\max,j}-1} \tilde{f}_{u+1,j,k} \; x_j^{u-i} \right) \bar{x}_j{}^i.\end{aligned}$$

□

We remind the reader again that the essence of simplifying (2.8) further to get (2.6) is to obtain a better height bound for the Dixon resultant R in Theorem 6.5. We now give the following example to illustrate how to use (2.6).

Example 2.4. Let

$$\hat{f}_1 = x_1^4 y_1^4 - x_2^3 + \sum_{j=3}^{100} y_j^3 x_j^3.$$

Then

$$\hat{\mathcal{C}}_{2,1} = \frac{\pi_2(\hat{f}_1) - \pi_1(\hat{f}_1)}{x_2 - \bar{x}_2} = \frac{x_2^3 - \bar{x}_2^3}{x_2 - \bar{x}_2} = \bar{x}_2{}^2 + \bar{x}_2 x_2 + x_2^2.$$

Using our new formula (2.6) which avoids polynomial divisions, we have

$$d_{\max,2} = 3, \tilde{f}_{0,2,1} = -\bar{x}_2^3, \tilde{f}_{1,2,1} = \tilde{f}_{2,2,1} = 0, \tilde{f}_{3,2,1} = 1.$$

Hence,

$$\begin{aligned}\hat{\mathcal{C}}_{2,1} &= (\tilde{f}_{1,2,1} x^{0-0} + \tilde{f}_{2,2,1} x_2^{1-0} + \tilde{f}_{3,2,1} x_2^{2-0}) + (\tilde{f}_{2,2,1} x_2^{1-1} + \tilde{f}_{3,2,1} x_2^{2-1}) \bar{x}_2 + (\tilde{f}_{3,2,1} x_2^{2-2}) \bar{x}_2{}^2 \\ &= \bar{x}_2{}^2 + \bar{x}_2 x_2 + x_2^2.\end{aligned}$$

To illustrate and quantify the gain realized by using (2.6) instead of (2.3) to construct a cancellation matrix, we provide the following real example.

Example 2.5 (Heron 3d system)**.** Consider the parametric system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \ldots, \hat{f}_6\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_6][x_1, x_2, \ldots, x_4]$ listed in Appendix A.4. Suppose we want to eliminate variables $X_e = \{x_2, x_3, \ldots, x_6\}$ from $\mathcal{F}$. Let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3, \ldots, \bar{x}_6\}$ be the set of new variables corresponding to X_e. Using (2.6), we have

$$\hat{\mathcal{C}} = \begin{pmatrix} \hat{f}_1 & \hat{f}_2 & \hat{f}_3 & \hat{f}_4 & \hat{f}_5 & \hat{f}_6 \\ x_2+\bar{x}_2 & x_2-2y_1+\bar{x}_2 & 0 & 0 & x_2-2x_4+\bar{x}_2 & 0 \\ x_3+\bar{x}_3 & x_3+\bar{x}_3 & 0 & 0 & x_3-2x_5+\bar{x}_3 & -x_6y_1 \\ 0 & 0 & x_4+\bar{x}_4 & x_4-2y_1+\bar{x}_4 & x_4-2\bar{x}_2+\bar{x}_4 & 0 \\ 0 & 0 & x_5+\bar{x}_5 & x_5+\bar{x}_5 & x_5-2\bar{x}_3+\bar{x}_5 & 0 \\ 0 & 0 & x_6+\bar{x}_6 & x_6+\bar{x}_6 & x_6+\bar{x}_6 & -\bar{x}_3y_1 \end{pmatrix}$$

Thus, taking the determinant of the $\hat{\mathcal{C}}$, we obtain the Dixon polynomial Δ_{X_e} which is

$$\begin{aligned}
&8x_2x_5^2\bar{x}_2\bar{x}_3y_1^3 - 4x_2x_5^2\bar{x}_3y_1^4 + 4x_2x_5^2\bar{x}_3y_1^2y_2^2 - 4x_2x_5^2\bar{x}_3y_1^2y_3^2 + 8x_2x_5\bar{x}_2\bar{x}_3\bar{x}_5y_1^3 \\
&- 4x_2x_5\bar{x}_3\bar{x}_5y_1^4 + 4x_2x_5\bar{x}_3\bar{x}_5y_1^2y_2^2 - 4x_2x_5\bar{x}_3\bar{x}_5y_1^2y_3^2 + 8x_2x_6^2\bar{x}_2\bar{x}_3y_1^3 - 4x_2x_6^2\bar{x}_3y_1^4 \\
&+ 4x_2x_6^2\bar{x}_3y_1^2y_2^2 - 4x_2x_6^2\bar{x}_3y_1^2y_3^2 + 8x_2x_6\bar{x}_2\bar{x}_3\bar{x}_6y_1^3 - 4x_2x_6\bar{x}_3\bar{x}_6y_1^4 + 4x_2x_6\bar{x}_3\bar{x}_6y_1^2y_2^2 \\
&- 4x_2x_6\bar{x}_3\bar{x}_6y_1^2y_3^2 - 8x_3x_4x_5\bar{x}_2\bar{x}_3y_1^3 + 4x_3x_4x_5\bar{x}_3y_1^4 - 4x_3x_4x_5\bar{x}_3y_1^2y_2^2 + 4x_3x_4x_5\bar{x}_3y_1^2y_3^2 \\
&- 8x_3x_4\bar{x}_2\bar{x}_3\bar{x}_5y_1^3 + 8x_3x_4\bar{x}_3^2\bar{x}_4y_1^3 - 4x_3x_4\bar{x}_3^2y_1^4 + 4x_3x_4\bar{x}_3^2y_1^2y_4^2 - 4x_3x_4\bar{x}_3^2y_1^2y_6^2 \\
&- 4x_3x_4\bar{x}_3\bar{x}_5y_1^2y_2^2 + 4x_3x_4\bar{x}_3\bar{x}_5y_1^2y_3^2 + 4x_3x_5\bar{x}_2\bar{x}_3y_1^4 - 4x_3x_5\bar{x}_2\bar{x}_3y_1^2y_4^2 + 4\bar{x}_2\bar{x}_3^2\bar{x}_5y_1^4 \\
&+ 4x_3x_5\bar{x}_2\bar{x}_3y_1^2y_6^2 + 8x_3x_5\bar{x}_3^2\bar{x}_5y_1^3 - 4x_3x_5\bar{x}_3y_1^3y_3^2 + 4x_3x_5\bar{x}_3y_1^3y_5^2 - 4x_3x_5\bar{x}_3y_1^3y_6^2 \\
&+ 4x_3\bar{x}_2\bar{x}_3\bar{x}_5y_1^4 - 4x_3\bar{x}_2\bar{x}_3\bar{x}_5y_1^2y_4^2 + 4x_3\bar{x}_2\bar{x}_3\bar{x}_5y_1^2y_6^2 - 4x_3\bar{x}_3^2\bar{x}_4y_1^4 + 4x_3\bar{x}_3^2\bar{x}_4y_1^2y_4^2 \\
&+ 8x_3\bar{x}_3^2y_1^3y_6^2 - 4x_3\bar{x}_3\bar{x}_5y_1^3y_3^2 + 4x_3\bar{x}_3\bar{x}_5y_1^3y_5^2 - 4x_3\bar{x}_3\bar{x}_5y_1^3y_6^2 - 8x_4x_5\bar{x}_2\bar{x}_3^2y_1^3 \\
&+ 4x_4x_5\bar{x}_3^2y_1^2y_3^2 - 8x_4\bar{x}_2\bar{x}_3^2\bar{x}_5y_1^3 + 8x_4\bar{x}_3^3\bar{x}_4y_1^3 - 4x_4\bar{x}_3^3y_1^4 + 4x_4\bar{x}_3^3y_1^2y_4^2 - 4x_4\bar{x}_3^3y_1^2y_6^2 \\
&- 4x_4\bar{x}_3^2\bar{x}_5y_1^2y_2^2 + 4x_4\bar{x}_3^2\bar{x}_5y_1^2y_3^2 - 4x_5^2\bar{x}_2\bar{x}_3y_1^4 + 4x_5^2\bar{x}_2\bar{x}_3y_1^2y_2^2 - 4x_5^2\bar{x}_2\bar{x}_3y_1^2y_3^2 - 8x_5^2\bar{x}_3^3y_1^3 \\
&+ 4x_5\bar{x}_2\bar{x}_3^2y_1^4 - 4x_5\bar{x}_2\bar{x}_3^2y_1^2y_4^2 + 4x_5\bar{x}_2\bar{x}_3^2y_1^2y_6^2 - 4x_5\bar{x}_2\bar{x}_3\bar{x}_5y_1^4 + 4x_5\bar{x}_2\bar{x}_3\bar{x}_5y_1^2y_2^2 \\
&- 4x_5\bar{x}_3^2y_1^3y_3^2 + 4x_5\bar{x}_3^2y_1^3y_5^2 - 4x_5\bar{x}_3^2y_1^3y_6^2 + 8x_5\bar{x}_3\bar{x}_5y_1^3y_3^2 - 4x_6^2\bar{x}_2\bar{x}_3y_1^4 + 4x_6^2\bar{x}_2\bar{x}_3y_1^2y_2^2 \\
&- 8x_6^2\bar{x}_3^3y_1^3 + 8x_6^2\bar{x}_3y_1^3y_3^2 - 4x_6\bar{x}_2\bar{x}_3\bar{x}_6y_1^4 + 4x_6\bar{x}_2\bar{x}_3\bar{x}_6y_1^2y_2^2 - 4x_6\bar{x}_2\bar{x}_3\bar{x}_6y_1^2y_3^2 + 8x_6\bar{x}_3\bar{x}_6y_1^3y_3^2 \\
&- 4\bar{x}_2\bar{x}_3^2\bar{x}_5y_1^2y_4^2 + 4\bar{x}_2\bar{x}_3^2\bar{x}_5y_1^2y_6^2 - 4\bar{x}_3^3\bar{x}_4y_1^4 + 4\bar{x}_3^3\bar{x}_4y_1^2y_4^2 - 4\bar{x}_3^3\bar{x}_4y_1^2y_6^2 - 4\bar{x}_3^2\bar{x}_5y_1^3y_3^2 + 8\bar{x}_3^3y_1^3y_6^2 \\
&- 4\bar{x}_3^2\bar{x}_5y_1^3y_6^2 - 48x_1x_3x_6\bar{x}_3y_1^2 - 48x_1x_3\bar{x}_3\bar{x}_6y_1^2 - 48x_1x_6\bar{x}_3^2y_1^2 - 48x_1\bar{x}_3^2\bar{x}_6y_1^2 + 4\bar{x}_3^2\bar{x}_5y_1^3y_2^2 \\
&+ 8x_3x_6\bar{x}_3^2\bar{x}_6y_1^3 + 8x_5^2\bar{x}_3y_1^3y_3^2 - 4x_3\bar{x}_3^2\bar{x}_4y_1^2y_6^2 + 4x_3x_4\bar{x}_3\bar{x}_5y_1^4 - 4x_6^2\bar{x}_2\bar{x}_3y_1^2y_3^2 + 4x_4x_5\bar{x}_3^2y_1^4 \\
&- 4x_4x_5\bar{x}_3^2y_1^2y_2^2 - 4x_5\bar{x}_2\bar{x}_3\bar{x}_5y_1^2y_3^2 + 4x_4\bar{x}_3^2\bar{x}_5y_1^4.
\end{aligned}$$

We determined that $|\Delta_{X_e}| = |\det(\hat{\mathcal{C}})| = 96$. If we choose to compute Δ_{X_e} using (2.3), then $|\det(\mathcal{C})| = 928$, which yields an expression swell factor of about 10.

Note that the order in which the variables to be eliminated X_e are replaced with the corresponding new variables $\overline{X}_e$ can result in different Dixon polynomials (but they are still equivalent) with respect to their various orderings. In this thesis, we do not focus on exploiting the variable orderings. For more details on ordering of these variables, we refer the reader to [Lewis, 2019] and [Chtcherba and Kapur, 2002].

2.2.2 Constructing a Dixon Matrix

After computing the Dixon polynomial, the second major step in the Dixon's resultant method is to build the Dixon matrix D from the Dixon polynomial Δ_{X_e}.

To do this, we first need degree bounds for the Dixon polynomial Δ_{X_e} in x_i and $\bar{x_i}$ using (2.3). Let $d_{\max,i} = \max_{\hat{f}\in\mathcal{F}} \deg(\hat{f}, x_i)$ denote the maximum partial degree of all the polynomials in $\mathcal{F}$ with respect to the variable x_i, and let the $n \times n$ cancellation matrix $\mathcal{C}$ be viewed as

$$\mathcal{C} = \begin{pmatrix} \text{Row}_1(\mathcal{C})(x_1, x_2, \ldots, x_n) \\ \text{Row}_2(\mathcal{C})(x_1, \bar{x}_2, \ldots, x_n) \\ \vdots \\ \text{Row}_n(\mathcal{C})(x_1, \bar{x}_2, \ldots, \bar{x}_n) \end{pmatrix}$$

where $\text{Row}_j(\mathcal{C})(x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n)$ indicates that all the entries of the $j-$th row of matrix $\mathcal{C}$ are polynomials in variables $x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n$. Since the evaluation map π_i defined in (2.1) does not affect x_1, we have that

$$\deg(\Delta_{X_e}, x_1) \leq n d_{\max,1}.$$

Notice that

$$\deg(\Delta_{X_e}, x_2) \leq d_{\max,2} - 1,$$

and

$$\deg(\Delta_{X_e}, \bar{x}_2) \leq (n-1) d_{\max,2} - 1,$$

because x_2 is only present in the first row, since it is been replaced with $\bar{x}_2$ from row 2 to row n of matrix $\mathcal{C}$, and after computing $\det(\mathcal{C})$, we also have to do a division by $x_2 - \bar{x}_2$ from P in (2.3). Using the same reasoning, for $2 \leq i \leq n$, it follows that

$$\deg(\Delta_{X_e}, x_i) \leq (i-1) d_{\max,i} - 1, \tag{2.9}$$

and

$$\deg(\Delta_{X_e}, \bar{x}_i) \leq (n-i+1) d_{\max,i} - 1. \tag{2.10}$$

Let $\overline{V}$ be a monomial column vector in $\overline{X}_e$ when Δ_{X_e} is viewed as a polynomial in $\overline{X}_e$ and let V be a monomial row vector in X_e when Δ_{X_e} is viewed as a polynomial in X_e. Notice that

$$|\overline{V}| \leq \prod_{i=2}^{n} (n-i+1) d_{\max,i} - 1 + 1 \leq (n-1)! \prod_{i=2}^{n} d_{\max,i},$$

because $|\{\bar{x}_i\}|= n-1$, and the maximum number of possible monomials that appears in $\overline{V}$ in $\bar{x}_i$ including the constant term 1 is at most $(n-i+1)d_{\max,i}$. Similarly, one can see that

$$|V| \leq \prod_{i=2}^{n}(i-1)d_{\max,i} - 1 + 1 \leq (n-1)! \prod_{i=2}^{n} d_{\max,i}.$$

Writing out all the possible elements of vectors V and $\overline{V}$ would be of the form

$$V = \left[\prod_{i=2}^{n} x_i^{(i-1)d_{\max,i}-1} \quad \dots \quad x_2^{d_{\max,2}-1} \quad \dots x_n^{(n-1)d_{max,n}-1} \quad \cdots \quad x_{n-1} \quad \dots, x_2 \quad 1 \right],$$

and

$$\overline{V} = \left[\prod_{i=2}^{n} \bar{x}_i^{(n-i+1)d_{\max,i}-1} \quad \dots \quad \bar{x_2}^{(n-1)d_{\max,2}-1} \quad \dots \bar{x}_n^{d_{max,n}-1} \quad \bar{x}_{n-1} \quad \dots, \bar{x}_2 \quad 1 \right]^T.$$

We remark that not all the monomials in V and $\overline{V}$ are typically present for non-generic polynomials.

Example 2.6. Let $X = \{x_1, x_2, x_3\}$ and $d_{\max,k} = 2$ for $1 \leq k \leq 3$. Suppose we want to eliminate $X_e = \{x_2, x_3\}$. The possible monomials in V and $\overline{V}$ can be expressed explicitly as

$$V = \left[1, x_2, x_3, x_2x_3, x_3^2, x_3^3, x_2x_3^2, x_2x_3^3\right],$$

and

$$\overline{V} = \left[1, \bar{x}_2, \bar{x}_3, \bar{x}_2^2, \bar{x}_2^3, \bar{x}_3\bar{x}_2^3, \bar{x}_3\bar{x}_2^2, \bar{x}_3\bar{x}_2\right].$$

Hence,

$$|V|= |\overline{V}|= 8 = (3-1)!\, 2^2.$$

Lemma 2.7. Let $\overline{V}$ be a monomial column vector in variables $\overline{X}_e$ when Δ_{X_e} is viewed as a polynomial in $\overline{X}_e$ and let V be a monomial row vector in X_e when Δ_{X_e} is viewed as a polynomial in X_e. The Dixon polynomial Δ_{X_e} can be written in bilinear form as

$$\Delta_{X_e} = VD\overline{V} \tag{2.11}$$

where D is an $s \times t$ matrix with entries in $\mathbb{Q}[Y, x_1]$ and $t, s \leq (n-1)! \prod_{i=2}^{n} d_{\max,i}$.

Proof. Since $\Delta_{X_e} \in \mathbb{Q}[Y, x_1][X_e, \overline{X}_e]$ is a polynomial in two disjoint set of variables X_e and $\overline{X}_e$, then it admits the bilinear form $\Delta_{X_e} = VD\overline{V}$. Finally, the dimension of D follows from the fact that $|V| \leq (n-1)! \prod_{i=2}^{n} d_{\max,i}$ and $|\overline{V}| \leq (n-1)! \prod_{i=2}^{n} d_{\max,i}$. Otherwise, the matrix multiplication of $VD\overline{V}$ will not be possible. □

Definition 2.8. The matrix D in the Lemma 2.7 is called the Dixon matrix.

Algorithm 4: ConstructDixon

Input: A parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \ldots, \hat{f}_n\} \subset \mathbb{Q}[Y][X]$ and the list of variables $X_e = \{x_2, x_3, \ldots, x_n\}$ to be eliminated from $\mathcal{F}$ such that $n \geq 2$.

Output: The Dixon matrix D as defined in Lemma 2.8.

Let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n\}$ be new variables corresponding to X_e.
Let $\hat{\mathcal{C}}$ be a $n \times n$ zero matrix.
for $j = 1, 2, \cdots n$ **do**
    $\hat{\mathcal{C}}_{1,j} \leftarrow \hat{f}_j$
end
for $i = 2, \cdots n$ **do**
    for $j = 1, 2, \cdots n$ **do**
        Construct $\hat{\mathcal{C}}_{i,j}$ using (2.6) //$\hat{\mathcal{C}}$ is the cancellation matrix
    end
end
$\Delta_{X_e} \leftarrow \det(\hat{\mathcal{C}})$ // Δ_{X_e} is the Dixon polynomial
Let V be the vector of monomials of Δ_{X_e} in variables X_e
Let $\overline{V}$ be the vector of monomials of Δ_{X_e} in variables $\overline{X}_e$
Let D be a $|V| \times |\overline{V}|$ zero matrix.
for $i = 1, 2, \ldots, |V|$ **do**
    $v \leftarrow V_i$
    $c \leftarrow \text{coeff}(\Delta_{X_e}, v)$ // the coefficient of Δ_{X_e} with respect to v
    for $j = 1, 2, \ldots, |\overline{V}|$ **do**
        $\overline{v} \leftarrow \overline{V}_j$
        $D_{i,j} \leftarrow \text{coeff}(c, \overline{v})$ // the coefficient of c with respect to $\overline{v}$
    end
end
return D

Given a parametric polynomial system $\mathcal{F}$, Algorithm 4 can be used to construct the Dixon matrix D.

2.2.3 Dixon Resultant

Definition 2.9. The Dixon resultant R of any generic polynomial system is the determinant of its corresponding Dixon matrix D.

We illustrate the construction of a Dixon matrix with the Heron $2d$ system.

Example 2.10 (Heron 2d)**.** Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3\} \subset \mathbb{Q}[Y][X]$ where

$$\hat{f}_1 = x_2^2 + x_3^2 - y_3^2$$
$$\hat{f}_2 = (x_2 - y_1)^2 + x_3^2 - y_2^2$$
$$\hat{f}_3 = -x_3 y_1 + 2x_1$$

with variables $X = \{x_1, x_2, x_3\}$ and parameters $Y = \{y_1, y_2, y_3\}$. Let $X_e = \{x_2, x_3\}$ be the variables to be eliminated and let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3\}$ be the new variables corresponding to X_e. Using (2.6), we construct the cancellation matrix

$$\hat{\mathcal{C}} = \begin{pmatrix} x_2^2 + x_3^2 - y_3^2 & (x_2 - y_1)^2 + x_3^2 - y_2^2 & -x_3y_1 + 2x_1 \\ x_2 + \bar{x}_2 & x_2 - 2y_1 + \bar{x}_2 & 0 \\ x_3 + \bar{x}_3 & x_3 + \bar{x}_3 & -y_1 \end{pmatrix},$$

and the Dixon polynomial

$$\begin{aligned} \Delta_{X_e} = (-2x_2y_1^2 + y_1^3 - y_1y_2^2 + y_1y_3^2)\bar{x}_2 + \left(-2x_3y_1^2 + 4x_1y_1\right)\bar{x}_3 \\ + \left(x_2y_1^3 - x_2y_1y_2^2 + x_2y_1y_3^2 - 2y_1^2y_3^2 + 4x_1x_3y_1\right). \end{aligned}$$

The Dixon polynomial Δ_{X_e} expressed in bilinear form yields

$$VD\overline{V} = \begin{bmatrix} x_2 & x_3 & 1 \end{bmatrix} \begin{pmatrix} -2y_1^2 & 0 & y_1^3 - y_1y_2^2 + y_1y_3^2 \\ 0 & -2y_1^2 & 4x_1y_1 \\ y_1^3 - y_1y_2^2 + y_1y_3^2 & 4x_1y_1 & -2y_1^2y_3^2 \end{pmatrix} \begin{bmatrix} \bar{x}_2 \\ \bar{x}_3 \\ 1 \end{bmatrix},$$

where D is the Dixon matrix (it is square and non-singular) and the Dixon resultant

$$R = \det(D) = 2y_1^4(16x_1^2 + y_1^4 - 2y_1^2y_2^2 - 2y_1^2y_3^2 + y_2^4 - 2y_2^2y_3^2 + y_3^4).$$

Remark 2.11. The parametric polynomial system $\mathcal{F}$ considered in Example 2.10 is not generic because for example, the coefficient of x_2^2 in $\hat{f}_1$ is 1, which is not an independent parameter according to our definition of generic systems on page 4. $\mathcal{F}$ is also not n-degree. In particular, notice that the maximum partial degree of $\mathcal{F}$ in x_2 and x_3 is 2, but the term $x_2^2x_3^2$ is missing. We were able to obtain the Dixon resultant for $\mathcal{F}$ in x_1 because the column of the Dixon matrix D corresponding to the monomial 1 in the row vector V is non-zero.

The Dixon resultant R may not give us the necessary condition for the existence of a solution (common root) to a polynomial system $\mathcal{F}$ if $\mathcal{F}$ is not generic $n-$degree because the linear homogeneous system formed when the Dixon polynomial is evaluated at roots of $\mathcal{F}$, and viewed in powers of products of $x_2, \ldots, x_n$, might not have a non-trivial solution despite the fact that $\mathcal{F}$ has a solution. This is because the constructed Dixon matrix might not contain the column corresponding to the monomial entry 1 present in the column vector of the formed linear homogeneous system (think of the Cayley-Bezout resultant formulation described in the introductory part of this thesis), and hence, if $\mathcal{F}$ only has a trivial solution then the linear homogeneous system only has a trivial solution which implies that the Dixon

resultant need not vanish at all zeros of $\mathcal{F}$. Furthermore, we might encounter a case where the Dixon matrix D has a column corresponding to the monomial 1. However, the Dixon matrix D could be singular, and in some cases, the Dixon matrix might even be rectangular [Kapur et al., 1994]. We will discuss how to address these shortcomings in Subsection 2.2.4.

Our next goal is to show that the Dixon resultant R is a polynomial in the elimination ideal $J = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle \cap \mathbb{Q}(Y)[x_1]$. We do this by exploiting some properties of the Dixon polynomial Δ_{X_e}.

Definition 2.12. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\}$ be a parametric polynomial system. Let us define

$$\Delta_{X_e,i} := \frac{\begin{vmatrix} \hat{f}_1 & \hat{f}_2 & \ldots & \hat{f}_{i-1} & 1 & \hat{f}_{i+1} & \ldots & \hat{f}_n \\ \pi_2(\hat{f}_1) & \pi_2(\hat{f}_2) & \ldots & \pi_2(\hat{f}_{i-1}) & 1 & \pi_2(\hat{f}_{i+1}) & \ldots & \pi_2(\hat{f}_n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\ \pi_n(\hat{f}_1) & \pi_n(\hat{f}_2) & \ldots & \pi_n(\hat{f}_{i-1}) & 1 & \pi_n(\hat{f}_{i+1}) & \ldots & \pi_n(\hat{f}_n) \end{vmatrix}}{\prod_{i=2}^{n}(x_i - \bar{x}_i)}. \tag{2.12}$$

That is, the Dixon polynomial $\Delta_{X_e} = \Delta_{X_e,i}$ if $\hat{f}_i = 1$ for any $i \in \{1, 2, \ldots, n\}$.

Using (2.6), we can write

$$\begin{aligned} \Delta_{X_e,i} &:= \begin{vmatrix} \hat{f}_1 & \hat{f}_2 & \ldots & \hat{f}_{i-1} & 1 & \hat{f}_{i+1} & \ldots & \hat{f}_n \\ \hat{\mathcal{C}}_{2,1} & \hat{\mathcal{C}}_{2,2} & \ldots & \hat{\mathcal{C}}_{2,i-1} & 0 & \hat{\mathcal{C}}_{2,i+1} & \ldots & \hat{\mathcal{C}}_{2,n} \\ \hat{\mathcal{C}}_{3,1} & \hat{\mathcal{C}}_{3,2} & \ldots & \hat{\mathcal{C}}_{3,i-1} & 0 & \hat{\mathcal{C}}_{3,i+1} & \ldots & \hat{\mathcal{C}}_{3,n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & & \\ \hat{\mathcal{C}}_{n,1} & \hat{\mathcal{C}}_{n,2} & \ldots & \hat{\mathcal{C}}_{n,i-1} & 0 & \hat{\mathcal{C}}_{n,i+1} & \ldots & \hat{\mathcal{C}}_{n,n} \end{vmatrix} \\ &= \begin{vmatrix} \hat{\mathcal{C}}_{2,1} & \ldots & \hat{\mathcal{C}}_{2,i-1} & \hat{\mathcal{C}}_{2,i+1} & \ldots & \hat{\mathcal{C}}_{2,n} \\ \hat{\mathcal{C}}_{3,1} & \ldots & \hat{\mathcal{C}}_{3,i-1} & \hat{\mathcal{C}}_{3,i+1} & \ldots & \hat{\mathcal{C}}_{3,n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \hat{\mathcal{C}}_{n,1} & \ldots & \hat{\mathcal{C}}_{n,i-1} & \hat{\mathcal{C}}_{n,i+1} & \ldots & \hat{\mathcal{C}}_{n,n} \end{vmatrix}. \end{aligned}$$

By expanding $\Delta_{X_e,i}$ and viewing it recursively in powers of $\{\bar{x}_2, \bar{x}_3, \cdots \bar{x}_n\}$, we can write

$$\Delta_{X_e,j} = \sum_{i_n=0}^{d_{\max,n}-1} \cdots \sum_{i_k=0}^{(n-k+1)d_{\max,k}-1} \cdots \sum_{i_2=0}^{(n-1)d_{\max,2}-1} H^j_{i_2,i_3,\ldots,i_n} \bar{x}_2^{i_2} \bar{x}_3^{i_3} \cdots \bar{x}_k^{i_k} \cdots \bar{x}_n^{i_n} \tag{2.13}$$

where the polynomials $H^j_{i_2,i_3,\ldots,i_n} \in \mathbb{Q}[Y][x_1, x_2, \ldots, x_n]$ for $1 \leq j \leq n$.

Theorem 2.13. Let $s = (n-1)! \prod_{i=2}^{n} d_{\max,i}$. Then

1. The Dixon polynomial Δ_{X_e} is an element of the ideal $I = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle$.

2. Let D be a square $s_1 \times s_1$ non-singular Dixon matrix where $s_1 \leq s$. The Dixon resultant $R = \det(D)$ is an element of the ideal $J = \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle \cap \mathbb{Q}(Y)[x_1]$.

To motivate the proof of Theorem 2.13, we revisit Example 2.10.

Example 2.14 (Example 2.10 revisited)**.** Recall

$$\begin{aligned}\hat{f}_1 &= x_2^2 + x_3^2 - y_3^2\\ \hat{f}_2 &= (x_2 - y_1)^2 + x_3^2 - y_2^2\\ \hat{f}_3 &= -x_3 y_1 + 2x_1\end{aligned}$$

- $X = \{x_1, x_2, x_3\}, Y = \{y_1, y_2, y_3\}, X_e = \{x_2, x_3\}, \overline{X}_e = \{\bar{x}_2, \bar{x}_3\}$.
- The cancellation matrix

$$\hat{\mathcal{C}} = \begin{pmatrix} x_2^2 + x_3^2 - y_3^2 & (x_2 - y_1)^2 + x_3^2 - y_2^2 & -x_3 y_1 + 2x_1 \\ x_2 + \bar{x}_2 & x_2 - 2y_1 + \bar{x}_2 & 0 \\ x_3 + \bar{x}_3 & x_3 + \bar{x}_3 & -y_1 \end{pmatrix}.$$

- The Dixon polynomial

$$\begin{aligned}\Delta_{X_e} = (-2x_2y_1^2 + y_1^3 - y_1y_2^2 + y_1y_3^2)\bar{x}_2 + \left(-2x_3y_1^2 + 4x_1y_1\right)\bar{x}_3\\ + \left(x_2y_1^3 - x_2y_1y_2^2 + x_2y_1y_3^2 - 2y_1^2y_3^2 + 4x_1x_3y_1\right).\end{aligned}$$

- The Dixon polynomial Δ_{X_e} expressed in bilinear form yields

$$VD\overline{V} = \begin{bmatrix} x_2 & x_3 & 1 \end{bmatrix} \begin{pmatrix} -2y_1^2 & 0 & y_1^3 - y_1y_2^2 + y_1y_3^2 \\ 0 & -2y_1^2 & 4x_1y_1 \\ y_1^3 - y_1y_2^2 + y_1y_3^2 & 4x_1y_1 & -2y_1^2y_3^2 \end{pmatrix} \begin{bmatrix} \bar{x}_2 \\ \bar{x}_3 \\ 1 \end{bmatrix},$$

where D is the Dixon matrix. The Dixon resultant

$$R = \det(D) = 2y_1^4(16x_1^2 + y_1^4 - 2y_1^2y_2^2 - 2y_1^2y_3^2 + y_2^4 - 2y_2^2y_3^2 + y_3^4).$$

Let $Q_{1,0}$ $Q_{0,1}$ be the polynomial coefficients of Δ_{X_e} in variables $\bar{x_2}$, and $\bar{x}_3$ respectively and let $Q_{0,0}$ be the constant polynomial term. So,

$$\begin{aligned}Q_{0,0} &= x_2y_1^3 - x_2y_1y_2^2 + x_2y_1y_3^2 - 2y_1^2y_3^2 + 4x_1x_3y_1\\ Q_{1,0} &= -2x_2y_1^2 + y_1^3 - y_1y_2^2 + y_1y_3^2\\ Q_{0,1} &= -2x_3y_1^2 + 4x_1y_1.\end{aligned}$$

Using the cancellation matrix $\hat{\mathcal{C}}$, observe that

$$\Delta_{X_e,1} = \begin{vmatrix} x_2 - 2y_1 + \bar{x}_2 & 0 \\ x_3 + \bar{x}_3 & -y_1 \end{vmatrix} = \underbrace{\left(-x_2y_1 + 2y_1^2\right)}_{H^1_{0,0}} + \underbrace{(-y_1)}_{H^1_{1,0}}\bar{x}_2$$

$$\Delta_{X_e,2} = \begin{vmatrix} x_2 + \bar{x}_2 & 0 \\ x_3 + \bar{x}_3 & -y_1 \end{vmatrix} = \underbrace{(-x_2y_1)}_{H^2_{0,0}} + \underbrace{(-y_1)}_{H^2_{1,0}}\bar{x}_2$$

$$\Delta_{X_e,3} = \begin{vmatrix} x_2 + \bar{x}_2 & x_2 - 2y_1 + \bar{x}_2 \\ x_3 + \bar{x}_3 & x_3 + \bar{x}_3 \end{vmatrix} = \underbrace{(2x_3y_1)}_{H^3_{0,0}} + \underbrace{(2y_1)}_{H^3_{0,1}}\bar{x}_3.$$

Thus,

$$\Delta_{X_e} = \hat{f}_1\Delta_{X_e,1} - \hat{f}_2\Delta_{X_e,2} + \hat{f}_3\Delta_{X_e,3}.$$

Let Q be a row vector such that

$$Q = [Q_1, Q_2, Q_3] = [Q_{1,0}, Q_{0,1}, Q_{0,0}]\,.$$

Observe that $Q = VD$. Also,

$$\begin{aligned} Q_{1,0} &= \hat{f}_1H^1_{1,0} - \hat{f}_2H^2_{1,0} + \hat{f}_3(0) \\ Q_{0,1} &= \hat{f}_1(0) - \hat{f}_2(0) + \hat{f}_3H^3_{0,1} \\ Q_{0,0} &= \hat{f}_1H^1_{0,0} - \hat{f}_2H^2_{0,0} + \hat{f}_3H^3_{0,0}. \end{aligned}$$

The adjoint matrix of the Dixon matrix D is

$$E = \begin{bmatrix} -4y_1^2\left(-y_1y_3 + 2x_1\right)\left(y_1y_3 + 2x_1\right) & 4x_1\,y_1^2\left(y_1^2 - y_2^2 + y_3^2\right) & 2y_1^3\left(y_1^2 - y_2^2 + y_3^2\right) \\ 4x_1\,y_1^2\left(y_1^2 - y_2^2 + y_3^2\right) & E_{2,2} & 8y_1^3x_1 \\ 2y_1^3\left(y_1^2 - y_2^2 + y_3^2\right) & 8y_1^3x_1 & 4y_1^4 \end{bmatrix}$$

where

$$E_{2,2} = -y_1^2\left(y_1 - y_2 + y_3\right)\left(y_1 + y_2 + y_3\right)\left(y_1 - y_2 - y_3\right)\left(y_1 + y_2 - y_3\right).$$

Finally, one can easily check that

$$R = QE_3 = Q_{1,0}E_{3,1} + Q_{0,1}E_{3,2} + Q_{0,0}E_{3,3}$$

where $E_{3,i}$ are the entries of the the third column E_3 of the adjoint matrix E.

We are now ready to give the proof of Theorem 2.13.

Proof of Theorem 2.13. We begin with claim (1). Recall

$$\Delta_{X_e} = \begin{vmatrix} \hat{f}_1 & \hat{f}_2 & \dots & \hat{f}_n \\ \hat{\mathcal{C}}_{2,1} & \hat{\mathcal{C}}_{2,2} & \dots & \hat{\mathcal{C}}_{2,n} \\ \hat{\mathcal{C}}_{3,1} & \hat{\mathcal{C}}_{3,2} & \dots & \hat{\mathcal{C}}_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{\mathcal{C}}_{n,1} & \hat{\mathcal{C}}_{n,2} & \dots & \hat{\mathcal{C}}_{n,n} \end{vmatrix}.$$

Expanding the above determinant along its first row and using (2.12) yields

$$\Delta_{X_e} = \sum_{j=1}^{n} (-1)^{1+j} \hat{f}_j \Delta_{X_e,j}. \tag{2.14}$$

Thus, the proof of claim (1) is complete. Now we prove claim (2). For claim (2), it suffices to only show that $R \in \left\langle \hat{f}_1, \hat{f}_2, \dots, \hat{f}_n \right\rangle$ since $R \in \mathbb{Q}(Y)[x_1]$ by definition. By substituting (2.13) into (2.14), we obtain

$$\Delta_{X_e} = \sum_{i_n=0}^{d_{\max,n}-1} \cdots \sum_{i_k=0}^{(n-k+1)d_{\max,k}-1} \cdots \sum_{i_2=0}^{(n-1)d_{\max,2}-1} Q_{i_2,i_3,\dots,i_n} \, \bar{x}_2^{i_2} \bar{x}_3^{i_3} \cdots \bar{x}_k^{i_k} \cdots \bar{x}_n^{i_n} \tag{2.15}$$

where

$$Q_{i_2,i_3,\dots,i_n} = \sum_{j=1}^{n} (-1)^{1+j} \hat{f}_j H^j_{i_2,i_3,\dots,i_n} \in \mathbb{Q}[Y][x_1, x_2, \dots, x_n]. \tag{2.16}$$

By viewing Δ_{X_e} in (2.15) as a polynomial in $\overline{X}_e$, we can write

$$\Delta_{X_e} = Q\overline{V} \tag{2.17}$$

where the entries of the $1 \times s_1$ row vector Q are the polynomials $Q_{i_2,i_3,\dots,i_n}$, and $\overline{V}$ is a $s_1 \times 1$ monomial column vector in $\overline{X}_e = \{\bar{x}_2, \bar{x}_3, \dots, \bar{x}_n\}$. For convenience, we assume $s_1 = s$ and write

$$Q = [Q_1, Q_2, \dots, Q_{s_1}] = [Q_{0,0,\dots,0}, Q_{0,0,\dots,1}, \dots, Q_{(n-1)d_{\max,2}-1,\dots,d_{\max,n}-1}].$$

Thus, by comparing (2.11) which says

$$\Delta_{X_e} = VD\overline{V},$$

and (2.17), it follows that

$$Q = VD.$$

Since D is a non-singular square matrix, it follows the row vector V has a monomial 1 and the column vector corresponding to it in D is non-zero [Kapur et al., 1994]. Otherwise, D

would be singular. Therefore, we have

$$QD^{-1} = V.$$

Let E denote the adjoint matrix of D. So,

$$QE = \det(D)V = RV$$

where $R = \det(D)$. Notice that one of the entries of the row vector RV is R and it is equal to one of the entries of the row vector QE. Thus, our claim follows since entries of the row vector QE belong to $\left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle$ and $Q_k \in \left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle$ by (2.16).

□

2.2.4 Extracting a maximal rank sub-matrix M from a Dixon Matrix D

As discussed in the introduction section of this thesis, the Dixon resultant method has shortcomings. The shortcomings include the singularity of a square Dixon matrix which provides no information about the solutions of the polynomial system, and the problem of computing the determinant of a rectangular Dixon matrix. We will construct the Dixon matrix for the Heron 3d system from Example 2.5 to illustrate one of the shortcomings.

Example 2.15 (Heron 3d system)**.** Constructing the Dixon matrix for $\mathcal{F}$ from the Dixon polynomial given in Example 2.5, we obtain the following rectangular 16×14 Dixon matrix

$$D = \begin{pmatrix}
0 & 0 & 0 & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C \\
0 & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C & 0 & 0 \\
0 & 0 & 0 & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C \\
0 & 0 & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C & 0 \\
0 & 0 & 0 & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C \\
0 & 0 & -A & 0 & 0 & 0 & 0 & A & 0 & 0 & -B & -C & 0 & 0 \\
0 & -A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C & 0 & 0 & 0 \\
-A & 0 & 0 & 0 & 0 & A & -B & 0 & C & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & B & 0 & 0 & 0 & A & 0 & 0 & 0 & 0 & F \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A & 0 & 0 & 0 & G \\
0 & 0 & B & 0 & 0 & 0 & 0 & -B & 0 & 0 & E & F & G & 0 \\
0 & 0 & 0 & 0 & -C & 0 & -A & 0 & 0 & 0 & 0 & 0 & 0 & K \\
0 & B & -C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & F & K & 0 & 0 \\
0 & 0 & 0 & 0 & C & 0 & 0 & -A & 0 & 0 & 0 & 0 & 0 & K \\
0 & 0 & 0 & C & 0 & 0 & 0 & 0 & 0 & 0 & G & 0 & K & 0 \\
B & 0 & 0 & 0 & 0 & 0 & -B & E & 0 & G & 0 & 0 & 0 & 0
\end{pmatrix},$$

where $\text{rank}(D) = 13$ and

$$A = 8y_1^3, \qquad B = 4y_1^4 - 4y_1^2y_4^2 + 4y_1^2y_6^2$$
$$C = -4y_1^4 + 4y_1^2y_2^2 - 4y_1^2y_3^2, \qquad K = 8y_1^3y_3^2$$
$$E = 8y_1^3y_6^2, \qquad F = -4y_1^3y_3^2 + 4y_1^3y_5^2 - 4y_1^3y_6^2$$
$$G = -48y_1^2x_1.$$

These shortcomings were addressed by Kapur, Saxena and Yang in [Kapur et al., 1994]. They proved that the determinant of any sub-matrix of the Dixon matrix D of maximal rank is an element of the elimination ideal $I \cap \mathbb{Q}(Y)[x_1]$. So, once a Dixon matrix D is constructed, the third major step is to identify a sub-matrix of maximal rank, and the final major step is to compute $R = \det(M)$. Hence, the requirement for $\mathcal{F}$ to be generic n-degree is no longer necessary.

Our new approach to select a sub-matrix M of a Dixon matrix D such that $\text{rank}(M) = \text{rank}(D)$ proceeds as follows. For simplicity, suppose D is a $t \times t$ matrix, and let $D_{\max} = \max_{j=1}^{t} \deg(D_{ij})$. We pick a random 62 bit prime p and choose an evaluation point $\beta \in \mathbb{Z}_p^{m+1}$ uniformly at random. Then we compute $B = D(\beta)$ and identify a sub-matrix of maximal rank from B in D with high probability. Our approach requires Gaussian elimination over $\mathbb{Z}_p$ only and in contrast to [Kapur et al., 1994] crucially avoids doing polynomial arithmetic in $\mathbb{Q}[Y, x_1]$.

We also remark that our algorithm (Algorithm 5) for extracting a sub-matrix of maximal rank does not explicitly switch rows. Instead, we use an ordered list to store the row indices of B, which is updated each time row operations are performed (See Lines 19-20 of Algorithm 5). This was done to improve the overall performance of our code.

Example 2.16. Consider the 4×5 rectangular Dixon matrix

$$D = \begin{pmatrix} 0 & y_1^2 + 2x_1 - y_2 & 0 & 0 & y_1^2 + x_1 - y_2 \\ 0 & 0 & 0 & x_1y_3y_2 + y_1^2 + y_3 & 0 \\ x_1y_3y_2 & 0 & 0 & 0 & 0 \\ 0 & y_1^2 + x_1 - y_2 & y_3 & 0 & 0 \end{pmatrix}.$$

Using Algorithm 5, we obtain a maximal rank sub-matrix

$$M = \begin{pmatrix} x_1y_3y_2 & 0 & 0 & 0 \\ 0 & y_1^2 + 2x_1 - y_2 & 0 & 0 \\ 0 & y_1^2 + x_1 - y_2 & y_3 & 0 \\ 0 & 0 & 0 & x_1y_3y_2 + y_1^2 + y_3 \end{pmatrix}$$

such that $\text{rank}(M) = \text{rank}(D) = 4$.

Algorithm 5: ExtractMinor

Input: A rectangular matrix D where $D_{i,j} \in \mathbb{Q}[x_1, y_1, \ldots, y_m]$.

Output: A sub-matrix M of D such that $\text{rank}(M) = \text{rank}(D)$.

Let n_r and n_c be the row and column dimension of D respectively.
Pick a 62 bit prime p and a random evaluation point $\beta \in \mathbb{Z}_p^{m+1}$
$B \leftarrow D(\beta)$ // B is the integer matrix over $\mathbb{Z}_p$ obtained by evaluating D at $x_1 = \beta_1, y_1 = \beta_2, \ldots, y_m = \beta_{m+1}$.
Initialize $T = [1, 2, \ldots, n_r]$ and $(P, C) \leftarrow ([\ \ ], [\ \ ])$ // T, P, C are ordered lists.
$\text{rank}(B) \leftarrow 0$
$(r, c) \leftarrow (1, 0)$
while $k \leq n_r$ and $c < n_c$ **do**
    $c \leftarrow c + 1$ // Next column
    Search for a non-zero pivot:
    $E \leftarrow 0$
    while $E = 0$ **do**
        for i from k to n_r do **do**
            if $B_{T_i,c} \neq 0$ **then**
                $E \leftarrow B_{T_i,c}$ //A pivot is found in column c of B.
                break;
            end
        end
        if $i = n_r$ and $E = 0$ then $c \leftarrow c + 1$ **end** // A non-zero pivot is not found, so we try the next column.
    end
    Swap rows and update the pivot columns and rows :
    $(T_k, T_i) \leftarrow (T_i, T_k), \quad \text{rank}(B) \leftarrow \text{rank}(B) + 1.$
    $(P, C) \leftarrow (P \cup T_k, C \leftarrow C \cup c)$
    Row operations :
    for i from $k + 1$ to r **do**
        for $j = c + 1$ *to* n_c **do**
            $B_{T_i,j} \leftarrow B_{T_i,j} - \frac{B_{T_i,c}}{B_{T_k,c}} B_{T_k,j} \bmod p$
        end
        $B_{T_i,c} \leftarrow 0$
    end
    $k \leftarrow k + 1$ // Next row
end
Let M be a $r \times r$ zero matrix where $r = \text{rank}(D)$.
for i from 1 to $\text{rank}(B)$ **do**
    for j from 1 to $\text{rank}(B)$ **do**
        $M_{i,j} \leftarrow D_{P_i,C_j}$
    end
end
return M

2.2.5 The Failure Probability of Algorithm 5

Here, our main goal is to give a failure probability bound for Algorithm 5. In order to do this, we first need the Schwartz-Zippel Lemma [Schwartz, 1980, Zippel, 1979].

Lemma 2.17 (Schwartz-Zippel Lemma)**.** Let $\mathbb{K}$ be a field and let $f \neq 0 \in \mathbb{K}[y_1, y_2, \ldots, y_m]$. Let F be a finite subset of $\mathbb{K}$. If α is chosen uniformly at random from F^m then

$$\Pr[f(\alpha) = 0] \leq \frac{\deg(f)}{|F|}.$$

We motivate the failure probability analysis for Algorithm 5 using the following example.

Example 2.18. Consider the 4×5 rectangular matrix

$$D = \begin{pmatrix} x_1^2 y_1 & 1 & 0 & 0 & 0 \\ 2x_1 y_1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 5 & x_1^2 & 0 \\ 7 & 8 & 10 & 4x_1 & 0 \end{pmatrix}.$$

Performing fraction free Gaussian elimination on D yields

$$J = \begin{pmatrix} x_1^2 y_1 & 1 & 0 & 0 & 0 \\ 0 & 2x_1^2 y_1 - 2x_1 y_1 & 0 & 0 & 0 \\ 0 & 0 & 10x_1^2 y_1 - 10x_1 y_1 & 2x_1^4 y_1 - 2x_1^3 y_1 & 0 \\ 0 & 0 & 0 & -20x_1^4 y_1 + 60x_1^3 y_1 - 40x_1^2 y_1 & 0 \end{pmatrix}.$$

Using the pivots from J, a correct sub-matrix M of D such that $\text{rank}(M) = \text{rank}(D)$ is

$$M = \begin{pmatrix} x_1^2 y_1 & 1 & 0 & 0 \\ 2x_1 y_1 & 2 & 0 & 0 \\ 1 & 0 & 5 & x_1^2 \\ 7 & 8 & 10 & 4x_1 \end{pmatrix}.$$

However, Algorithm ExtractMinor (Algorithm 5) may return an incorrect sub-matrix

$$M_c = \begin{pmatrix} x_1^2 y_1 & 1 & 0 \\ 2x_1 y_1 & 2 & 0 \\ 1 & 0 & 5 \end{pmatrix}$$

with $\text{rank}(M_c) = 3 < \text{rank}(D) = \text{rank}(M) = 4$, if the input random prime $p = 2^{62} - 57$, and the evaluation point β selected at random in Line 2 of Algorithm 5 is $\beta = (2, 5) \in \mathbb{Z}_p^2$. Therefore, it is important that we give a failure probability bound for Algorithm 5.

Theorem 2.19. Let D be a $s \times s$ Dixon matrix of polynomials in $\mathbb{Z}[Y][x_1]$ and let $D_{\max} = \max_{i,j=1}^{s} \deg(D_{ij})$. Let p be a prime such that $\text{rank}(D) = \text{rank}(D \bmod p)$. Let β be an evaluation point chosen at random from $\mathbb{Z}_p^{m+1}$ and let $B = D(\beta)$. Then

$$\Pr[\text{rank}(B) < \text{rank}(D)] \leq \frac{sD_{\max}}{p}.$$

Proof. Let $r = \text{rank}(D)$ and let M be a $r \times r$ sub-matrix of D such that $\text{rank}(M) = r$, which can be found using fraction free Gaussian elimination in $\mathbb{Q}[Y, x_1]$. Since M is of full rank, it follows that $\det(M) \neq 0$. Clearly,

$$\deg(\det(M)) = rD_{\max} \leq sD_{\max}.$$

In Algorithm ExtractMinor (Algorithm 5), we have to compute $B = D(\beta)$, and then perform row operations on B over $\mathbb{Z}_p$ using ordinary Gaussian elimination to extract a sub-matrix of maximal rank. Thus, Algorithm 5 will return an incorrect sub-matrix of maximal rank if

$$\text{rank}(B) < r \Longrightarrow \det(M(\beta)) = 0.$$

By Lemma 2.17, it follows that

$$\Pr[\text{rank}(B) < \text{rank}(D)] \leq \Pr[\det(M(\beta)) = 0] \leq \frac{\deg(M)}{p} \leq \frac{sD_{\max}}{p}.$$

□

Example 2.20 (Example 2.18 revisited)**.** In Example 2.18, notice that $D_{\max} = 3$ and assume that $s = 5$. Since $p = 2^{62} - 57$, we have that

$$\Pr[\text{rank}(B) < \text{rank}(D)] \leq \frac{3 \times 5}{p} \leq 3.25 \times 10^{-18}.$$

In Theorem 2.19, we assumed that $\text{rank}(D) = \text{rank}(D \bmod p)$ in order to get a failure probability bound. This is because p can also cause Algorithm 5 to return an incorrect sub-matrix of maximal rank. Thus, it is important to bound $\Pr[\text{rank}(D \bmod p) < \text{rank}(D)]$.

To do this, suppose Algorithm 5 selects a random prime p from a list of primes $P = \{p_1, p_2, \cdots, p_N\}$ and $p_{\min} = \min(P)$. Then

$$\Pr[\text{rank}(D \bmod p) < \text{rank}(D)] \leq \frac{\log_{p_{\min}} \|\det(M)\|_\infty}{N}$$

where $\|\det(M)\|_\infty$ denotes the largest integer coefficient of the determinant of a sub-matrix M of maximal rank. We did not include this failure probability bound in Theorem 2.19 because we do not have the tools to bound $\|\det(M)\|_\infty$ now. A bound for $\|\det(M)\|_\infty$ is provided in Theorem 6.5(iv) of Chapter 6.

Chapter 3

Sparse Interpolation Tools

3.1 Summary of Contributions

The materials in this chapter are background materials. However, we have made a hybrid Maple implementation of Zippel's sparse interpolation algorithm to interpolate the Dixon resultant R in expanded form (see Subsection 5.4.3 for benchmarks). We have also implemented the Maximal Quotient Rational Function Reconstruction algorithm (Algorithm 8) modulo a prime for performing dense rational function interpolation in one variable in C.

3.2 Sparse Polynomial Interpolation

In this section, we review two sparse polynomial interpolation algorithms, namely, Zippel's sparse interpolation algorithm and the Ben-Or/Tiwari algorithm. Both algorithms can be used to interpolate the Dixon resultant R. We will modify the Ben-Or/Tiwari algorithm to serve as the primary sparse polynomial algorithm in our proposed Dixon resultant algorithm and in the new black box algorithm for solving parametric linear systems. For an extensive bibliography on sparse polynomial interpolation, we advise the reader to see [Roche, 2018].

3.2.1 Zippel's sparse interpolation

Zippel's sparse interpolation algorithm was developed in 1979 to originally compute GCD problems in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$. It is the primary built-in algorithm in many computer algebra systems for performing multivariate polynomial GCD computation. Zippel's algorithm is a probabilistic algorithm which interpolates the desired sparse multivariate polynomial one variable at a time. The interpolation of the target polynomial f can be done recursively, and at the base of the recursion for one variable, we use dense polynomial interpolation [von zur Gathen and Gerhard, 2013, Section 5.2, page 101]. We demonstrate how the algorithm works with the following example.

Example 3.1. Let prime $p = 3137$ and suppose

$$f = 2y_1^2y_2^2 + y_3^2 + 5 \in \mathbb{Z}_p[y_1, y_2, y_3]$$

is represented by a black box that outputs $f(\alpha)$ mod p when we input an evaluation point $\alpha \in \mathbb{Z}_p^3$ and a prime p to it. Let us assume that the partial degrees of f in y_1, y_2, y_3 denoted by $d_{y_1}, d_{y_2}, d_{y_3}$ are given. So, we have $d_{y_1} = d_{y_2} = d_{y_3} = 2$. We interpolate f as follows.

Using $d_{y_1} + 1 = 3$ probes to the black box, we begin by picking two random points say $\beta_1 = 2, \theta_1 = 3$ in order to densely interpolate $f(y_1, \beta_1, \theta_1)$. Now we pick $\alpha_i \in \mathbb{Z}_p$ for $1 = 1, 2, \ldots, d_{y_1}$ at random, and compute

$$z_i = f(\alpha_i, \beta_1, \theta_1)$$

for $i = 1, 2, 3$ to interpolate y_1. For simplicity, suppose $\alpha_i = i$ for $i = 1, 2, 3$. Then

$$z_1 = 22, z_2 = 46, z_3 = 86.$$

By performing dense interpolation using Newton's algorithm or Lagrange algorithm [von zur Gathen and Gerhard, 2013], one finds

$$f(y_1, 2, 3) = 8y_1^2 + 14.$$

Now we assume that if we interpolate $f(y_1, \beta_2, \theta_2)$, it will have the same support as $f(y_1, \beta_1, \theta_1)$. That is, we assume $f(y_1, y_2, y_3)$ has no linear term in y_1. This assumption is called the sparse assumption. So with high probability, the skeleton polynomial (the current structure of f) will be

$$G_{y_1} = A_2(y_2, y_3)y_1^2 + A_0(y_2, y_3)$$

where $A_2(y_2, y_3), A_0(y_2, y_3) \in \mathbb{Z}_p[y_2, y_3]$. Thus, $f(y_1, \beta_2, \theta_1)$ can be viewed as

$$f(y_1, \beta_2, \theta_1) = A_2(\beta_2, \theta_1)y_1^2 + A_0(\beta_2, \theta_1),$$

so we only need to solve for A_0 and A_2. Suppose $\beta_2 = 3$, and the two points $3, 4$ are selected at random for y_1. We form the system of linear equations:

$$\begin{aligned} 9A_2 + A_0 &= f(3, 3, 3) = 176 \\ 16A_2 + A_0 &= f(4, 3, 3) = 302 \end{aligned}$$

where $\theta_1 = 3$. Solving the above system yields $\{A_2 = 18, A_0 = 14\}$, so

$$f(y_1, 3, 3) = f(y_1, \beta_2, \theta_1) = 18y_1^2 + 14.$$

Notice that we used two probes on y_1, whereas densely interpolating $f(y_1, 3, 3)$ would require three probes. Since the degree $d_{y_2} = 2$, we need one more image to interpolate the polynomial coefficients in y_2. Let $\beta_3 = 4, \theta_1 = 3$, and let the random points selected for y_1 be $5, 6$. Repeating the same process as before, we obtain the following system of linear equations

$$25A_2 + A_0 = f(5, 4, 3) = 814$$
$$36A_2 + A_0 = f(6, 4, 3) = 1166$$

which yields $\{A_2 = 32, A_0 = 14\}$. Hence,

$$f(y_1, 4, 3) = 32y_1^2 + 14.$$

We now proceed to interpolate the coefficients in y_2. Since all the constant terms of $f(y_1, \beta_i, 3)$ for $1 \leq i \leq 3$ are all 14, no interpolation is needed since the same value 14 will be returned if any interpolation is done. Now, using the points $(2, 8), (3, 18), (4, 32)$ whose x components are the β_i's, and the y components are the leading coefficients of $f(y_1, \beta_i, 3)$, we densely interpolate the polynomial $2y_2^2 \in \mathbb{Z}_p[y_2]$. So,

$$f(y_1, y_2, 3) = 2y_2^2y_1^2 + 14.$$

Thus, the updated skeleton polynomial now becomes

$$G_{y_1,y_2} = B_2(y_3)y_2^2y_1^2 + B_0(y_3)$$

where $B_2(y_3), B_0(y_3) \in \mathbb{Z}_p[y_3]$. In order to solve for B_0, B_2, we only need two evaluation points to set up a system of linear equations. Let $\theta_1 = 4, \alpha_1 = 8, \alpha_2 = 9$ and let $\gamma_1, = 6, \gamma_2 = 7$. We set up a linear system by equating

$$f(\alpha_i, \gamma_i, \theta_1) = G_{y_1,y_2}(\alpha_i, \gamma_i, \theta_1)$$

which yields:

$$900B_2 + B_0 = f(8, 6, 4) = 1821$$
$$832B_2 + B_0 = f(9, 7, 4) = 412.$$

Solving the above systems yields $\{B_2 = 2, B_0 = 21\}$, so

$$f(y_1, y_2, 4) = 2y_1^2y_2^2 + 21.$$

Since $d_{y_3} = 2$, we need 2 more images to interpolate the polynomial coefficients of y_3. Repeating the interpolating process as before for $\theta_2 = 5, \alpha_1 = 9, \alpha_2 = 10$ and letting $\gamma_1, = 8, \gamma_2 = 9$, we get

$$f(y_1, y_2, 5) = 2y_1^2 y_2^2 + 30.$$

We now have enough images to interpolate y_3. Notice that the leading coefficients of all the images $f(y_1, y_2, \theta_i)$ are 2, so no interpolation is needed. However, interpolation has to be performed using the non-identical constant coefficients. Using the points, $(3, 14), (4, 21), (5, 30)$, we finally obtain $y_3^2 + 5$. Hence, $f = 2y_1^2 y_2^2 + y_3^2 + 5$ and we are done.

Remark 3.2. The polynomial in the above example was interpolated using 11 black box probes. However, if dense interpolation were performed, it would require a total of 27 probes to the black box. Also, note that in the process of trying to get more images at the sparse interpolation step, it is possible that the coefficient matrix of the linear system formed is singular since we use random evaluation points. This issue can be fixed by using more random evaluation points to add more equations until the determinant of the coefficient matrix of the linear system is not zero.

The number of probes required by Zippel's algorithm is derived as follows. Let $f \in \mathbb{Z}_p[y_1, y_2, \ldots, x_n]$ where p is prime. Let d_{x_i} denote the partial degrees of f in each variable x_i, and let t_j be an upper bound on the number of non-zero terms in f after it is evaluated at $x_{j+1} = \alpha_{j+1}, \ldots, x_n = \alpha_n$. At the base of the recursion, we need $d_{y_1} + 1$ evaluation points to perform dense interpolation in y_1. This yields the first skeleton polynomial in y_1. Thus only d_{y_2} images are needed to interpolate y_2. For each of the d_{y_2} images, only t_1 probes are needed. Repeating the process, it follows that the total number of black box probes is

$$O(d_{y_1} + d_{y_2} t_1 + d_{y_3} t_2 + \cdots d_{x_n} t_{n-1}).$$

Zippel's sparse interpolation requires $O(Dt)$ black probes to interpolate f where $D = \sum_{i=1}^{n} d_i$.

Remark 3.3. Let n be the number of variables in the polynomial f to be interpolated. It might not be obvious from Example 3.1 that the gain of Zippel's method over dense interpolation is exponential in n. To see this, if a polynomial f with $d = 10, n = 10$ and $t = 10$ is to be interpolated, then using dense interpolation requires 11^{10} points, which means a gain of a factor of $\geq 10^7$ is realized if Zippel's method is used.

3.2.2 Ben-Or/Tiwari Interpolation

The Ben-Or/Tiwari sparse interpolation algorithm is a deterministic algorithm (not randomized) for interpolating sparse polynomials, and unlike, Zippel's algorithm, it does not

interpolate the target polynomial one variable at a time. The algorithm requires that a term bound $T \geq t$ be known where t is the number of terms in the polynomial f to be interpolated. It does $2T$ probes to the black box in order to interpolate f. The Ben-Or/Tiwari algorithm is as follows. Consider the black box representation of

$$f = \sum_{i=1}^{t} a_i M_i(y_1, y_2, \ldots, y_n) \in \mathbb{Z}[y_1, y_2, \ldots, y_n]$$

where the integer coefficients $a_i \neq 0$, the monomials

$$M_i = \prod_{j=1}^{n} y_j^{e_{i,j}}$$

and $e_{i,j}$ denotes the exponent of y_j in M_i. The Ben-Or/Tiwari algorithm uses $2T$ prime point sequence of the form $\{(2^j, 3^j, \ldots, p_n^j) : 0 \leq j < 2T\}$ to interpolate f, where p_n denotes the n-th prime. Let $m_i = M_i(2, 3, \ldots, p_n)$ be a monomial evaluation. Then the black box evaluations

$$v_j = f(2^j, 3^j, 5^j, \ldots, p_n^j) = \sum_{i=1}^{t} a_i (2^j)^{e_{i,1}} (3^j)^{e_{i,2}} \cdots (p_n^j)^{e_{i,n}} = \sum_{i=1}^{t} a_i m_i^j.$$

We have that $m_k \neq m_j \iff k \neq j$ since m_k and m_j have different factorizations.

The Ben-Or/Tiwari interpolation algorithm can be divided into two main steps, namely, first finding the value of t and the monomial evaluations m_i, then solving for the a_i coefficients. Once the monomial evaluations m_i are known, the monomials M_i are obtained by performing repeated trial divisions by the successive primes $p_1 = 2, p_2 = 3, \cdots, p_n$ to recover the exponents in the M_i's. For the second step, after we have determined m_i and t, we set up the transposed Vandermonde system:

$$Va = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ m_1 & m_2 & \cdots & m_t \\ \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \cdots & m_t^{t-1} \end{pmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v \tag{3.1}$$

where the transpose matrix of V, denoted by V^T is a Vandermonde matrix and the determinant of V denoted by $\det(V) = \prod_{1 \leq i < j \leq t}(m_i - m_j)$. Since the monomial evaluations are distinct ($m_i \neq m_j$), it follows that V^{-1} exists which implies that (3.1) has a unique solution.

Using transposed Vandermonde systems to solve for the coefficients was actually proposed by Zippel in [Zippel, 1990] where he discussed how to choose powers of random evaluation points of the form α^j in the sparse interpolation step of his algorithm when solving for the unknown coefficients. Zippel also shows how to solve the transposed Vandermonde

system using $O(t^2)$ arithmetic operations and $O(t)$ space. This is a huge improvement because solving a system of linear equations, say with t unknown coefficients using classical algorithms such as Gaussian elimination costs $O(t^3)$ arithmetic operations and uses $O(t^2)$ space. For the sake of brevity, we will not present the details involved in solving for the coefficients a_i using Zippel's quadratic algorithm. However, a modified version of Zippel's quadratic algorithm will be presented in Chapter 4.

Now we discuss how to find the m_i and t, which means we find the exponents $e_{i,j}$'s. As we have discussed earlier, this is the first major step of the Ben-Or/Tiwari algorithm. Ben-Or/Tiwari's idea to determine the exponents begins with a linear generator $\lambda(z)$, the monic univariate polynomial

$$\lambda(z) = \prod_{i=1}^{t}(z - m_i) = z^t + \lambda_{t-1}z^{t-1} + \cdots + \lambda_1 z + \lambda_0$$

whose roots are the monomial evaluations m_i where $\lambda_i \in \mathbb{Z}$.

To determine the t coefficients $\lambda_0, \lambda_1, \ldots, \lambda_{t-1}$ of the linear generator $\lambda(z)$, we first construct and solve an associated linear system involving the coefficients λ_i as follows. Let $L \geq 0$. Now, consider the sum

$$\begin{aligned}
\sum_{i=1}^{t} a_i m_i^L \lambda(m_i) &= \sum_{i=1}^{t} a_i m_i^L \left(\sum_{j=0}^{t} \lambda_j m_i^j\right) \\
&= \sum_{i=1}^{t}\sum_{j=0}^{t} a_i \lambda_j m_i^{L+j} = \sum_{j=0}^{t} \lambda_j \left(\sum_{i=1}^{t} a_i m_i^{L+j}\right) \\
&= \lambda_0 \left(\sum_{i=1}^{t} a_i m_i^L\right) + \lambda_1 \left(\sum_{i=1}^{t} a_i m_i^{L+1}\right) + \cdots + \lambda_{t-1} \left(\sum_{i=1}^{t} a_i m_i^{L+t-1}\right) \\
&+ \lambda_t \left(\sum_{i=1}^{t} a_i m_i^{L+t}\right).
\end{aligned}$$

Observe that

$$\sum_{i=1}^{t} a_i m_i^L \lambda(m_i) = \sum_{i=1}^{t} a_i m_i^L \prod_{i=1}^{t}(m_i - m_i) = 0.$$

Hence,

$$\lambda_0 \left(\sum_{i=1}^{t} a_i m_i^L\right) + \lambda_1 \left(\sum_{i=1}^{t} a_i m_i^{L+1}\right) + \cdots \lambda_{t-1} \left(\sum_{i=1}^{t} a_i m_i^{L+t-1}\right) + \left(\sum_{i=1}^{t} a_i m_i^{L+t}\right) = 0.$$

Let $v_{L+k} = \sum_{i=1}^{t} a_i m_i^{L+k}$. Then it follows that

$$\lambda_0 v_L + \lambda_1 v_{L+1} + \lambda_1 v_{L+1} + \cdots \lambda_{t-1} v_{L+t-1} + v_{L+t} = 0 \text{ for } L \geq 0,$$

which gives $\lambda_0 v_L + \lambda_1 v_{L+1} + \lambda_1 v_{L+1} + \cdots \lambda_{t-1} v_{L+t-1} = -v_{L+t}$ for $L \geq 0$. Since we need to solve for t coefficients (the λ_i's), we take $L = 0, 1, 2, \ldots, t-1$ to form the system of linear equations in λ_i :

$$H_t\lambda = \begin{pmatrix} v_0 & v_1 & \cdots & v_{t-1} \\ v_1 & v_2 & \cdots & v_t \\ \vdots & \vdots & \vdots & \vdots \\ v_{t-1} & v_t & \cdots & v_{2t-2} \end{pmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{t-1} \end{bmatrix} = \begin{bmatrix} -v_t \\ -v_{t+1} \\ \vdots \\ -v_{2t-1} \end{bmatrix} = v.$$

The square $T \times T$ matrix

$$H_T = \begin{pmatrix} v_0 & v_1 & \cdots & v_{T-1} \\ v_1 & v_2 & \cdots & v_T \\ \vdots & \vdots & \vdots & \vdots \\ v_{T-2} & v_{r-1} & \cdots & v_{2rT-3} \\ v_{T-1} & v_T & \cdots & v_{2T-2} \end{pmatrix}$$

is a Hankel matrix of size T.

Theorem 3.4. [Ben-Or and Tiwari, 1988, Main theorem in Section 4, page 303] If $T \geq t$ then $\text{rank}(H_T) = t$.

The linear system $H_t\lambda = v$ can be solved using Gaussian elimination which does $O(t^3)$ arithmetic operations. But in coding theory, the Berlekamp-Massey Algorithm [Atti et al., 2006] solves the linear system using $O(t^2)$ arithmetic operations with $O(t)$ space. Moreover, one can modify the Extended Euclidean Algorithm (EEA) to also solve it [von zur Gathen and Gerhard, 2013]. Thus, obtaining the values for the λ_i means that we have recovered the monomial evaluations. The following pseudocode summarizes the Ben-Or/Tiwari algorithm.

The Ben-Or/Tiwari algorithm
Input: A black box for the polynomial $f = \sum_{i=1}^{t} a_i M_i(y_1, y_2, \ldots, y_n) \in \mathbb{Z}[y_1, y_2, \ldots, y_n]$.
Output: $f = \sum_{i=1}^{t} a_i M_i(y_1, y_2, \ldots, y_n) \in \mathbb{Z}[y_1, y_2, \ldots, y_n]$.

1. Compute $v_j = f(2^j, 3^j, \ldots, p_n^j) \in \mathbb{Z}$ via the black box for $j = 0, 1, 2, \ldots, 2t-1$.
2. Compute $\lambda(z)$ using the Berlekamp-Massey algorithm (BMA)[Atti et al., 2006].
3. Compute the roots $m_1, m_2, \ldots, m_t$ of $\lambda(z)$.
4. Obtain the exponents $e_{i,j}$ for M_i for $1 \leq j \leq n$ by factoring m_i via repeated trial divisions by the successive primes $2, 3, \ldots, p_n$. For example, $88200 = 2^{\mathbf{3}}3^{\mathbf{2}}5^{\mathbf{2}}7^{\mathbf{2}}$ which yields the monomial $y_1^3 y_2^2 y_3^2 y_4^2$.
5. Let $V_{ij} = m_i^{j-1}$ for $1 \leq i, j \leq t$ and let $v = [v_0, v_1, \ldots, v_{t-1}]$. Solve the $t \times t$ transposed Vandermonde system $Va = v$ for the coefficients a_i.

6. Output $\sum_{i=1}^{t} a_i M_i(y_1, y_2, \ldots, y_n)$

Remark 3.5. Recall that $v_j = f(2^j, 3^j, \ldots, p_n^j)$. Notice that the size of the evaluations v_j may be as large as ${p_n}^{d(2T-1)}$ where $d = \deg(f)$ which is $O(Td \log p_n)$ bits long, so as the parameters grow large, this bound will be very large to handle in practice. To avoid large intermediate integers, the algorithm must be performed over $\mathbb{Z}_p$ where $p > m_i \leq p_n^d$ so that the monomial evaluations m_i are uniquely determined.

3.2.3 Using discrete logarithms in the Ben-Or/Tiwari algorithm

Consider the black box representation of the polynomial

$$f = \sum_{k-1}^{t} a_i M_i(y_1, y_2, \ldots, y_n) \in \mathbb{Z}[y_1, y_2, \ldots, y_n]$$

where $d_i = \deg(f, y_i)$ and $d = \deg(f)$. Murao and Fujise in [Murao and Fujise, 1996] were the first to modify the Ben-Or/Tiwari algorithm to use the discrete logarithm approach in order to improve the prime requirement from $p > m_i \leq p_n^d$ to be $p > (d+1)^n$. Thus, the size of the prime p needed is now $O(n \log d)$ bits long. Such a prime p can be easily determined by picking small primes q_i satisfying $2|q_1, q_i > d_i$ and $\gcd(q_i, q_j) = 1$ for $1 \leq i \neq j \leq n$ such that $p = 1 + \prod_{i=1}^{n} q_i$ [Kaltofen, 2010, Subsection 2.1].

Now that we know how to get our working prime $p = 1 + \prod_{i=1}^{n} q_i$, we pick a generator α of $\mathbb{Z}_p^*$ at random and set $\omega_k = \alpha^{\frac{p-1}{q_k}} \implies \omega^{q_k} = 1$. We probe the black box and compute evaluations $v_j = f(\omega_1^j, \omega_2^j, \ldots, \omega_n^j)$ for $0 \leq j \leq 2T-1$ where $T \geq t$. Then we input these evaluations to the Berlekamp-Massey algorithm to obtain a feedback polynomial $\lambda(z)$ whose roots over $\mathbb{Z}_p$ are the monomial evaluations $m_i = M_i(\omega_1, \omega_2, \ldots, \omega_n) \in \mathbb{Z}_p$. So the prime point sequence $\{(2^j, 3^j, \ldots, p_n^j)\}$ which served as evaluation points in the Ben-Or/Tiwari algorithm is replaced with point sequence $\{(\omega_1^j, \omega_2^j, \ldots, \omega_n^j)\}$ for $0 \leq j \leq 2T-1$.
The exponent in all the variables present in a monomial M_i can be recovered as follows: Let

$$M_i = \prod_{j=1}^{n} y_j^{e_{i,j}}$$

so that $m_i = M_i(\omega_1, \omega_2, \ldots, \omega_n)$. Then we can write

$$m_i = \prod_{j=1}^{n} \omega_j^{e_{i,j}} = \alpha^{\frac{(p-1)e_{i,1}}{q_1} + \frac{(p-1)e_{i,2}}{q_2} + \cdots + \frac{(p-1)e_{i,n}}{q_n}}.$$

Applying $\log_\alpha$ to both sides of the above equation yields

$$\log_\alpha m_i = \log_\alpha \left(\alpha^{\sum_{k=1}^{n} \frac{(p-1)e_{i,k}}{q_k}} \right) = \sum_{k=1}^{n} \frac{(p-1)e_{i,k}}{q_k}.$$

Let $x = \log_\alpha m_i$. We have

$$e_{i,k} = x \left(\frac{p-1}{q_k}\right)^{-1} \bmod q_k.$$

Observe that

$$\left(\frac{p-1}{q_k}\right)^{-1}$$

exists since $\gcd(q_i, q_j) = 1$. Hence, step (d) of the Ben-Or/Tiwari algorithm is replaced with solving the discrete logarithm problem $x = \log_\alpha m_i$. We note that solving the discrete logarithm problem $x = \log_\alpha m_i$ is equivalent to solving for $x \in [0, p-2]$ such that $\alpha^x = m_i$ in $\mathbb{Z}_p$. In general, no efficient algorithm (i.e., polynomial time in $\log p$) is known for solving the discrete logarithm problem. However, the discrete logarithm problem can be solved using the Pohlig-Hellman algorithm [Pohlig and Martin, 1978] which costs $O\left(\sum_{i=1}^{k} d_i \left(\log p + \sqrt{p_i}\right)\right)$ [Hu and Monagan, 2016] for a prime decomposition of $p - 1 = \prod_{i=1}^{k} p_i^{d_i}$. To use the Pohlig-Hellman algorithm, we must choose p so that $p-1$ has small prime factors, thus keeping the cost of computing the discrete logarithms low. Thus, the algorithm works because $p-1$ has small prime factors and the monomial evaluations do not collide, i.e. $m_i \neq m_j$ for $i \neq j$.

Remark 3.6. In our proposed new sparse rational function algorithm which will be presented in Chapter 4, we reduce the size of the primes needed for our new algorithm which requires that we map a multivariate rational function $A = f/g$ to become a univariate rational function $K_r(A) \in \mathbb{Z}_p(y)$ using a Kronecker substitution K_r. Thus, we only have to interpolate univariate polynomials in $\mathbb{Z}_p[y]$ which means the point sequence $\{(\omega_1^j, \omega_2^j, \ldots, \omega_n^j)\}$ for $0 \leq j \leq 2T-1$ cannot be used. Instead, we pick a prime p of the form $p = 2^k s + 1$ where s is small, and we pick a generator α for $\mathbb{Z}_p^*$ in order to compute $v_j = H(\alpha^j)$ for $0 \leq j \leq 2T-1$, where $H \in \mathbb{Z}_p[y]$ is one of the polynomials to be interpolated in $K_r(A)$. More detail will be provided in Chapter 4.

3.3 Rational Function Interpolation

In this section, we describe the problem of interpolating a rational function in the univariate case because it is the main ingredient in our work. We begin with the discussion of the Extended Euclidean Algorithm, as it is the basic component in most rational function reconstruction algorithms.

3.3.1 The Extended Euclidean Algorithm

The classical Euclidean algorithm computes the greatest common divisor of two integers. It can also be used to compute the greatest common divisor (gcd) of polynomials in $F[x]$ where F is a field. The Euclidean algorithm is one of the important algorithms in computer algebra, and it has many nice properties and applications that go far beyond computing a

gcd . The Euclidean algorithm performs the division algorithm repeatedly for say l times, and outputs the last non-zero remainder as the greatest common divisor.

Lemma 3.7. [von zur Gathen and Gerhard, 2013, Lemma 3.8] For $0 \leq i \leq l+1$, we have $s_i\overline{m} + ut_i = r_i$ in Algorithm 6. In particular, $s_l\overline{m} + ut_l = \gcd(\overline{m}, u)$ up to a unit.

The above result tells us that the classical Euclidean algorithm can be extended so that it produces elements s_l and t_l and not only the gcd . This extension is what is referred to as the Extended Euclidean Algorithm (EEA). The main property of the EEA is that it produces s_i, t_i such that $s_i\overline{m} + ut_i = r_i$ at every division step. When the EEA terminates, we have $s_l\overline{m} + ut_l = r_l$ where $r_l = \gcd(\overline{m}, u)$.

Algorithm 6: Extended Euclidean Algorithm (EEA)

Input: $\overline{m}, u \in F[x]$ where F is a field and $\deg(\overline{m}) \geq \deg(u) \geq 0$.
Output: $r_l, s_l, t_l \in F[x]$ such that $s_l\overline{m} + ut_l = r_l$ where $l \in \mathbb{N}$.

$r_0 \leftarrow \overline{m}, \quad s_0 \leftarrow 1 \quad t_0 \leftarrow 0.$
$r_1 \leftarrow u, \quad s_1 \leftarrow 0, \quad t_1 \leftarrow 1.$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
  $q_i \leftarrow r_{i-1} \text{ quo } r_i$
  $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
  $s_{i+1} \leftarrow (s_{i-1} - q_i s_i)$
  $t_{i+1} \leftarrow (t_{i-1} - q_i t_i)$
  $i \leftarrow i + 1$
end
$l \leftarrow i - 1$
return $r_l, s_l, t_l \in F[x]$.

For $0 \leq i \leq l+1$, the elements r_i, s_i, t_i are generally referred to as the ith row of the EEA [von zur Gathen and Gerhard, 2013]. The r_i's are the remainders and q_i's are the quotients. The total cost of the EEA is $O(\deg(\overline{m})\deg(u))$ arithmetic operations in F, for any $\overline{m}, u \in F[x]$.

Lemma 3.8. [Khodadad and Monagan, 2006, Lemma 2.2] Let F be a field and let $\overline{m}, u \in F[x]$ such that $\deg(\overline{m}) > \deg(u) \geq 0$. In the EEA for $\overline{m}$ and u, we have $\deg(r_i) + \deg(t_i) + \deg(q_i) = \deg(\overline{m})$ for $1 \leq i \leq l$ where l is the total number of division steps.

We give the following lemma which describes some nice properties of the EEA.

Lemma 3.9. [von zur Gathen and Gerhard, 2013, Lemma 3.8] For $0 \leq i \leq l$, we have

1. $\gcd(f, g) \sim \gcd(r_i, r_{i+1}) \sim r_l$,
2. $s_i f + t_i g = r_i$,

3. $s_i t_{i+1} - t_i s_{i+1} = (-1)^i$,

4. $\gcd(r_i, t_i) \sim \gcd(f, t_i)$,

5. $f = (-1)^i (t_{i+1} r_i - t_i r_{i+1})$,

6. $g = (-1)^{i+1} (s_{i+1} r_i - s_i r_{i+1})$,

with the convention that $r_{l+1} = 0$. Here $a \sim b$ means that a and b are associates.

Theorem 3.10. [von zur Gathen and Gerhard, 2013, Corollary 3.9] Let $a, b \in \mathbb{D}$ where $\mathbb{D}$ is a Euclidean domain. Then there exist $s, t \in \mathbb{D}$ such that $sa + tb = \gcd(a, b)$.

3.3.2 The Monic Extended Euclidean Algorithm

Let $\gcd(\overline{m}, u)$ denote a greatest common divisor of $\overline{m}$ and u. Since it is possible to have more than one $\gcd(\overline{m}, u)$, the question of deciding which gcd to choose arises naturally. Whenever this happens, they differ by a unit. An extra condition may be imposed on gcd computations if we want it to be unique. For example, we force the gcd to be positive when computing the gcd of two integers. We are always concerned with computing the gcd of polynomials, and we want it to be unique for our purposes. Thus, we ensure that our gcd is unique by making it monic.

Algorithm 7: Monic Extended Euclidean Algorithm (MEEA)

Input: $\overline{m}, u \in F[x]$ where F is a field and $\deg(\overline{m}) \geq \deg(u) \geq 0$.

Output: $r_l, s_l, t_l \in F[x]$ such that $l \in \mathbb{N}$.

$\rho_0 \leftarrow \mathrm{lc}(\overline{m}), \quad r_0 \leftarrow \overline{m}/\rho_0, \quad s_0 \leftarrow \rho_0^{-1}, \quad t_0 \leftarrow 0.$

$\rho_1 \leftarrow \mathrm{lc}(u), \quad r_1 \leftarrow u/\rho_1, \quad s_1 \leftarrow 0, \quad t_1 \leftarrow \rho_1^{-1}.$

$i \leftarrow 1$

while $r_i \neq 0$ **do**

  $q_i \leftarrow r_{i-1} \text{ quo } r_i$

  $r_{i+1} \leftarrow r_{i-1} - q_i r_i$

  $s_{i+1} \leftarrow (s_{i-1} - q_i s_i)$

  $t_{i+1} \leftarrow (t_{i-1} - q_i t_i)$

  if $r_{i+1} \neq 0$ **then**

    $\rho_{i+1} \leftarrow \mathrm{lc}(r_{i+1})$

    $r_{i+1} \leftarrow r_{i+1}/\rho_{i+1}$

    $s_{i+1} \leftarrow s_{i+1}/\rho_{i+1}$

    $t_{i+1} \leftarrow t_{i+1}/\rho_{i+1}$

  end

  $i \leftarrow i + 1$

end

$l \leftarrow i - 1$

return $r_l, s_l, t_l \in F[x]$.

Similar to the EEA, the elements r_i, s_i, t_i form the ith row of the MEEA [von zur Gathen and Gerhard, 2013] for $0 \leq i \leq l+1$. The r_i's are the remainders and q_i's are the quotients. The elements s_l and t_l satisfying

$$s_l f + t_l g = \gcd(f, g)$$

are referred to as the Bezout coefficients of f and g. Since the main objective of the MEEA is to guarantee uniqueness of the gcd, we restate Lemma 3.9 as follows for the entries of the Monic Extended Euclidean Algorithm.

Lemma 3.11. [von zur Gathen and Gerhard, 2013, Lemma 3.15] Using the notations of Algorithm 7 for $0 \leq i \leq l$,we have

1. $\gcd(f, g) = \gcd(r_i, r_{i+1}) = r_l$.
2. $s_i t_{i+1} - t_i s_{i+1} = (-1)^i (\rho_0 \rho_1 \cdots \rho_{i+1})^{-1}$.
3. $\gcd(r_i, t_i) = \gcd(f, t_i)$.
4. $f = (-1)^i (\rho_0 \rho_1 \cdots \rho_{i+1})(t_{i+1} r_i - t_i r_{i+1})$.
5. $g = (-1)^{i+1} (\rho_0 \rho_1 \cdots \rho_{i+1})(s_{i+1} r_i - s_i r_{i+1})$.

3.3.3 Univariate Rational Function Reconstruction

Let F be a field and let $(\alpha_1, \alpha_2, \cdots, \alpha_d) \in F^d$ such that $\alpha_i \neq \alpha_j$ for $i \neq j$ and $d > 0$. Univariate rational function interpolation (Cauchy interpolation) is the problem of finding a rational function

$$B(x) = \frac{f(x)}{g(x)} \in F(x)$$

with $f, g \in F[x]$ such that

$$g(\alpha_i) \neq 0, \quad B(\alpha_i) = \frac{f(\alpha_i)}{g(\alpha_i)} = y_i \text{ for } 1 \leq i \leq d.$$

Without loss of generality, we require g to be monic [von zur Gathen and Gerhard, 2013] and $\gcd(f, g) = 1$. Newton or Lagrange interpolation algorithm [von zur Gathen and Gerhard, 2013, Section 5.2] can be used to find the unique polynomial $u(x) \in F[x]$ of degree less than d such that $u(\alpha_i) = y_i$ for $1 \leq i \leq d$. The implication of this is that

$$u(x) \equiv y_i \pmod{(x - \alpha_i)}.$$

Thus, for $1 \leq i \leq d$, we have

$$B(x) = \frac{f(x)}{g(x)} \equiv u(x) \pmod{(x - \alpha_i)}. \tag{3.2}$$

Let

$$\overline{m}(x) = \prod_{i=1}^{d}(x - \alpha_i).$$

By the Chinese remainder theorem [von zur Gathen and Gerhard, 2013], (3.2) becomes

$$B(x) \equiv u(x) \pmod{\overline{m}(x)} \text{ and } \gcd(\overline{m}, g) = 1.$$

Therefore, with the help of the Chinese remainder theorem, the rational function interpolation problem now becomes a new problem of finding the rational function $B(x) = \frac{f(x)}{g(x)} \in F(x)$, given a polynomial $\overline{m}(x)$ of degree d and the unique polynomial $u(x)$ of degree less than d. This problem is what we refer to as the rational function reconstruction problem [von zur Gathen and Gerhard, 2013]. We restate the rational function reconstruction problem formally.

Problem 3.12. Let $\overline{m}, u \in F[x]$ with $d = \deg(\overline{m}) > 0$, and $d_u = \deg(u)$ such that $d_u < d$. We seek a rational function $B \in F(x)$ where

$$B(x) = \frac{f(x)}{g(x)},$$

with the property that

$$\frac{f}{g} \equiv u \bmod \overline{m}$$

such that $\gcd(f, g) = \gcd(\overline{m}, u) = 1$, g is monic, and $\deg(f) + \deg(g) < d$.

Let N, D be degree bounds such that $N \geq \deg(f)$ and $D \geq \deg(g)$, with $N + D < d$. We consider the following 2 cases:

(1) Degree Bounds Known

If N and D are known, the univariate rational function $B(x) = \frac{f(x)}{g(x)}$ can be recovered using the EEA [von zur Gathen and Gerhard, 2013, Theorem 5.16]. For the sake of brevity, this theorem will not be stated here, but the main steps involved are:

(i) Construct polynomials $\overline{m}(x)$ and $u(x)$ such that $\deg(\overline{m}) = d$ and $\deg(u) < d$, and use the polynomials $\overline{m}(x)$ and $u(x)$ as inputs in the EEA.

(ii) Terminate the EEA when $\deg(r_k) \leq N$.

(iii) The output should be $\frac{r_k}{t_k}$ with $\deg(t_k) \leq D$ and $\gcd(t_k, \overline{m}) = 1$.

(iv) To uniquely determine $B(x) = \frac{f(x)}{g(x)}$, make t_k monic so that $\frac{r_k}{t_k} \equiv B(x) = \frac{f(x)}{g(x)}$.

The method described above uses $O(d^2)$ arithmetic operations in F. Another approach is to set up a linear system of $d = N + D + 1$ equations involving d unknowns which can be solved using Gaussian elimination. This method is expensive as it uses $O(d^3)$ arithmetic operations in F.

Example 3.13. Let $F = \mathbb{Z}_{19}$ and let

$$B = \frac{f}{g} = \frac{3x^2 + 4}{x + 2}.$$

Suppose we aim to reconstruct $B = f/g \in F(x)$ with degree bounds $N = 2$ and $D = 1$. So, using $d = 4 > N + d$, suppose we are given values $B(15) = 13, B(13) = 10, B(6) = 14$ and $B(2) = 4$. We construct

$$\overline{m} = (x - 15)(x - 13)(x - 6)(x - 2) = x^4 + 2x^3 + 13x^2 + 4x + 3,$$

and the interpolating polynomial

$$u = 13x^3 + x + 12.$$

The following table shows the values of r_i, t_i when the EEA is called with inputs $\overline{m}$ and u.

Table 3.1: EEA computations for input polynomials $\overline{m}$ and u

i	r_i	t_i
0	$x^4 + 2x^3 + 13x^2 + 4x + 3,$	0
1	$13x^3 + x + 12$	1
2	$10x^2 + 7$	$16x + 13$
3	$9x + 2$	$2x^2 + 4x + 1$
4	10	$2x^3 + 14x^2 + 18x + 18$
5	0	$2x^4 + 4x^3 + 7x^2 + 8x + 6$

Observe that in row 2 of Table 3.1, the degree of $r_2 = N = 2$ and $\deg(t_2) = D = 1$. So, the correct output is

$$\frac{r_2}{t_2} = \frac{10x^2 + 7}{16x + 13}.$$

Making t_2 monic, we obtain our desired rational function

$$B = \frac{f}{g} = \frac{3x^2 + 4}{x + 2}.$$

(2) Degree Bounds Unknown

For a black box representation of a univariate rational function, the degree bounds N and D are unknown. We modify the EEA to use maximal quotient rational function reconstruction to discover the degree of the numerator f and the denominator g with high probability, provided we use more than enough evaluation points, say d points with $d > \deg(f) + \deg(g) + 1$. Maximal quotient rational function reconstruction by Khodadad & Monagan [Khodadad and Monagan, 2006] can be used to find the desired

univariate rational function with high probability. Maximal quotient rational function reconstruction is based on Monagan's work [Monagan, 2004] for reconstructing a rational number from its integer image modulo another integer. We note that Paul Wang was the first one who presented an algorithm based on the EEA that allows rational number reconstruction [Wang, 1981].

Algorithm 8: Maximal Quotient Rational Function Reconstruction Algorithm (MQRFR)

Input: $\overline{m}, u, \in F[x]$ where F is a field and $\deg(m) > \deg(u) \geq 0$ or $u = 0$ and $\deg(\overline{m}) \geq 1$.

Output: Either $f, g \in F[x]$ satisfying

$$f/g \equiv u \bmod \overline{m}, \gcd(u, g) = \gcd(f, g) = 1, \mathrm{lc}(g) = 1,$$

and $\deg(f) + \deg(g) + 1 < \deg(\overline{m})$, or FAIL implying that no such solution exists.

Remark : The degree requirement $\deg(f) + \deg(g) + 1 < \deg(m)$ is met by requiring that one of the quotients q_i in the Euclidean algorithm has degree at least 2.

```
1  if u = 0 then return (f, g) ← (0, 1) end
2  (r_0, r_1) ← (m̄, u)    (t_0, t_1) ← (0, 1)
3  (f, g) ← (r_1, t_1)    q_max ← 1
4  i ← 1
5  while r_i ≠ 0 do
6  |   q_i ← r_{i−1} quo r_i
7  |   if deg q_i > q_max then
8  |   |   q_max ← deg q_i
9  |   |   (f, g) ← (r_i, t_i)
10 |   end
11 |   (r_{i+1}, t_{i+1}) ← (r_{i−1} − q_i r_i, t_{i−1} − q_i t_i)
12 |   i ← i + 1
13 end
14 if q_max ≤ 1 or gcd(f, g) ≠ 1 then
15 |   return FAIL
16 end
17 return (f/lc(g), g/lc(g)).
```

Let l denote the total number of division steps for the EEA with inputs $\overline{m}$ and u. The maximal quotient rational function reconstruction algorithm returns a rational function $\frac{r_i}{t_i} \equiv B = \frac{f}{g}$ with $\deg r_i + t_i$ minimal for $i = 1, 2, \cdots l$, because Lemma 3.8 guarantees the maximality of the quotient degree as long as d is large enough. Thus, we recover the univariate rational function represented by a black box using the following steps.

1. Pick $(\alpha_1, \alpha_2, \cdots, \alpha_d) \in \mathbb{Z}_p^d$ at random where p is prime with $d = \deg(f) + \deg(g) + 2$.
2. Compute $\overline{m}(x) = \prod_{i=1}^{d}(x - \alpha_i) \in \mathbb{Z}_p[x]$, and the interpolating polynomial $u(x)$ with $d = \deg(\overline{m}) > \deg(u) \geq 0$.

3. Call the MQRFR algorithm with input polynomials $\overline{m}$ and u.

Note that Algorithm 8 is probabilistic, which means it could output a wrong answer. Thus, the evaluation points must be chosen randomly from $\mathbb{Z}_p$ and p must be large so that the output is correct with high probability.

Example 3.14. Let $F = \mathbb{Z}_{19}$ and let

$$B = \frac{f}{g} = \frac{3x^2 + 4}{x + 2}.$$

Suppose we seek a rational function $B = f/g \in F(x)$ with $d = 5$,

$$\overline{m} = x(x-1)(x+1)(x-2)(x+3) = x^5 + x^4 + 12x^3 + 18x^2 + 6x,$$

and the interpolating polynomial

$$u = 12x^4 + 7x^3 + 16x^2 + 16x + 2.$$

The following table shows the values of q_i, r_i, t_i when the MEEA is called with inputs $\overline{m}$ and u.

Table 3.2: MEEA computations for input polynomials $\overline{m}$ and u

i	q_i	r_i	t_i
0	–	$x^5 + x^4 + 12x^3 + 18x^2 + 6x$	0
1	$x + 2$	$x^4 + 18x^3 + 14x^2 + 14x + 16$	8
2	$x^2 + 18x$	$x^2 + 14$	$13x + 7$
3	$x + 13$	$x + 6$	$7x^3 + 7x^2 + 5x + 3$
4	$x + 6$	1	$x^4 + 14x^3 + 11x^2 + 16x + 10$
5	–	0	$18x^5 + 18x^4 + 7x^3 + x^2 + 13x$

Observe that in row 2 of Table 3.2, the degree of quotient q_2 is maximal when compared to other q_i's. So we get

$$\frac{r_2}{t_2} = \frac{x^2 + 14}{13x + 7}, \quad (d = 5 > \deg(f) + \deg(g) = 3).$$

Making g monic, we have that

$$B = \frac{f}{g} = \frac{3x^2 + 4}{x + 2}.$$

Also, notice in Table 3.2 that

$$\deg q_i + \deg r_i + \deg t_i = 5 = \deg(\overline{m}) \quad \text{for } 1 \leq i \leq 4,$$

which illustrates Lemma 3.8.

Note that if $F = \mathbb{Q}$, then our rational function computations must be done modulo sufficiently many primes to recover the rational coefficients in $\mathbb{Q}(x)$ using Chinese remaindering and rational number reconstruction [Monagan, 2004].

3.4 Sparse Multivariate Rational Function Interpolation

Let $A = f/g$ be a sparse multivariate rational function where f and g are polynomials in $y_1, y_2, \ldots, y_m$. Suppose that $A = f/g$ is represented by a black box. Let the degree of A be $\deg(A) = \deg(f) + \deg(g)$ and let $t = \max(\#f, \#g)$ be the maximum number of terms in $A = f/g$.

Kaltofen and Trager developed a method in [Kaltofen and Trager, 1990] that evaluates the numerator and denominator of A separately. Their central idea is to separately evaluate a rational function $A \in F(y_1, y_2, \cdots, y_m)$ where F is a field of characteristic 0, is to interpolate a bivariate function $T(x, y)$ such that

$$\begin{aligned} T(x,y) &= A(x, a_2x + b_2 + y(\alpha_2 - a_2\alpha_1 - b_2) \cdots, a_mx + b_m + y(\alpha_m - a_m\alpha_1 - b_m)) \\ &= \frac{f(x, a_2x + b_2 + y(\alpha_2 - a_2\alpha_1 - b_2) \cdots, a_mx + b_m + y(\alpha_m - a_m\alpha_1 - b_m))}{g(x, a_2x + b_2 + y(\alpha_2 - a_2\alpha_1 - b_2) \cdots, a_mx + b_m + y(\alpha_m - a_m\alpha_1 - b_m))} \end{aligned} \tag{3.3}$$

where $(b_2, b_3, \cdots, b_m), (a_1, a_2, a_3, \cdots, a_m) \in S \subset F^{m-1}$ are chosen randomly, and $(\alpha_1, \alpha_2, \cdots, \alpha_n)$ is the evaluation point. Evaluating the bivariate function $T(x, y)$ at $x = \alpha_1$ and $y = 1$ produces the desired separation $f(\alpha_1, \alpha_2, \cdots, \alpha_m)$ and $g(\alpha_1, \alpha_2, \cdots, \alpha_m)$.

Kaltofen and Yang presented a more efficient black box multivariate rational function separation algorithm in [Kaltofen and Yang, 2007]. This algorithm interpolates the univariate rational function

$$\begin{aligned} T(x) &= A(x, x\beta_2 - \beta_2\sigma_1 + \sigma_2, \cdots, x\beta_m - \beta_m\sigma_1 + \sigma_m) \\ &= \frac{f(x, x\beta_2 - \beta_2\sigma_1 + \sigma_2, \cdots, x\beta_n - \beta_m\sigma_1 + \sigma_m)}{g(x, x\beta_2 - \beta_2\sigma_1 + \sigma_2, \cdots, x\beta_m - \beta_m\sigma_1 + \sigma_m)}, \end{aligned} \tag{3.4}$$

for an evaluation point $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_m)$, such that $\beta = (\beta_2, \beta_2, \cdots, \beta_m)$ is a point chosen at random. One clearly sees that $T(\sigma_1) = \frac{f(\sigma_1, \sigma_2, \cdots, \sigma_m)}{g(\sigma_1, \sigma_2, \cdots, \sigma_m)}$ is the desired output. This black box separation algorithm can be combined with any sparse polynomial interpolation algorithm to interpolate A, by simultaneously interpolating the numerator and denominator of A from the evaluations $f(\sigma_1, \sigma_2, \cdots, \sigma_m)$ and $g(\sigma_1, \sigma_2, \cdots, \sigma_m)$. Kaltofen and Yang's method uses $O(\deg(A))$ probes to interpolate $T(x)$. Using the Ben-Or/Tiwari algorithm to interpolate f and g needs $O(t)$ probes which implies $O(t \deg(A))$ probes are needed to recover $A = f/g$.

The *RATZIP* algorithm by Monagan and De Kleine in [de Kleine et al., 2005] requires $O(mt \deg(A))$ probes to the black box in order to interpolate A. This algorithm reconstructs A from monic univariate images using a sparse, one variable at a time, rational function

interpolation algorithm. For example, suppose $A = f/g$ in variables y_1, y_2 is to be interpolated. The algorithm first reconstructs A at y_1, then normalizes the leading coefficient of the denominator so that it is monic. Then it uses enough points to interpolate the second variable y_2 so that

$$A \equiv \frac{\sum_{i=0}^{d_f} \frac{a_i(y_2)}{b_i(y_2)} y_1^i}{y_1^{d_g} + \sum_{i=0}^{d_g-1} \frac{n_i(y_2)}{d_i(y_2)} y_1^i}$$

where $d_f = \deg(f, y_1)$ and $d_g = \deg(g, y_1)$.

Let $f = \sum_{i=0}^{\deg(f)} f_i$ and let $g = \sum_{i=0}^{\deg(g)} g_i$ such that the polynomials f_i and g_i are homogeneous and $\deg(f_i) = \deg(g_i) = i$. Let $t_{\max} = \max_i(\#f_i, \#g_i)$. To ensure that the smallest number of black box probes are used to interpolate a sparse multivariate rational function $A = f/g$ in our proposed algorithms, we modify Cuyt and Lee's [Cuyt and Lee, 2011] sparse multivariate rational function algorithm for our purposes. This is because Cuyt and Lee's method requires $O(t_{\max} \deg(A))$ probes to interpolate A if the Ben-Or/Tiwari algorithm is used as the main sparse polynomial interpolation algorithm to interpolate f and g. Thus, Cuyt and Lee's method uses a factor of $O(d)$ less probes than Kaltofen and Yang's method. For example, in Example 3.15, Cuyt and Lee's method needed 42 probes in comparison with Kaltofen and Yang's method which required 56 probes to interpolate f/g. Unfortunately, Cuyt and Lee's algorithm is more complicated than Kaltofen and Yang's method.

3.4.1 Cuyt and Lee's algorithm

Let $\mathbb{K}$ be a field and let $A = f/g$ be a sparse multivariate rational function where $f, g \in \mathbb{K}[y_1, \ldots, y_m]$ such that $\gcd(f, g) = 1$. Cuyt and Lee's algorithm [Cuyt and Lee, 2011] to interpolate f/g must be combined with a sparse polynomial interpolation algorithm to interpolate the sparse polynomials f and g.

The first step in their algorithm is to introduce a homogenizing variable z to form the auxiliary function

$$\frac{f(y_1 z, \ldots, y_m z)}{g(y_1 z, \ldots, y_m z)},$$

which can be written as

$$\frac{f(y_1 z, \ldots, y_m z)}{g(y_1 z, \ldots, y_m z)} = \frac{f_0 + f_1(y_1, \ldots, y_m) z + \cdots + f_{\deg(f)}(y_1, \ldots, y_m) z^{\deg(f)}}{g_0 + g_1(y_1, \ldots, y_m) z + \cdots + g_{\deg(g)}(y_1, \ldots, y_m) z^{\deg(g)}}, \tag{3.5}$$

and then normalize it using either $f_0 \neq 0$ or $g_0 \neq 0$. Observe that if $z = 1$ in (3.5) then $f = \sum_{j=0}^{\deg(f)} f_j(y_1, y_2, \ldots, y_m)$ and $g = \sum_{j=0}^{\deg(g)} g_j(y_1, y_2, \ldots, y_m)$, where $\deg(f_j) = \deg(g_j) = j$.

If $A = f/g$ is represented by a black box, it is easy to detect the presence of g_0. This can be done by inputting $(0, 0, \ldots, 0)$ to the black box and the output $A(0, 0, \ldots, 0) \in \mathbb{K}$ confirms the occurrence of g_0. However, it is not uncommon to have $f_0 = g_0 = 0$. Thus in

the case when both constant terms g_0 and f_0 are zero, Cuyt and Lee choose a basis shift $\beta \neq 0 \in \mathbb{K}^m$ such that $g(\beta) \neq 0$ and form a new auxiliary rational function as

$$\frac{\hat{f}(y_1 z, \ldots, y_m z)}{\hat{g}(y_1 z, \ldots, y_m z)} := \frac{f(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}{g(y_1 z + \beta_1, \ldots, y_m z + \beta_m)} = \frac{\sum_{j=0}^{\deg(f)} \hat{f}_j(y_1, \ldots, y_m) z^j}{\sum_{j=0}^{\deg(g)} \hat{g}_j(y_1, \ldots, y_m) z^j}.$$

The introduction of the basis shift β leads to the production of a constant term in $\hat{f}/\hat{g}$ so that $\hat{f}/\hat{g}$ can be normalized using either $\hat{f}_0$ or $\hat{g}_0$. Thus, we may write

$$\frac{\hat{f}(y_1 z, \ldots, y_m z)}{\hat{g}(y_1 z, \ldots, y_m z)} = \frac{\sum_{j=0}^{\deg(f)} \frac{\hat{f}_j(y_1, \ldots, y_m) z^j}{\hat{g}_0}}{1 + \sum_{j=1}^{\deg(g)} \frac{\hat{g}_j(y_1, \ldots, y_m) z^j}{\hat{g}_0}}.$$

Note that $\hat{g}_0 = \tilde{c} \times g(\beta_1, \beta_2, \ldots, \beta_m) \neq 0$ for some $\tilde{c} \in \mathbb{K}$.

If a rational function $A = f/g \in \mathbb{Q}(y_1, y_2, \ldots, y_m)$ is represented by a black box, we can recover it by first densely interpolating the univariate auxiliary rational functions

$$\hat{A}(\alpha^j, z) = \frac{\frac{\hat{f}_0}{\hat{g}_0} + \frac{\hat{f}_1(\alpha^j)}{\hat{g}_0} z + \cdots + \frac{\hat{f}_{\deg(f)}(\alpha^j)}{\hat{g}_0} z^{\deg(f)}}{1 + \frac{\hat{g}_1(\alpha^j)}{\hat{g}_0} z + \cdots + \frac{\hat{g}_{\deg(g)}(\alpha^j)}{\hat{g}_0} z^{\deg(g)}} \in \mathbb{Z}_p(z) \text{ for } j = 0, 1, 2, \cdots$$

for $\alpha \in \mathbb{Z}_p^m$ from the black box, and then use the coefficients of $\hat{A}(\alpha^j, z)$ via sparse interpolation to recover f/g. In order to densely interpolate $\hat{A}(\alpha^j, z)$, we use the Maximal Quotient Rational Function Reconstruction algorithm (MQRFR) [Monagan, 2004] which requires $\geq \deg(f) + \deg(g) + 2$ black box random probes on z.

Note that the use of a basis shift in the formation of the auxiliary rational function does not affect the total degrees and the leading coefficients of the numerator and denominator of the rational function $A = f/g$, but it does affect the lower degree coefficients because the sparsity of $A = f/g$ is destroyed. The effect of this basis shift can be removed by adjusting the coefficients of the lower degree terms in the numerator and denominator of $\hat{A}(\alpha^j, z)$ using the contributions from the higher degree terms, before sparse interpolation is performed. We explain how this works using the numerator part of $A = f/g$. Let

$$f = \sum_{j=0}^{\deg(f)} f_j(y_1, y_2, \ldots, y_m),$$

and let

$$\frac{f(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}{g(y_1 z + \beta_1, \ldots, y_m z + \beta_m)} = \frac{\overline{f}_0 + \overline{f}_1(y_1, \ldots, y_m) z + \cdots + \overline{f}_{\deg(f)}(y_1, \ldots, y_m) z^{\deg(f)}}{\overline{g}_0 + \overline{g}_1(y_1, \ldots, y_m) z + \cdots + \overline{g}_{\deg(g)}(y_1, \ldots, y_m) z^{\deg(g)}}.$$

Since the basis shift β does not affect the leading coefficients and the total degrees of $f(y_1 z + \beta_1, \ldots, y_m z + \beta_m)$ and $g(y_1 z + \beta_1, \ldots, y_m z + \beta_m)$ in z, suppose we have interpolated

the leading term polynomial

$$f_{\deg(f)}(y_1, y_2, \ldots, y_m) = \overline{f}_{\deg(f)}(y_1, y_2, \ldots, y_m)$$

in the numerator part and the basis shift β applied is known. Then we move on to interpolate the next lower degree term denoted by $f_{\deg(f)-1}(y_1, y_2, \ldots, y_m)$. Recall that the sparse representation form of f is written as

$$f = \sum_{k=1}^{s} a_k y_1^{d_{k,1}} y_2^{d_{k,2}} \cdots y_m^{d_{k,m}}.$$

Notice that in the expansion of $f(y_1 z + \beta_1, y_2 z + \beta_2, \ldots, y_m z + \beta_m)$

$$\begin{aligned}
&= \sum_{k=1}^{s} a_k (y_1 z + \beta_1)^{d_{k,1}} (y_2 z + \beta_2)^{d_{k,2}} \cdots (y_m z + \beta_m)^{d_{k,m}} \\
&= \sum_{j=0}^{\deg(f)} \left(\sum_{d_{k,1}+d_{k,2}+\cdots d_{k,m}=j} a_k (y_1 z + \beta_1)^{d_{k,1}} \cdots (y_m z + \beta_m)^{d_{k,m}} \right),
\end{aligned}$$

the coefficient of $z^{\deg(f)-1}$ is being contributed to by the expansions of

$$\sum_{d_{k,1}+d_{k,2}+\cdots d_{k,m}=\deg(f)} a_k (y_1 z + \beta_1)^{d_{k,1}} (y_2 z + \beta_2)^{d_{k,2}} \cdots (y_m z + \beta_m)^{d_{k,m}} \tag{3.6}$$

and

$$\sum_{d_{k,1}+d_{k,2}+\cdots d_{k,m}=\deg(f)-1} a_k (y_1 z + \beta_1)^{d_{k,1}} \cdots (y_m z + \beta_m)^{d_{k,m}}. \tag{3.7}$$

Since $\overline{f}_{\deg(f)}(y_1 z + \beta_1, \ldots, x_m z + \beta_m)$ can be written as

$$\begin{aligned}
&\overline{f}_{\deg(f)}(y_1 z + \beta_1, \ldots, x_m z + \beta_m) \\
&= \sum_{d_{k,1}+d_{k,2}+\cdots d_{k,m}=\deg(f)} a_k (y_1 z + \beta_1)^{d_{k,1}} (y_2 z + \beta_2)^{d_{k,2}} \cdots (y_m z + \beta_m)^{d_{k,m}} \\
&= \sum_{j=0}^{\deg(f)} u_j(y_1, y_2, \cdots, y_m) z^j,
\end{aligned}$$

we remove the effect of β for the coefficient of $z^{\deg(f)-1}$ in the numerator by computing

$$f_{\deg(f)-1}(y_1, y_2, \ldots, y_m) = \overline{f}_{\deg(f)-1}(y_1, y_2, \ldots, y_m) - u_{\deg(f)-1}(y_1, y_2, \cdots, y_m). \tag{3.8}$$

We emphasize that we do not actually have (3.8) since the sparse multivariate rational function $A = f/g$ is represented by a black box. Thus, for inputs $\alpha^i = (\alpha_1^i, \alpha_2^i, \cdots, \alpha_m^i)$ to

the black box representing f/g, the images

$$f_{\deg(f)-1}(\alpha_1^i, \alpha_2^i, \cdots, \alpha_n^i) = \overline{f}_{\deg(f)-1}(\alpha_1^i, \alpha_2^i, \cdots, \alpha_m^i) - u_{\deg(f)-1}(\alpha_1^i, \alpha_2^i, \cdots, \alpha_m^i)$$

are computed via the dense univariate rational functions in z to interpolate $f_{\deg(f)-1}$, and these values must be stored in order to be reused when interpolating the lower degree polynomials $f_i(y_1, y_2, \ldots, y_m)$, until one gets to the constant term. Continuing in similar fashion for the rest of the terms in the denominator and numerator, one successfully recovers the rational function $A = f/g$. We give the following example to help the reader understand the reconstruction of a sparse multivariate rational function using Cuyt and Lee's method, with the Ben-Or/Tiwari algorithm serving as the primary sparse interpolation algorithm.

Example 3.15. Let

$$A(x, y, z) = \frac{f(y_1, y_2, y_3)}{g(y_1, y_2, y_3)} = \frac{y_1^2 + y_2^2 + y_3^2 + y_3}{y_1 y_2 y_3}$$

be a rational function represented by a black box $\mathbf{B}$, which outputs **FAIL** if a division by 0 occurs. In other words, $A(0,0,0) = \frac{f(0,0,0)}{g(0,0,0)} = \frac{0}{0}$. Suppose we want to interpolate A. Since A is represented by $\mathbf{B}$, the only information available to use is the number of variables $n = 3$. Let prime $p = 3137$.

1. We first compute $\mathbf{B}(0,0,0)$. Since $A(0,0,0)$ is not defined, $\mathbf{B}(0,0,0)$ returns FAIL so we need to use a basis shift β.

2. We pick a random basis shift $\beta \in \mathbb{Z}_{3137}$ until $A(\beta) \neq 0$. Here, we use $\beta = (2811, 1186, 1298)$.

3. Algorithms for computing $\deg(f)$ and $\deg(g)$ probabilistically will be presented in the next chapter. For this example, we assume that the total degrees $\deg(f) = 2$ and $\deg(g) = 3$, and the maximum number of terms $t = \max(\#f, \#g) = 4$ are known.

4. Since we need to perform dense interpolation of auxiliary functions in the new homogenizing variable z, we pick random evaluation points in $\mathbb{Z}_{3137}$ for z. Let

   $$\{2909, 2799, 1325, 2016, 6, 3003, 2348\}$$

   be the seven $(\deg(f)+\deg(g)+2)$ random points needed to perform univariate rational interpolation using the MQRFR algorithm. Let T denote the number of auxiliary rational functions needed. In this case, $T = 6 < 2t$.

   We input the prime p and the evaluation points

   $$(2^i z + 2811, 3^i z + 1186, 5^i z + 1298)$$

for $i = 0, 1, 2, \cdots, T-1$ to the black box at each point

$$z = 2909, 2799, 1325, 2016, 6, 3003, 2348,$$

and then densely interpolate the auxiliary rational functions. The interpolated auxiliary functions $F_i = N_i/D_i$ for $i = 0, 1, 2, \ldots, 5$ are:

$$\begin{aligned}
F_0(z) &= \frac{58z^2 + 1900z + 2967}{z^3 + 2158z^2 + 1860z + 595} \\
F_1(z) &= \frac{2826z^2 + 1608z + 2967}{580z^3 + 900z^2 + 683z + 1} \\
F_2(z) &= \frac{365z^2 + 3014z + 2967}{1715z^3 + 608z^2 + 884z + 1} \\
F_3(z) &= \frac{2669z^2 + 2785z + 2967}{1258z^3 + 2255z^2 + 3024z + 1} \\
F_4(z) &= \frac{320z^2 + 1478z + 2967}{96z^3 + 1091z^2 + 1367z + 1} \\
F_5(z) &= \frac{1414z^2 + 43z + 2967}{2880z^3 + 1768z^2 + 2912z + 1}.
\end{aligned}$$

5. Starting from the highest degree in the numerator of F_i which is 2, we collect the sequence of coefficients of z^2 which is given by

   $$v = [58, 2826, 365, 2669, 320, 1414].$$

   Then we input v to the Berlekamp-Massey Algorithm (BMA) to obtain the feedback polynomial $\lambda(z) = z^3 + 3099z^2 + 361z + 2237 \in \mathbb{Z}_p[z]$.

6. Next we compute the roots of $\lambda(z)$. We obtain $\hat{m} = \{4 = 2^2, 9 = 3^2, 25 = 5^2\}$. Performing repeated trial divisions on the elements of $\hat{m}$, we obtain $\{y_1^2, y_2^2, y_3^2\}$. Next, we set up and solve the transposed Vandermonde system

   $$Va = \begin{bmatrix} 1 & 1 & 1 \\ \hat{m}_1 = 4 & \hat{m}_2 = 9 & \hat{m}_3 = 25 \\ \hat{m}_1^2 = 16 & \hat{m}_2^2 = 81 & \hat{m}_3^2 = 625 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 58 \\ 2826 \\ 365 \end{bmatrix} = v.$$

   Thus, we recover

   $$f_2 = 1065y_1^2 + 1065y_2^2 + 1065y_3^2.$$

7. The next lower total degree in the numerator of $A = f/g$ is 1 (the final degree in our example is 1 as there is no constant term), so we move on to interpolate any degree one polynomial in f. However, we need to remove the effect of the basis shift. We do this as follows. Suppose we compute $H_i = f_2(2^i z + \beta_1, 3^i z + \beta_2, 5^i z + \beta_3) \in \mathbb{Z}_p[z]$

directly for $0 \leq i \leq 5$, such that

$$\begin{aligned}
H_0(z) &= 58z^2 + 835z + 877 \\
H_1(z) &= 2826z^2 + 2557z + 877 \\
H_2(z) &= 365z^2 + 1485z + 877 \\
H_3(z) &= 2669z^2 + 1414z + 877 \\
H_4(z) &= 320z^2 + 897z + 877 \\
H_5(z) &= 1414z^2 + 275z + 877.
\end{aligned}$$

8. Next, we compute

$$v = [1065, 2188, 1529, 1371, 581, 2905]$$

where $v_i = \text{Coeff}(N_i, z^1) - \text{Coeff}(H_i, z^1)$ for $0 \leq i \leq 5$. Applying the BMA on v yields the feedback polynomial $\lambda(z) = z - 5$. Clearly, the root of $\lambda(z)$ is 5, so the corresponding monomial is y_3. Thus, $f_1 = 1065y_3$.

9. The same process can be repeated for the denominator part.

We note that it took 42 probes to the black box to reconstruct

$$A = f/g \equiv \frac{1065y_1^2 + 1065y_2^2 + 1065y_3^2 + 1065y_3}{1065y_1y_2y_3} \in \mathbb{Z}_{3137}(y_1, y_2, y_3).$$

Finally, we normalize A using 1065^{-1} to get our desired rational function $A = f/g$.

We emphasize again that the interpolation of the numerator and denominator polynomials can be done at the same time. Also, if we were to use Kaltofen and Yang's separation algorithm [Kaltofen and Yang, 2007] to interpolate $A(x, y, z)$, it would take 56 probes to the black box to reconstruct $A = f/g$.

Chapter 4

Modified Interpolation using a Kronecker Substitution

4.1 Summary of Contributions

All the materials presented in this chapter are new. Our main contribution in this chapter is a new sparse multivariate rational function interpolation method for interpolating a sparse multivariate rational function $A = f/g$ over $\mathbb{Q}$ represented by a black box which requires the same number of black box probes as the Cuyt and Lee's sparse multivariate rational function algorithm (previously described in Subsection 3.4.1).

This new sparse multivariate rational function interpolation method modifies the Cuyt and Lee's algorithm and the Ben-Or/Tiwari sparse polynomial interpolation algorithm to use a Kronecker substitution and a new set of randomized evaluation points.

The use of a Kronecker substitution in our new approach leads to a reduction in the size of the prime p required for our algorithm to work, and consequently transforms the problem of interpolating a multivariate rational function in $\mathbb{Z}_p(y_1, y_2, \ldots, y_m)$ to the problem of interpolating one univariate rational function in $\mathbb{Z}_p(y)$. We also randomize our new evaluation point sequence in order to avoid unlucky evaluation points with high probability.

Probabilistic algorithms for computing the degrees required to interpolate $A = f/g$ are also presented. We note that parts of this chapter have been published in the Proceedings of CASC '22 [Jinadu and Monagan, 2022b].

4.2 Introduction

Let $A = f/g$ be a sparse multivariate rational function over $\mathbb{Q}$ represented by a black box. In this thesis, we adapt the sparse multivariate rational function interpolation algorithm of Cuyt and Lee to interpolate A because it requires the fewest number of black box probes when combined with the Ben-Or/Tiwari algorithm as the primary sparse polynomial interpolation algorithm. However, the use of the Ben-Or/Tiwari algorithm poses the following two problems, namely,

(1) The working prime p needed to interpolate $f = \sum_{k=1}^{t} a_k M_k(y_1, \ldots, y_m) \in \mathbb{Z}[y_1, \ldots, y_m]$ over $\mathbb{Z}_p$ must satisfy $p > \max_{i=1}^{t} \hat{m}_i \leq p_m^{\deg(f)}$ so that the monomial evaluations $\hat{m}_i = M_i(2, 3, \ldots, p_m)$ are unique and can be uniquely determined where p_m is the m-th prime. Unfortunately, such a prime p may be too large for machine arithmetic use if the total degree d is large. This is the main drawback of using the Ben-Or/Tiwari algorithm point sequence $(2^i, 3^i, \ldots, p_m^i)$ for $i = 0, 1, 2, \ldots$.

Example 4.1. Suppose $m = 6$, $\deg(f, y_i) = 10$ and $\deg(f) = 60$ (the total degree of the Dixon resultant R of the robot arms system described in Example 1.10 is 128). Then $p > 13^{60} = 6.8 \times 10^{66}$ which is larger than a 64 bit prime.

(2) Using the Ben-Or/Tiwari algorithm point sequence $(2^j, 3^j, \ldots, p_n^j)$ can result in unlucky evaluation. We give the following example to help the reader understand what an unlucky evaluation point means.

Example 4.2. Let $a, b \in \mathbb{Z}[y_1, y_2, y_3]$. Suppose that $a = \bar{a}g$ and $b = \bar{b}g$ such that $\bar{a} \neq 0$ and $g = \gcd(a, b)$. Suppose $\bar{b} = \bar{a} + (y_2 - y_3)$. Consider the evaluation point $\beta = (2^0, 3^0)$. Suppose we want to compute $g(y_1, 1, 1)$. Observe that $\gcd(\bar{a}, \bar{b}) = 1$ but

$$\gcd(\bar{a}(y_1, 1, 1), \bar{b}(y_1, 1, 1)) = \bar{a}(y_1, 1, 1).$$

Thus, in order to use the fewest number of black box probes possible in our proposed Dixon resultant algorithm, by using the Ben-Or/Tiwari algorithm as the primary sparse polynomial algorithm in Cuyt and Lee's algorithm, these two problems must be addressed. We address these problems in the next section.

4.3 Using a Kronecker substitution on the parameters

Our goal now is to reduce the size of the prime needed in our sparse rational function interpolation algorithm. Our idea to reduce the size of the prime is based on using a Kronecker substitution to map a multivariate rational function into a univariate rational function, and then we evaluate at powers of a generator α of $\mathbb{Z}_p^*$ instead of powers of prime $(2^j, 3^j, \ldots, p_n^j)$.

To invert a Kronecker substitution, we first need to determine the partial degrees of f and g for all the variables involved in a multivariate rational function $A = f/g$. Thus, we first discuss how to compute the partial degrees of $A = f/g$ with high probability for all the variables involved in A.

4.3.1 Pre-computing the partial degrees of $A = f/g$ in each variable

Let $A = f/g \in \mathbb{Q}(y_1, y_2, y_3, \ldots, y_m)$ be a multivariate rational function in variables $y_1, \ldots, y_m$ represented by a black box $\mathbf{B}$. Let $d_{f_i} = \deg(f, y_i)$ and $d_{g_i} = \deg(g, y_i)$ be the partial degrees

of f and g in variables y_i respectively for $1 \leq i \leq m$. Let A be viewed as

$$A = f/g = \frac{\sum_{k=0}^{d_{f_i}} a_k(y_1, \ldots, y_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m) y_i^k}{\sum_{k=0}^{d_{g_i}} b_k(y_1, \ldots, y_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m) y_i^k} \tag{4.1}$$

such that $f, g \in \mathbb{Q}[y_1, y_2, \ldots, y_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m][y_i]$. Let p be a prime and let z be a new variable. Let $\alpha = (\alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_{i+1}, \ldots, \alpha_m) \in (\mathbb{Z}_p \setminus \{0\})^{m-1}$ be selected at random.

To obtain the partial degrees $\deg(f, y_i)$ and $\deg(g, y_i)$, we pick $\theta \in \mathbb{Z}_p \setminus \{0\}$ at random, and we use enough random distinct points (at least $d_{f_i} + d_{g_i} + 2$ points) for z selected from $\mathbb{Z}_p$ to interpolate the univariate rational function

$$H_i(z) := H_{f_i}/H_{g_i} = A(\alpha_1, \ldots, \alpha_{i-1}, \underbrace{\theta z}_{\text{the } i\text{-th component}}, \alpha_{i+1}, \cdots, \alpha_m) \in \mathbb{Z}_p(z)$$

such that $\deg(H_{f_i}, z) = d_{f_i}$ and $\deg(H_{g_i}, z) = d_{g_i}$ with high probability.

Algorithm 9 implements this approach and it uses Algorithm MQRFR (Algorithm 8) to discover the partial degrees of f and g with high probability. However, if $\mathrm{LC}(f, y_i)(\alpha) = 0$ or $\mathrm{LC}(g, y_i)(\alpha) = 0$, then the wrong partial degrees would be obtained. We give the following example to illustrate this.

Example 4.3. Let

$$A = f/g = \frac{(2 - y_3)y_1^2 y_2 + y_1}{y_1 + y_2}$$

and suppose we want to determine $\deg(f, y_1)$ and $\deg(g, y_1)$. Let $\alpha = (\alpha_2, \alpha_3)$, where $\alpha_j \neq 0$ for $j = 2, 3$ and prime $p = 3137$. Let z be a new variable. Clearly, we have

$$H_1(z) = H_{f_1}/H_{g_1} = A(\theta z, \alpha_2, \alpha_3) = \frac{(2 - \alpha_3)(\theta z)^2 \alpha_2 + \theta z}{\theta z + \alpha_2} = \frac{\alpha_2 \theta^2 (2 - \alpha_3) z^2 + \theta z}{\theta z + \alpha_2}.$$

Observe that if $\alpha_3 = 2$, then $\mathrm{LC}(f, y_1)(\alpha_2, 2) = (2 - 2)\alpha_2 = 0$, for any $\alpha_2 \in \mathbb{Z}_p \setminus \{0\}$. So, the wrong partial degree of f in y_1 will be returned in this case, because

$$H_1(z) = H_{f_1}/H_{g_1} = A(\theta z, \alpha_2, 2) = \frac{\theta z}{\theta z + \alpha_2}.$$

Since $\mathrm{LC}(f, y_i)(\alpha) = 0$ or $\mathrm{LC}(g,_i)(\alpha) = 0$ implies that $\mathrm{LC}(f)(\alpha)\mathrm{LC}(g)(\alpha) = 0$, we have that

$$\Pr[\mathrm{LC}(f, y_i)(\alpha)\mathrm{LC}(g, y_i)(\alpha) = 0] \leq \frac{\deg(\mathrm{LC}(f, y_i)) + \deg(\mathrm{LC}(g, y_i))}{p - 1} \leq \frac{\deg(f) + \deg(g)}{p - 1}$$

by the Schwartz-Zippel Lemma (Lemma 2.17). Thus, it is important that we pick prime $p \gg \deg(f) + \deg(g)$, and the evaluation point α randomly.

Given a black box $\mathbf{B}$ for $A = f/g \in \mathbb{Q}(y_1, y_2, \ldots, y_m)$ and an input prime p such that $p \gg \deg(f) + \deg(g)$, Algorithm 9 probes the black box $\mathbf{B}$ with evaluation points

$(\alpha_1, \ldots, \alpha_{i-1}, \theta\gamma_d, \alpha_{i+1}, \ldots, \alpha_m)$ to output the black box evaluations

$$a_d = A(\alpha_1, \ldots, \alpha_{i-1}, \theta\gamma_d, \alpha_{i+1}, \ldots, \alpha_m)$$

in Line 6 for $d = 1, 2, \ldots$ in order to interpolate the unique polynomial $u \in \mathbb{Z}_p[z]$ of degree $< d$ and to compute the product polynomial $\overline{m}(z) = \prod_{j=1}^{d}(z - \gamma_j) \in \mathbb{Z}_p[z]$. It then attempts to discover the partial degrees $\deg(f, y_i)$ and $\deg(g, y_i)$ for any i such that $1 \leq i \leq m$ using the polynomials $\overline{m}(z)$ and $u(z)$ as inputs to the Algorithm MQRFR in Line 11. Algorithm 9 breaks out of the main for loop (Line 4-18) and returns the partial degrees with high probability if the output of Algorithm MQRFR in Line 12 labelled as $h(z) \neq$ FAIL.

Algorithm 9: PartialDegreeBound

Input: A prime $p \gg \deg(f) + \deg(g)$ and the black box $\mathbf{B}\left(\mathbb{Z}_p^m, p\right) \to \mathbb{Z}_p$ for the rational function $\frac{f(y_1, y_2, \ldots, y_m)}{g(y_1, y_2, \ldots, y_m)}$ over $\mathbb{Q}$ which returns *"division by zero"* if $g(\gamma) = 0$ for some point $\gamma \in \mathbb{Z}_p^m$ and i such that $1 \leq i \leq m$.

Output: $(\deg(\bar{f}), \deg(\bar{g}))$ where $\deg(\bar{f}) = \deg(f, y_i)$ and $\deg(\bar{g}) = \deg(g, y_i)$ with high probability.

```
1  Pick α = (α_1, ..., α_{i-1}, α_{i+1}, ..., α_m) ∈ (Z_p \ {0})^{m-1} and θ ∈ Z_p \ {0} at random.
2  Let z be a new variable.
3  m̄(z) ← 1
4  for d = 1, 2, ··· do
5      Pick γ_d ∈ Z_p at random such that γ_d ≠ γ_j for 1 ≤ j < d.
6      a_d ← B((α_1, ..., α_{i-1}, θγ_d, α_{i+1}, ···, α_m), p) ∈ Z_p. // Next black box evaluation.
7      if a_d = "division by zero" then restart the algorithm (go to line 1) end.
8      if d ≥ 2 then
9          Interpolate the polynomial u(z) ∈ Z_p[z] of degree < d using points
             (γ_i, a_i : 1 ≤ i ≤ d).
10         Compute m̄(z) = (z − γ_i) × m̄(z) ∈ Z_p[z]. // m̄(z) = ∏_{i=1}^{d}(z − γ_i) ∈ Z_p[z].
11         h(z) ← MQRFR (m̄, u, p). // d = deg(m̄) > deg(u) ≥ 0.
12         if h(z) ≠ FAIL then
13             Let h(z) = f̄(z)/ḡ(z) with gcd(f̄, ḡ) = 1.
14             return (deg(f̄, ) deg(ḡ))
15         end
16     end
17     d ← d + 1
18 end
```

We remark that if $\mathrm{LC}(f, y_i)(\alpha) \neq 0$ and $\mathrm{LC}(g, y_i)(\alpha) \neq 0$, and the black box for $A = f/g$ does not output *"division by zero"*, then Algorithm 9 computes the partial degrees of f and g in variable y_i for $1 \leq i \leq m$ with high probability (See Section 4.6 for a failure probability bound).

4.3.2 Kronecker substitution

Using a Kronecker substitution in Cuyt and Lee's method, we reduce the problem of interpolating a sparse multivariate rational function to many univariate rational function interpolations.

Definition 4.4. Let $\mathbb{K}$ be an integral domain and let $A = f/g \in \mathbb{K}(y_1, \ldots, y_m)$. Let $r = (r_1, r_2, \ldots, r_{m-1}) \in \mathbb{Z}^{m-1}$ with $r_i > 0$. Let $K_r : \mathbb{K}(y_1, \ldots, y_m) \to \mathbb{K}(y)$ be the Kronecker substitution

$$K_r(A) = \frac{f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}{g(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}.$$

Let $d_i = \max\{(\deg(f), y_i), \deg(g, y_i)\}$ for $1 \leq i \leq m$. Provided we choose $r_i > d_i$ for $1 \leq i \leq m-1$, then K_r is invertible, $g \neq 0$ and $K_r(A) = 0 \iff f = 0$.

Unfortunately, we cannot use the original presentation and definition of the auxiliary rational function given by Cuyt and Lee to interpolate the univariate mapped function $K_r(A)$. Thus, we need a new method to interpolate the corresponding auxiliary rational function relative to the mapped univariate rational function $K_r(A)$, and not the original sparse multivariate rational function $A = f/g$.

Using a new variable z, we define our new auxiliary rational function

$$F(y, z) = \frac{f(zy, zy^{r_1}, \ldots, zy^{r_1 r_2 \cdots r_{m-1}})}{g(zy, zy^{r_1}, \ldots, zy^{r_1 r_2 \cdots r_{m-1}})} \in \mathbb{K}[y](z). \tag{4.2}$$

We need to guarantee the existence of a constant term in the denominator of $F(y, z)$. So, we use a basis shift $\beta \in (\mathbb{K} \setminus \{0\})^m$, and instead formally define an auxiliary rational function with a Kronecker substitution as follows.

Definition 4.5. Let $\mathbb{K}$ be a field and let $f/g \in \mathbb{K}(y_1, \ldots, y_m)$ such that $\gcd(f, g) = 1$. Let $r = (r_1, \ldots, r_{m-1})$ with $r_i > d_i = \max\{(\deg(f), y_i), \deg(g, y_i)\}$. Let z be the homogenizing variable and let K_r be the Kronecker substitution. Let $\beta \in \mathbb{K}^m$ be a basis shift. We define

$$F(y, z, \beta) := \frac{f^\beta(y, z)}{g^\beta(y, z)} = \frac{f(zy + \beta_1, zy^{r_1} + \beta_2, \ldots, zy^{r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(zy + \beta_1, zy^{r_1} + \beta_2, \ldots, zy^{r_1 r_2 \cdots r_{m-1}} + \beta_m)} \in \mathbb{K}[y](z)$$

as an auxiliary rational function with a Kronecker substitution K_r.

We will often refer to $F(y, z, \beta)$ simply as an auxiliary rational function.

Notice in the above definition that for $\beta = 0$,

$$F(y, 1, 0) = \frac{f^0(y, 1)}{g^0(y, 1)} = K_r(A).$$

Thus, the univariate rational function $K_r(A)$ can be recovered using the coefficients of z^i in $F(\alpha^i, z, \beta)$ for some evaluation point $\alpha \in \mathbb{Z}_p^*$ and $i \geq 0$. If g has a constant term, then one can use $\beta = (0, \ldots, 0)$.

Although the degree of y of the mapped univariate rational function $K_r(A)$ is exponential in m, the degree of the auxiliary function $F(y,z)$ in z through which the univariate rational function $K_r(A)$ is interpolated remains the same. Consequently, the number of terms and the number of probes needed to interpolate $A = f/g$ does not change. To uniquely recover the exponents in y and to also make our discrete logarithm computations in $\mathbb{Z}_p^*$ feasible, we follow [Kaltofen, 2010] and pick a prime $p > \prod_{j=1}^{m} r_j$ such that $p-1 = 2^k s$ where s is small. Any prime p of the form $p-1 = 2^k s$ where s is small is said to be a smooth prime. A prime p is said to be y-smooth if the largest prime factor of $p-1$ is at most y.

Example 4.6. In Example 4.1, $m = 6, \deg(f, y_i) = 10$, so, using a Kronecker substitution now implies that $p > 11^6 = 1.7 \times 10^6$ which is small.

4.3.3 Randomizing the evaluation point sequence

Let p be a prime. Since we map a multivariate rational function $A = f/g$ in variables $y_1, y_2, \ldots, y_m$ to become a univariate rational function $K_r(A) \in \mathbb{Z}_p(y)$ using a Kronecker substitution K_r, we are now tasked to interpolate univariate polynomials in $\mathbb{Z}_p[y]$.

Let $H = \sum_{j=1}^{t} a_j M_j(y)$ be one of the univariate polynomials to be interpolated in either the numerator or denominator of $K_r(A)$. We avoid unlucky evaluation point with high probability for our purposes, by randomizing the evaluation points α^j for $j \geq 0$. The implication of doing this in our new sparse rational function method is that we do not want to lose the total degree of the numerator and the denominator of the univariate auxiliary rational functions in z with high probability. This modification is done as follows.

We pick a random shift $\hat{s} \in [0, p-2]$ and compute $v_j = H(\alpha^{\hat{s}+j})$ for $0 \leq j \leq t-1$. Changing the point sequence from α^j to $\alpha^{\hat{s}+j}$ does not affect the way we recover the univariate monomials M_i in y of H using our new approach. However, solving for the coefficients a_i means we now have to solve the shifted transposed Vandermonde system [Hu and Monagan, 2016]

$$Va = \begin{bmatrix} \hat{m}_1^{\hat{s}} & \hat{m}_2^{\hat{s}} & \cdots & \hat{m}_t^{\hat{s}} \\ \hat{m}_1^{\hat{s}+1} & \hat{m}_2^{\hat{s}+1} & \cdots & \hat{m}_t^{\hat{s}+1} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{\hat{s}+t-1} & \hat{m}_2^{\hat{s}+t-1} & \cdots & \hat{m}_t^{\hat{s}+t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v,$$

where $\hat{m}_i^j = M_i(\alpha^j)$. We solve for the coefficients a_i by first using Zippel's $O(t^2)$ algorithm [Zippel, 1990] to solve the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \hat{m}_1 & \hat{m}_2 & \cdots & \hat{m}_t \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{t-1} & \hat{m}_2^{t-1} & \cdots & \hat{m}_t^{t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v,$$

which yields $c = W^{-1}v$. Notice that

$$V = WD = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \hat{m}_1 & \hat{m}_2 & \cdots & \hat{m}_t \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{t-1} & \hat{m}_2^{t-1} & \cdots & \hat{m}_t^{t-1} \end{bmatrix} \begin{bmatrix} \hat{m}_1^{\hat{s}} & 0 & \cdots & 0 \\ 0 & \hat{m}_2^{\hat{s}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \hat{m}_t^{\hat{s}} \end{bmatrix}$$

where D is a $t \times t$ diagonal matrix with entries $D_{ii} = \hat{m}_i^{\hat{s}}$. Thus, we obtain the unknown coefficients a_i using $a_i = \hat{m}_i^{-\hat{s}} c_i$ since

$$Va = v \Longrightarrow (WD)a = v \Longrightarrow (Da) = W^{-1}v = c \Longrightarrow a = D^{-1}c.$$

Note that the number of arithmetic operations performed in $\mathbb{Z}_p$ to solve a shifted transposed Vandermonde system is $O(t^2 + t\log(\hat{s}))$ where $\log(\hat{s})$ is the cost of inverting $\hat{m}_i^{\hat{s}}$. Therefore, for our new sparse rational function interpolation method which uses a Kronecker substitution, we randomize our points by picking any generator $\alpha \in \mathbb{Z}_p^*$, and a random shift $\hat{s} \in [0, p-2]$ where p is prime to compute

$$F(\alpha^{\hat{s}+i}, z, \beta) = \frac{f^{\beta}(\alpha^{\hat{s}+i}, z)}{g^{\beta}(\alpha^{\hat{s}+i}, z)} = \frac{f(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)} \in \mathbb{Z}_p(z)$$

for $i = 0, 1, 2, \cdots$. Randomizing our evaluation points ensures that $\deg(f^{\beta}(\alpha^{\hat{s}+i}, z)) = \deg(f)$ and $\deg(g^{\beta}(\alpha^{\hat{s}+i}, z)) = \deg(g)$ with high probability.

4.4 An illustrative example of our new sparse rational function interpolation method

We demonstrate how our new sparse rational function interpolation method works with the following example. The new algorithm is presented in Section 4.5. We advise the reader to first go through this example before checking the steps involved in the new algorithm.

Suppose we are given a black box for the sparse multivariate rational function

$$A = f/g = \frac{y_1^{11} + y_2^{11} + y_3^{11} + y_4^5 + y_5^5 + y_8}{y_6^{11} + y_7^{11} + y_8^{11} + y_6} \in \mathbb{Z}(y_1, y_2, \ldots, y_8),$$

and we want to interpolate A. Let

$f_{11} = y_1^{11} + y_2^{11} + y_3^{11}, f_5 = y_4^5 + y_5^5, f_1 = y_8$ and $g_{11} = y_6^{11} + y_7^{11} + y_8^{11}$ and $g_1 = y_6$.

So, $f = f_{11} + f_5 + f_1$ and $g = g_{11} + g_1$.

Suppose we have discovered the total degrees of f and g denoted by $\deg(f) = \deg(g) = 11$ with high probability (we will discuss how to do this in Subsection 4.4.1), and the maximum

partial degrees of f and g in each variable denoted by d_i for $1 \leq i \leq 8$ have been discovered using Algorithm 9. Then we use $r = (d_1+1, d_2+1, \ldots, d_8+1) = (12, 12, 12, 6, 6, 12, 12, 12)$.

For our new sparse rational function interpolation method, we use a smooth prime $p = 3 \cdot 2^{30}+1 > \prod_{i=1}^{8} r_i$ whereas the Ben-Or/Tiwari algorithm requires $p > 19^{11} = 1.1 \times 10^{14}$. We use a Kronecker substitution $K_r : \mathbb{Z}_p(y_1, \ldots, y_m) \to \mathbb{Z}_p(y)$ where

$$K_r(A) = \frac{f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}{g(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})} = \frac{y^{8957952} + y^{51840} + y^{8640} + y^{1584} + y^{132} + y^{11}}{y^{98537472} + y^{8211456} + y^{684288} + y^{62208}}.$$

So, $K_r(A)$ is what we want to interpolate. Suppose we pick $\alpha = 5$ which is a generator for $\mathbb{Z}_p^*$ and let $\hat{s} = 2 \in [0, p-2]$ be picked at random. Let the random basis shift be $\beta = (85132797, 34637621, 26107038, 10607916, 37522487, 4922352, 51780437, 52449057)$.

Step 1: Using the Berlekamp-Massey algorithm (BMA), suppose we know that the (minimum) number of auxiliary rational functions needed to interpolate $K_r(A)$ is 6, and suppose we directly compute (one must interpolate these univariate rational functions because the black box only accepts list of integers and a prime. We also do not know what A is) the auxiliary rational functions

$$F(\alpha^{\hat{s}+i}, z, \beta) = \frac{f^{\beta}(\alpha^{\hat{s}+i}, z)}{g^{\beta}(\alpha^{\hat{s}+i}, z)} = \frac{f(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)} \in \mathbb{Z}_p(z)$$

for $0 \leq i \leq 5$, so that

$$\begin{aligned}
f^{\beta}(\alpha^{\hat{s}+0}, z) &= 504456095z^{11} + 1271576935z^{10} + 2894674426z^9 + 1922536865z^8 + 2566228163z^7 \\
&\quad + 2781003124z^6 + 2961517103z^5 + 2105161407z^4 + 2174550986z^3 + 2221014634z^2 \\
&\quad + 1642140628z + 2484490329 \\
f^{\beta}(\alpha^{\hat{s}+1}, z) &= 1799751883z^{11} + 51851774z^{10} + 1692905233z^9 + 1926634179z^8 + 99438716z^7 \\
&\quad + 1281205593z^6 + 1758033894z^5 + 2591331267z^4 + 147591147z^3 + 1020284629z^2 \\
&\quad + 375656678z + 2484490329 \\
f^{\beta}(\alpha^{\hat{s}+2}, z) &= 690622067z^{11} + 2722183515z^{10} + 1348427258z^9 + 1220429635z^8 + 1781861026z^7 \\
&\quad + 113355454z^6 + 2572143706z^5 + 1003239085z^4 + 1984454376z^3 + 267259118z^2 \\
&\quad + 1976525844z + 2484490329 \\
f^{\beta}(\alpha^{\hat{s}+3}, z) &= 2531520139z^{11} + 875452800z^{10} + 506750217z^9 + 2482068730z^8 + 1109636639z^7 \\
&\quad + 1808481201z^6 + 1953791611z^5 + 1567806275z^4 + 2686077811z^3 + 1383825822z^2 \\
&\quad + 1185059664z + 2484490329
\end{aligned}$$

$$
\begin{aligned}
f^{\beta}(\alpha^{\hat{s}+4}, z) &= 1902062431z^{11} + 1144884717z^{10} + 1283699848z^{9} + 1928785662z^{8} + 1621964187z^{7} \\
&\quad + 1791281689z^{6} + 2892449471z^{5} + 2166912829z^{4} + 2462053546z^{3} + 199748103z^{2} \\
&\quad + 2233025122z + 2484490329 \\
f^{\beta}(\alpha^{\hat{s}+5}, z) &= 3066602361z^{11} + 850554450z^{10} + 1398158196z^{9} + 1025235762z^{8} + 192614344z^{7} \\
&\quad + 1797193281z^{6} + 1209785003z^{5} + 2020073210z^{4} + 1967922306z^{3} + 2214513559z^{2} \\
&\quad + 772210495z + 2484490329 \\
g^{\beta}(\alpha^{\hat{s}+0}, z) &= 1580679741z^{11} + 3116990146z^{10} + 1057902518z^{9} + 3198338027z^{8} + 286841619z^{7} \\
&\quad + 2890513996z^{6} + 1809795283z^{5} + 2072664216z^{4} + 1193173354z^{3} + 2003803474z^{2} \\
&\quad + 2713215529z + 1 \\
g^{\beta}(\alpha^{\hat{s}+1}, z) &= 46861898z^{11} + 2102486493z^{10} + 2997800097z^{9} + 1246736249z^{8} + 411444783z^{7} \\
&\quad + 1280104659z^{6} + 1416745533z^{5} + 2733211723z^{4} + 2806481616z^{3} + 2308132560z^{2} \\
&\quad + 2788585009z + 1 \\
g^{\beta}(\alpha^{\hat{s}+2}, z) &= 298845476z^{11} + 2898326650z^{10} + 3005507166z^{9} + 275950621z^{8} + 897554295z^{7} \\
&\quad + 2246083541z^{6} + 1104481969z^{5} + 1029137782z^{4} + 1858508920z^{3} + 1918855279z^{2} \\
&\quad + 2704579471z + 1 \\
g^{\beta}(\alpha^{\hat{s}+3}, z) &= 1565430745z^{11} + 2205463094z^{10} + 1710958486z^{9} + 2563730374z^{8} + 2292970312z^{7} \\
&\quad + 1336214124z^{6} + 2211218900z^{5} + 2783288643z^{4} + 2930129794z^{3} + 1176308200z^{2} \\
&\quad + 3036359035z + 1 \\
g^{\beta}(\alpha^{\hat{s}+4}, z) &= 46844537z^{11} + 2791209780z^{10} + 429168896z^{9} + 99940710z^{8} + 1386582108z^{7} \\
&\quad + 1060098921z^{6} + 2458077644z^{5} + 2171712250z^{4} + 48510025z^{3} + 653163610z^{2} \\
&\quad + 3002106356z + 1 \\
g^{\beta}(\alpha^{\hat{s}+5}, z) &= 30453847z^{11} + 2861695808z^{10} + 3122487314z^{9} + 910916656z^{8} + 148650945z^{7} \\
&\quad + 2086024934z^{6} + 344222140z^{5} + 1261916191z^{4} + 683211027z^{3} + 2211776667z^{2} \\
&\quad + 1000246311z + 1
\end{aligned}
$$

For the sake of brevity, suppose we only want to interpolate f. Since $\deg(f) = 11$, we will attempt to interpolate all homogeneous polynomials f_k in f of degrees $k = 11, 10, 9, 8, \ldots, 0$, in that order using the coefficients of $f^{\beta}(\alpha^{\hat{s}+i}, z)$.

Step 2: First, we check if $\deg(f^{\beta}(\alpha^{\hat{s}+i}, z)) = 11$ for all i, and then we collect the leading coefficient sequence

$$
v = [504456095, 1799751883, 690622067, 2531520139, 1902062431, 3066602361]\,,
$$

such that $v_i = \text{LC}(f^{\beta}(\alpha^{\hat{s}+i}, z))$ for $0 \leq i \leq 5$. Next, we run the BMA on v which generates the feedback polynomial

$$\lambda_{11}(z) = z^3 + 1905095726z^2 + 2338018633z + 1009320183 \in \mathbb{Z}_p[z].$$

Step 3: Computing the roots of $\lambda_{11}(z)$ yields the monomial evaluations

$$\hat{m} = \{48828125, 1303095659, 3185431436\}.$$

Using the baby step method due to Shanks [Shanks, 1971] and the Pohlig-Helman algorithm [Pohlig and Martin, 1978], we solve the discrete logarithms $\{5^{e_1} = 48828125, 5^{e_2} = 1303095659, 5^{e_3} = 3185431436\}$ in $\mathbb{Z}_p^*$ to obtain the exponents $\{e_1 = 11, e_2 = 132, e_3 = 1584\}$. Thus, the corresponding monomials in y are $\{y^{11}, y^{132}, y^{1584}\}$.

Step 4: Now we solve for the coefficients of the monomials in y that we just found. Since we only have three univariate monomials, we only need the first three entries from v to solve for their coefficients. To do this, we set up the 3×3 shifted transposed Vandermonde system

$$Va = \begin{bmatrix} \hat{m}_1^{\hat{s}+0} = 2199625621 & \hat{m}_2^{\hat{s}+0} = 2208617479 & \hat{m}_3^{\hat{s}+0} = 2865126349 \\ \hat{m}_1^{\hat{s}+1} = 406005153 & \hat{m}_2^{\hat{s}+1} = 1548294981 & \hat{m}_3^{\hat{s}+1} = 305961711 \\ \hat{m}_1^{\hat{s}+2} = 1902207400 & \hat{m}_2^{\hat{s}+2} = 676383069 & \hat{m}_3^{\hat{s}+2} = 1310850391 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = v$$

where

$$v = \begin{bmatrix} 504456095 \\ 1799751883 \\ 690622067 \end{bmatrix}$$

and then solve for the coefficients a_i. As explained in Subsection 4.3.3, this can be done by first solving the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 & 1 \\ \hat{m}_1 = 48828125 & \hat{m}_2 = 67242306 & \hat{m}_3 = 86401899 \\ \hat{m}_1^2 = 2199625621 & \hat{m}_2^2 = 2208617479 & \hat{m}_3^2 = 2865126349 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = v$$

using Zippel's quadratic algorithm. Solving $Wc = v$ yields

$$\{c_1 = 1679199919, c_2 = 545880638, c_3 = 1500601011\},$$

and then we compute $a_i = \dfrac{c_i}{m_i^{\hat{s}}}$ for $1 \leq i \leq 3$. The a_i's are

$$a_1 = 2686038870, a_2 = 2686038870, a_3 = 2686038870.$$

Step 5: Now we invert the Kronecker map K_r using base conversion to obtain

$$y^{11} \mapsto y_1, y^{132} \mapsto y_2^{11}, \text{ and } y^{1584} \mapsto y_3^{11}.$$

Thus, the highest degree homogeneous polynomial of degree 11 in f is

$$f_{11} = 2686038870y_1^{11} + 2686038870y_2^{11} + 2686038870y_3^{11}.$$

Step 6: Next, we move on to interpolate f_{10} (a homogeneous polynomial of degree 10 in f) using the coefficients of z^{10} in $f^{\beta}(\alpha^{\hat{s}+i}, z)$. However, we remind the reader that the coefficients of z^{10} in $f^{\beta}(\alpha^{\hat{s}+i}, z)$ for all i have to be adjusted in order to remove the effect of the basis shift. Thus, computing the univariate polynomials

$$H_i(z) = f_{11}(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1r_2\cdots r_{m-1}} + \beta_m)$$

directly for $i = 0, 1, 2, \cdots, 5$, (these univariate polynomials must be interpolated to avoid expression swell). We obtain

$$\begin{aligned}
H_0(z) &= 504456095z^{11} + 1271576935z^{10} + 2894674426z^9 + 1922536865z^8 + 2566228163z^7 \\
&\quad + 2781003124z^6 + 1711435757z^5 + 1891039209z^4 + 2684787759z^3 + 3036780286z^2 \\
&\quad + 224593306z + 2783755731 \\
H_1(z) &= 1799751883z^{11} + 51851774z^{10} + 1692905233z^9 + 1926634179z^8 + 99438716z^7 \\
&\quad + 1281205593z^6 + 2790993731z^5 + 1678491156z^4 + 2884452674z^3 + 2128925856z^2 \\
&\quad + 1058209323z + 2783755731 \\
H_2(z) &= 690622067z^{11} + 2722183515z^{10} + 1348427258z^9 + 1220429635z^8 + 1781861026z^7 \\
&\quad + 113355454z^6 + 1833567179z^5 + 2795550342z^4 + 679811099z^3 + 2271350410z^2 \\
&\quad + 2609405332z + 2783755731 \\
H_3(z) &= 2531520139z^{11} + 875452800z^{10} + 506750217z^9 + 2482068730z^8 + 1109636639z^7 \\
&\quad + 1808481201z^6 + 1330250644z^5 + 2885672376z^4 + 1641102280z^3 + 2927081373z^2 \\
&\quad + 3020245364z + 2783755731 \\
H_4(z) &= 1902062431z^{11} + 1144884717z^{10} + 1283699848z^9 + 1928785662z^8 + 1621964187z^7 \\
&\quad + 1791281689z^6 + 2899006778z^5 + 1953636377z^4 + 2325234854z^3 + 1116772202z^2 \\
&\quad + 2665504468z + 2783755731 \\
H_5(z) &= 3066602361z^{11} + 850554450z^{10} + 1398158196z^9 + 1025235762z^8 + 192614344z^7 \\
&\quad + 1797193281z^6 + 1018282945z^5 + 2140833910z^4 + 2250676795z^3 + 2314588079z^2 \\
&\quad + 1320761667z + 2783755731
\end{aligned}$$

Step 7: To interpolate a possible degree 10 polynomial in f using the coefficients of z^{10} in both $H_i(z)$ and $f^{\beta}(\alpha^{s+i}, z)$, we compute $v = [0, 0, 0, 0, 0, 0]$ where

$$v_i = \text{Coeff}(f^{\beta}(\alpha^{s+i}, z), z^{10}) - \text{Coeff}(H_i(z), z^{10}) \quad \text{for } 0 \leq i \leq 5.$$

The above computation is the coefficient adjustment that must be done to correct the contributions due to the basis shift β. Unfortunately, we do not have a homogeneous polynomial of degree 10 in f since all the entries of v are zero. Observe that the sequence of coefficients of z^j in $H_i(z)$ are the same as the sequence of coefficients of z^j in $f^{\beta}(\alpha^{\hat{s}+i}, z)$ for $j = 9, 8, 7, 6$, So,

$$v_i = \text{Coeff}(f^{\beta}(\alpha^{\hat{s}+i}, z), z^j) - \text{Coeff}(H_i(z), z^j) = 0, \quad \text{for } 0 \leq i \leq 5.$$

This implies that there are no homogeneous polynomials of degree $9, \ldots, 6$ in f.

Step 8: Now, we will attempt to interpolate any possible homogeneous polynomial of total degree 5 in f. Observe that

$$v = [1250081346, 2188265636, 738576527, 623540967, 3214668166, 191502058]$$

where $v_i = \text{Coeff}(f^{\beta}(\alpha^{\hat{s}+i}, z), z^5) - \text{Coeff}(H_i(z), z^5) \quad \text{for } 0 \leq i \leq 5$.

Step 9: Next, we run the BMA on v which generates the feedback polynomial

$$\lambda_5(z) = z^2 + 744046774z + 2377407692 \in \mathbb{Z}_p[z].$$

Computing the roots of $\lambda_5(z)$ yields the monomial evaluations $\hat{m} = \{2469640426, 7538273\}$.

Step 10: Next, we solve the discrete logarithms $\{5^{e_1} = 2469640426, 5^{e_2} = 7538273\}$ in $\mathbb{Z}_p^*$ to obtain the exponents $\{e_1 = 51840, e_2 = 8640\}$. Thus, the corresponding monomials in y are $\{y^{51840}, y^{8640}\}$.

Step 11: Based on the two univariate monomials that we found, we only need the first two entries from v to solve for their coefficients. Next, we set up the 2×2 shifted transposed Vandermonde system

$$Va = \begin{bmatrix} m_1^{\hat{s}+0} = 2113008848 & m_2^{\hat{s}+0} = 3142478809 \\ m_1^{\hat{s}+1} = 2617252129 & m_2^{\hat{s}+1} = 21544114 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1250081346 \\ 2188265636 \end{bmatrix} = v,$$

in order to solve for their coefficients a_i. But first, we solve the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 \\ \hat{m}_1 = 2469640426 & \hat{m}_2 =, 7538273 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1250081346 \\ 2188265636 \end{bmatrix} = v$$

using Zippel's quadratic algorithm to get $\{c_1 = 1019250191, c_2 = 230831155\}$. Then we compute $a_i = \frac{c_i}{m_i^{\hat{s}}}$ for $1 \leq i \leq 2$. The a_i's are $a_1 = 2686038870, a_2 = 2686038870$.

Step 12: Inverting the Kronecker map K_r yields

$$y^{51840} \mapsto y_5^5, y^{8640} \mapsto y_4^5.$$

Thus

$$f_5 = 2686038870y_4^5 + 2686038870y_5^5.$$

Step 13: Before we attempt to interpolate any possible polynomial of total degree 4 in f, we must first update the H polynomials by computing

$$H_i(z) = H_i(z) + f_5(z\alpha^{s+i} + \beta_1, z\alpha^{(s+i)r_1} + \beta_2, \ldots, z\alpha^{(s+i)r_1r_2\cdots r_{m-1}} + \beta_m)$$

directly for $i = 0, 1, 2, \cdots, 5$, because of the contributions by the polynomials f_{11} and f_5 due to the basis shift β. We get

$$\begin{aligned}
H_0(z) &= 504456095z^{11} + 1271576935z^{10} + 2894674426z^9 + 1922536865z^8 + 2566228163z^7 \\
&\quad + 2781003124z^6 + 2961517103z^5 + 2105161407z^4 + 2174550986z^3 + 2221014634z^2 \\
&\quad + 2880688883z + 2554986495 \\
H_1(z) &= 1799751883z^{11} + 51851774z^{10} + 1692905233z^9 + 1926634179z^8 + 99438716z^7 \\
&\quad + 1281205593z^6 + 1758033894z^5 + 2591331267z^4 + 147591147z^3 + 1020284629z^2 \\
&\quad + 3614696z + 2554986495 \\
H_2(z) &= 690622067z^{11} + 2722183515z^{10} + 1348427258z^9 + 1220429635z^8 + 1781861026z^7 \\
&\quad + 113355454z^6 + 2572143706z^5 + 1003239085z^4 + 1984454376z^3 + 267259118z^2 \\
&\quad + 223656964z + 2554986495 \\
H_3(z) &= 2531520139z^{11} + 875452800z^{10} + 506750217z^9 + 2482068730z^8 + 1109636639z^7 \\
&\quad + 1808481201z^6 + 1953791611z^5 + 1567806275z^4 + 2686077811z^3 + 1383825822z^2 \\
&\quad + 80104098z + 2554986495 \\
H_4(z) &= 1902062431z^{11} + 1144884717z^{10} + 1283699848z^9 + 1928785662z^8 + 1621964187z^7 \\
&\quad + 1791281689z^6 + 2892449471z^5 + 2166912829z^4 + 2462053546z^3 + 199748103z^2 \\
&\quad + 179979030z + 2554986495 \\
H_5(z) &= 3066602361z^{11} + 850554450z^{10} + 1398158196z^9 + 1025235762z^8 + 192614344z^7 \\
&\quad + 1797193281z^6 + 1209785003z^5 + 2020073210z^4 + 1967922306z^3 + 2214513559z^2 \\
&\quad + 2891090635z + 2554986495
\end{aligned}$$

Step 14: For the sake of brevity, we note that the sequence of coefficients of z^j in $H_i(z)$ are the same as the sequence of coefficients of z^j in $f^\beta(\alpha^{\hat{s}+i}, z)$ for $j = 4, 3, 2$. So there are no polynomial terms of degree $4, 3, 2$ in f.

Step 15: To determine a possible polynomial of degree 1 in f, we compute

$$v = [1982677218, 372041982, 1752868880, 1104955566, 2053046092, 1102345333]$$

where $v_i = \text{Coeff}(f^\beta(\alpha^{\hat{s}+i}, z), z^1) - \text{Coeff}(H_i(z), z^1) \quad \text{for } 0 \leq i \leq 5$. Applying the Berlekamp-Massey Algorithm to v generates the feedback polynomial

$$\lambda_1(z) = 460205760 + z \in \mathbb{Z}_p[z].$$

Step 16: Computing the roots of $\lambda_1(z)$ yields the monomial evaluation $\hat{m} = \{2761019713\}$. Next, we solve the discrete logarithms $\{5^{e_1} = 2761019713\}$ to obtain the exponent $\{e_1 = 8957952\}$. Thus, the corresponding monomial in y is $y^{8957952}$. Next, we set up the shifted transposed Vandermonde system

$$Va = \left[\ m_1^{\hat{s}} = 2761019713 \ \right] \Big[a_1\Big] = \Big[1982677218\Big] = v.$$

Solving the above system yields $a = 2686038870$ and inverting the Kronecker map yields $f_1 = 2686038870 y_8$.

Step 17: Next we update the H polynomials for $0 \leq i \leq 5$ by computing

$$H_i(z) = H_i(z) + f_1(z\alpha^{s+i} + \beta_1, z\alpha^{(s+i)r_1} + \beta_2, \ldots, z\alpha^{(s+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)$$

in order to interpolate f_0 of f. Fortunately, there is no constant term in f.

Step 18: We repeat the same process for the denominator part and we are done.

It took $144 = 6 \times (11 + 11 + 2)$ probes to the black box to reconstruct

$$A = f/g \equiv \frac{2686038870\left(y_1^{11} + y_2^{11} + y_3^{11} + y_4^5 + y_5^5 + y_8\right)}{2686038870\left(y_6^{11} + y_7^{11} + y_8^{11} + y_6\right)} \in \mathbb{Z}_p(y_1, y_2, \ldots, y_8).$$

Finally, we multiply the numerator and denominator by 2686038870^{-1} to get A.

Remark 4.7. Suppose

$$f = \sum_{i=0}^{\deg(f)} f_i(y_1, \ldots, y_m) \text{ and } g = \sum_{j=0}^{\deg(g)} g_j(y_1, \ldots, y_m)$$

where f_i and g_j are homogeneous polynomials and $\deg(f_i) = i$ and $\deg(g_j) = j$. Based on the above example, we make the following two important remarks.

(1) Notice that the leading degree polynomials $f_{\deg(f)}$ and $g_{\deg(g)}$ in f and g respectively are the first polynomials to be interpolated using the leading coefficients of the auxiliary rational functions in z, before we interpolate the lower total degree polynomials $f_{\deg(f)-1}, f_{\deg(f)-2}, f_{\deg(f)-3}, \cdots f_0$ and $g_{\deg(g)-1}, g_{\deg(g)-2}, g_{\deg(g)-3}, \cdots g_0$ in that order. However, if f and g are very sparse, but the number of auxiliary rational functions computed to interpolate $f_{\deg(f)}$ or $g_{\deg(f)}$ is high, then a lot of unnecessary computation will be done in an attempt to perform coefficient adjustments in order to interpolate non-existent polynomials. This coefficient adjustment computation becomes particularly expensive when interpolating a large number of multivariate rational function coefficients (Our proposed Dixon resultant algorithm).

Example 4.8. Suppose

$$f = \sum_{i=0}^{\deg(f)} f_i(y_1, y_2, \ldots, y_m) = f_{100000} + f_0,$$

where $\deg(f) = 100000$. After interpolating f_{100000}, one will attempt to interpolate polynomials $f_{99999}, f_{99998}, \ldots, f_0$, whereas the only polynomial that needs to be interpolated is the constant term f_0. This is because f is represented by a black box, and we do not know the total degrees of the polynomials f_i and g_i beforehand. We will address this issue in Subsection 4.4.2.

(2) Since $A = f/g$ is represented by a black box, we do not know the size of the supports of the polynomials f_i and g_i. Let $\#f_i$ and $\#g_i$ denote the size of the supports of the polynomials f_i and g_i respectively. By design, we discover $\#f_{\deg(f)}$ and $\#g_{\deg(g)}$ first, by inputting the sequence of leading coefficients from $f^\beta(\alpha^{s+i}, z)$ and $g^\beta(\alpha^{s+i}, z)$ respectively from the auxiliary rational functions $F(\alpha^{s+i}, z, \beta)$ for $i = 0, 1, \ldots,$ to the BMA to generate some polynomials $\lambda_1(z)$ and $\lambda_2(z)$, and then wait until both the degrees of $\lambda_1(z)$ and $\lambda_2(z)$ are unchanged.

Let t be the number of auxiliary functions in $\mathbb{Z}_p(z)$ needed to determine $\#f_{\deg(f)}$ and $\#g_{\deg(g)}$, and to interpolate the polynomials $f_{\deg(f)}$ and $g_{\deg(g)}$. If the sequence of coefficients from the already computed t auxiliary functions are used to interpolate the lower total degree polynomials, it is possible that the wrong lower total degree polynomials are interpolated. This is because $\#f_i$ could be greater than $\#f_{\deg(f)}$ or $\#g_j$ could be greater than $\#g_{\deg(g)}$, where $i \neq \deg(f)$ and $j \neq \deg(g)$. Thus, more auxiliary coefficients are needed to interpolate the correct f_i or g_i.

We handle this problem by running the Berlekamp-Massey Algorithm on the sequence of corresponding coefficients from the auxiliary rational functions to generate a feedback polynomial $\lambda(z)$, and we check if $\deg(\lambda, z) < \frac{i}{2}$ [Kaltofen et al., 2000]. The condition $\deg(\lambda, z) < \frac{i}{2}$ ensures that $\lambda(z)$ is correct with high probability. If the con-

dition is not satisfied, then more auxiliary rational functions are needed. More detail about obtaining the correct number of terms is presented in Subsection 6.3.6.

4.4.1 Pre-computing the total degrees of f and g in $A = f/g$

Given a multivariate rational function $A = f/g$ in variables $y_1, y_2, \ldots, y_m$ represented by a black box **B**, we describe how to find the total degrees of the polynomials f and g. As we have demonstrated in the working example (Section 4.4), the total degrees of f and g are needed in order to densely interpolate the auxiliary rational functions with a Kronecker substitution K_r in variable z whose coefficients are needed to recover A. We discuss how to do this as follows.

Let $\alpha, \beta \in (\mathbb{Z}_p \setminus \{0\})^m$. Let p be a prime and let z be a new variable. Using enough random points for z from $\mathbb{Z}_p$, the total degrees of f and g that we seek can be obtained by interpolating the univariate rational function $h(z)$ where

$$h(z) := \frac{\bar{f}(z)}{\bar{g}(z)} = \frac{f(\beta_1 z + \alpha_1, \ldots, \beta_m z + \alpha_m)}{g(\beta_1 z + \alpha_1, \ldots, \beta_m z + \alpha_m)} \tag{4.3}$$

such that α, β are selected at random. We emphasize that $\alpha, \beta \in (\mathbb{Z}_p \setminus \{0\})^m$ are selected at random to ensure that $\deg(\bar{f}) = \deg(f)$ and $\deg(\bar{g}) = \deg(g)$ with high probability. Algorithm 10 implements this approach. To give reader an idea of how this approach works, we give the following example.

Example 4.9. Consider the rational function

$$A = f/g = \frac{y_2^2 + y_3}{y_1 + y_3}$$

such that $\deg(f) = 2$ and $\deg(g) = 1$. If A is represented by a blackbox then we need to interpolate $h(z) = \frac{\bar{f}(z)}{\bar{g}(z)}$ as described in (4.3) in order to determine the total degrees of f and g with high probability. Computing $h(z)$ directly yields

$$\begin{aligned} h(z) = \frac{\bar{f}(z)}{\bar{g}(z)} &= \frac{f(\beta_1 z + \alpha_1, \beta_2 z + \alpha_2, \beta_3 z + \alpha_3)}{g(\beta_1 z + \alpha_1, \beta_2 z + \alpha_2, \beta_3 z + \alpha_3)} \\ &= \frac{z^2\beta_2^2 + (2\alpha_2\beta_2 + \beta_3)\, z + \alpha_2^2 + \alpha_3}{(\beta_1 + \beta_3)\, z + \alpha_1 + \alpha_3}. \end{aligned}$$

Thus, $\deg(\bar{f}) = \deg(f) = 2$ and $\deg(\bar{g}) = \deg(g) = 1$. However, if $\alpha = (0, 0, 0)$, then

$$h(z) = \frac{z^2\beta_2^2 + \beta_3 z}{(\beta_1 + \beta_3)\, z} = \frac{z\beta_2^2 + \beta_3}{\beta_1 + \beta_3}$$

which gives the wrong degrees because $\deg(\bar{f}) = 1 \neq \deg(f)$ and $\deg(\bar{g}) = 0 \neq \deg(g)$.

Algorithm 10: TotalDegreeBound

Input: A prime p and the black box $\mathbf{B}\left(\mathbb{Z}_p^m, p\right) \rightarrow \mathbb{Z}_p$ for the rational function $\frac{f(y_1,y_2,\ldots,y_m)}{g(y_1,y_2,\ldots,y_m)}$ over $\mathbb{Q}$ which returns *"division by zero"* if $g(\gamma) = 0$ for some point $\gamma \in \mathbb{Z}_p^m$.

Output: $(\deg(\bar{f}), \deg(\bar{g}))$ where $\deg(\bar{f}) = \deg(f)$ and $\deg(\bar{g}) = \deg(g)$ with high probability.

1 Pick α and β in $(\mathbb{Z}_p \setminus \{0\})^m$ at random.
2 Let z be a new variable.
3 $\overline{m}(z) \leftarrow 1$.
4 **for** $d = 1, 2, \cdots$ **do**
5 Pick $\theta_d \in \mathbb{Z}_p$ at random such that $\theta_d \neq \theta_j$ for $1 \leq j < d$.
6 $a_d \leftarrow \mathbf{B}\left((\beta_1\theta_d + \alpha_1, \beta_2\theta_d + \alpha_2, \ldots, \beta_n\theta_d + \alpha_m), p\right) \in \mathbb{Z}_p$
7 **if** $a_d =$ *"division by zero"* **then** restart the algorithm (go to line 1) **end**.
8 **if** $d \geq 2$ **then**
9 Interpolate $u(z) \in \mathbb{Z}_p[z]$ of degree $< d$ using points $(\theta_i, a_i : 1 \leq i \leq d)$.
10 Compute $\overline{m}(z) = (z - \theta_d) \times \overline{m}(z) \in \mathbb{Z}_p[z]$. // $\overline{m}(z) = \prod_{i=1}^{d}(z - \theta_i) \in \mathbb{Z}_p[z]$.
11 $h(z) \leftarrow$ MQRFR $(\overline{m}, u, p)$. // $d = \deg(\overline{m}) > \deg(u) \geq 0$
12 **if** $h(z) \neq$ FAIL **then**
13 Let $h(z) = \frac{\bar{f}(z)}{\bar{g}(z)}$ such that $\gcd(\bar{f}(z), \bar{g}(z)) = 1$.
14 **return** $(\deg(\bar{f}), \deg(\bar{g}))$
15 **end**
16 **end**
17 $d \leftarrow d + 1$
18 **end**

Algorithm 10 discovers the total degrees $\deg(f)$ and $\deg(g)$ with high probability (see Section 4.6 for probability of failure).

4.4.2 Pre-computing the total degrees of the homogeneous polynomials f_i of f and g_i of g in $A = f/g$

Let $A = f/g$ be a sparse rational function in $y_1, y_2, \ldots, y_m$ over $\mathbb{Q}$ and suppose A is represented by a black box $\mathbf{B}$. Let

$$f = \sum_{i=0}^{\deg(f)} f_i(y_1, y_2, \ldots, y_m) \text{ and } g = \sum_{j=0}^{\deg(g)} g_j(y_1, y_2, \ldots, y_m)$$

where f_i and g_j are homogeneous polynomials such that $\deg(f_i) = i$ and $\deg(g_j) = j$. We discover the total degrees $\deg(f_i)$ and $\deg(g_i)$ with high probability as follows.

Let p be a sufficiently large prime and suppose we have obtained the total degrees $\deg(f)$ and $\deg(g)$ correctly. Then pick $\alpha \in (\mathbb{Z}_p \setminus \{0\})^m$ at random, and use enough random distinct

points for z selected from $\mathbb{Z}_p \setminus \{0\}$ to interpolate the univariate rational function

$$W(z) = \frac{\overline{N}}{\overline{D}} = \frac{\sum_{j=0}^{d_f} \bar{N}_i(z)}{\sum_{i=0}^{d_g} \bar{D}_i(z)} = \frac{f(\alpha_1 z, \ldots, \alpha_m z)}{g(\alpha_1 z, \ldots, \alpha_m z)} \in \mathbb{Z}_p(z),$$

where $d_f = \deg(\overline{N})$ and $d_g = \deg(\overline{D})$. Now, if $d_f = \deg(f)$ and $d_g = \deg(g)$, then $\deg(f_i) = \deg(\bar{N}_i)$ and $\deg(g_i) = \deg(\bar{D}_i)$ with high probability. But, if there is no constant term in f or g, which we do not know beforehand, then $\deg(f) \neq d_f$ or $\deg(g) \neq d_g$ because $e = \deg(\gcd(\overline{N}, \overline{D}))$ might be greater than zero.

Since we do not know what e is, it follows that, if $e = \deg(f) - d_f = \deg(g) - d_g$ with high probability, then $\deg(f_i) = \deg(\bar{N}_i) + e$ and $\deg(g_i) = \deg(\bar{D}_i) + e$ with high probability. We provide the following example to help the reader understand the approach.

Example 4.10. Let

$$A = \frac{f}{g} = \frac{y_1^3 + y_1 y_2}{y_2^2 + y_3}$$

where $f_3 = y_1^3, f_2 = y_1 y_2, g_2 = y_2^2$ and $g_1 = y_3$. Then

$$W(z) = \frac{f(\alpha_1 z, \alpha_2 z, \alpha_3 z)}{g(\alpha_1 z, \alpha_2 z, \alpha_3 z)} = \frac{\alpha_1^3 z^3 + \alpha_1 \alpha_2 z^2}{\alpha_2^2 z^2 + \alpha_3 z} = \frac{z(\alpha_1^3 z^2 + \alpha_1 \alpha_2 z)}{z(\alpha_2^2 z + \alpha_3)} = \frac{\alpha_1^3 z^2 + \alpha_1 \alpha_2 z}{\alpha_2^2 z + \alpha_3}.$$

So, $\overline{N} = \alpha_1^3 z^2 + \alpha_1 \alpha_2 z$ and $\overline{D} = \alpha_2^2 z + \alpha_3$. Thus, $d_f = 2$ and $d_g = 1$. We already know that $\deg(f) = 3$ and $\deg(g) = 2$ so we may compute $e = \deg(f) - d_f = \deg(g) - d_g = 1$ which implies that $\deg(f_3) = 2 + e = 3$, $\deg(f_2) = 1 + e = 2$, and $\deg(g_2) = 1 + e = 2$ and $\deg(g_1) = 0 + e = 1$. However, if

$$A = \frac{f}{g} = \frac{y_1^3 + y_1 y_2}{y_2^2 + y_3 + a}$$

then $f_3 = y_1^3, f_2 = y_1 y_2, g_2 = y_2^2, g_1$ and $g_0 = a \neq 0$, then we get

$$W(z) = \frac{f(\alpha_1 z, \alpha_2 z, \alpha_3 z)}{g(\alpha_1 z, \alpha_2 z, \alpha_3 z)} = \frac{\alpha_1^3 z^3 + \alpha_1 \alpha_2 z^2}{\alpha_2^2 z^2 + \alpha_3 z + a},$$

as long as the working prime $p \nmid a$. So,

$$\overline{N} = \alpha_1^3 z^2 + \alpha_1 \alpha_2 z, \text{ and } \overline{D} = \alpha_2^2 z + \alpha_3 + a$$

Clearly, $\deg(f) = d_f$ and $\deg(g) = 2 = d_g$, and we have that

- $\deg(f_3) = 3$, $\deg(f_2) = 2$,
- $\deg(g_2) = 2$, $\deg(g_1) = 1$, and $\deg(g_0) = 0$.

With high probability (see Section 4.6 for probability of failure), we obtain $\deg(f_i)$ and $\deg(g_i)$ using Algorithm 11.

Algorithm 11: PolyDegreeBound

Input: The total degrees $\deg(f)$ and $\deg(g)$, a prime p and the black box $\mathbf{B}\left(\mathbb{Z}_p^m, p\right) \rightarrow \mathbb{Z}_p$ for the rational function $\frac{f(y_1,y_2,\ldots,y_m)}{g(y_1,y_2,\ldots,y_m)}$ over $\mathbb{Q}$ which returns *"division by zero"* if $g(\gamma) = 0$ for some evaluation point $\gamma \in \mathbb{Z}_p^m$. Let

$$f = \sum_{i=0}^{\deg(f)} f_i(y_1, y_2, \ldots, y_m) \text{ and } g = \sum_{j=0}^{\deg(g)} g_j(y_1, y_2, \ldots, y_m)$$

where f_i and g_j are homogeneous such that $\deg(f_i) = i$ and $\deg(g_j) = j$.

Output: $[\deg(f_i) : 0 \leq i \leq \deg(f)]$ and $[\deg(g_i) : 0 \leq i \leq \deg(g)]$ with high probability.

```
1  Pick (α_1, α_2, ···, α_m) ∈ (Z_p \ {0})^m at random.
2  Let z be a new variable.
3  m̄(z) ← 1.
4  for d = 1, 2, ··· do
5      Pick θ_d ∈ Z_p at random such that θ_d ≠ θ_i for 1 ≤ i < d.
6      a_d ← B((α_1 θ_i, α_2 θ_i, ..., α_m θ_i), p) ∈ Z_p
7      if a_d = "division by zero" then restart the algorithm (go to line 1) end.
8      if d ≥ 2 then
9          Interpolate the unique polynomial u(z) ∈ Z_p[z] of degree < d using points
             (θ_i, a_i : 1 ≤ i ≤ d).
10         Compute m̄(z) = (z − θ_d) × m̄(z) ∈ Z_p[z]. // m̄(z) = ∏_{i=1}^{d} (z − θ_i) ∈ Z_p[z].
11         h(z) ← MQRFR (m̄, u, p). // deg(m̄) > deg(u) ≥ 0
12         if h(z) ≠ FAIL then
13             Let h(z) = f̄(z)/ḡ(z) such that f̄ = Σ_{i=0}^{deg(f̄)} f̄_i and ḡ = Σ_{i=0}^{deg(ḡ)} ḡ_i
14             e_1 ← deg(f) − deg(f̄)
15             e_2 ← deg(g) − deg(ḡ)
16             if e_1 = e_2 then
17                 return [deg(f̄_i) + e_1 : 0 ≤ i ≤ deg(f)], [deg(ḡ_i) + e_1 : 0 ≤ i ≤ deg(g)]
18             else
19                 return FAIL
20             end
21         end
22     end
23     d ← d + 1
24 end
```

The following pseudocode (Section 4.5) highlights the steps involved to interpolate a sparse multivariate rational function $A = f/g$ using our new sparse rational function interpolation method. More details will be presented later when we apply our new sparse multivariate rational function interpolation algorithm to interpolate many rational function

coefficients over $\mathbb{Q}$ in our proposed Dixon resultant algorithm, and in our proposed black box algorithm for solving parametric linear systems.

4.5 New sparse multivariate rational function interpolation algorithm

Input: The black box $\mathbf{B} : (\mathbb{Z}_p^m, p) \to \mathbb{Z}_p$ for the rational function $A = f/g$ in $y_1, \ldots, y_m$ over $\mathbb{Q}$ which returns *"division by zero"* if $g(\gamma) = 0$ for some evaluation point $\gamma \in \mathbb{Z}_p^m$.
Remark: The input prime p for the black box will be determined while the algorithm is executing. In particular, it will be determined when the partial degrees of f and g are discovered for all variables involved with high probability.
Output: $A = f/g \bmod p$ with high probability.

(a) Let $f = \sum_{i=0}^{\deg(f)} f_i(y_1, \ldots, y_m)$ and $g = \sum_{i=0}^{\deg(g)} g_i(y_1, \ldots, y_m)$ where f_i and g_i are homogeneous polynomials such that $\deg(f_i) = \deg(g_i) = i$. Let $t_{\max} = \max(\#f_i, \#g_i)$.

(b) Compute the total degrees $\deg(f)$ and $\deg(g)$ using Algorithm 10, the maximum partial degrees $d_i = \max(\deg(f, y_i), \deg(g, y_i))$ for $1 \le i \le m$ using Algorithm 9, and the total degrees $\deg(f_i)$ and $\deg(g_i)$ using Algorithm 11.

(c) Pick a smooth prime $p = 2^k s + 1 > \prod_{j=1}(d_i + 1)$, a random shift $\hat{s} \in [0, p-2]$, and any generator α for $\mathbb{Z}_p^*$. Note that p is our working prime that will be used in the black box $\mathbf{B}$ for A.

(d) Let $K_r : \mathbb{Z}_p(y_1, \ldots, y_m) \to \mathbb{Z}_p(y)$ be the Kronecker substitution with $r_i > d_i$.

(e) Let $\beta = (0, 0, \ldots, 0) \in \mathbb{Z}^m$ be a basis shift.

While $\mathbf{B}(\beta, p) =$ *"division by zero"* or $\mathbf{B}(\beta, p) = 0$ **do**

Pick a new random basis shift $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$.

end do

(f) Pick $\deg(f) + \deg(g) + 2$ random points for z and probe the black box to interpolate the auxiliary rational functions

$$F(\alpha^{\hat{s}+i}, z, \beta) = \frac{f^\beta(\alpha^{\hat{s}+i}, z)}{g^\beta(\alpha^{\hat{s}+i}, z)} = \frac{f(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)}$$

where $F(\alpha^{\hat{s}+i}, z, \beta) \in \mathbb{Z}_p(z)$ such that $g^\beta(\alpha^{\hat{s}+i}, z)$ is of the form $1 + \sum_{k=1}^{\deg(g)} a_k z^k$ for $i = 0, 1, 2, \ldots, 2t_{\max} - 1$.

(g) Interpolate the polynomials $f_{\deg(f)}$ and $g_{\deg(g)}$ first using the sequence of leading coefficients $\mathrm{LC}(f^\beta(\alpha^{\hat{s}+i}, z))$ and $\mathrm{LC}(g^\beta(\alpha^{\hat{s}+i}, z))$ respectively.

(h) Remove the effect of the basis of shift β by adjusting the coefficients of z^k in $f^{\beta}(\alpha^{s+i}, z)$ and z^j in $g^{\beta}(\alpha^{s+i}, z)$, before interpolating the homogeneous polynomials f_k and g_j for $k = \deg(f) - 1, \deg(f) - 2, \ldots, 0$ and $j = \deg(g) - 1, \deg(g) - 2, \ldots, 0$ respectively.

(i) Construct polynomials $f = \sum_{i=0}^{\deg(f)} f_i(y_1, \ldots, y_m)$ and $g = \sum_{i=0}^{\deg(g)} g_i(y_1, \ldots, y_m)$.

(j) Output $A = f/g \bmod p$.

4.6 The Failure Probability Analysis of Algorithm 10

Let $\deg(f)$ and $\deg(g)$ denote the actual total degrees of f and g in the multivariate rational function $A = f/g$ represented by a black box $\mathbf{B}$, and let $\deg(\bar{f})$ and $\deg(\bar{g})$ denote the output degrees of Algorithm 10. We first remind the reader that Algorithm 10 is designed to output $\deg(\bar{f})$ and $\deg(\bar{g})$ such that $\deg(\bar{f}) = \deg(f)$ and $\deg(\bar{g}) = \deg(g)$.

Since Algorithm 10 calls the MQRFR algorithm (Algorithm 8) at every step d in Line 11, a univariate rational function $h(z) = \bar{f}(z)/\bar{g}(z)$ satisfying the degree restriction $d > \deg(\bar{f}) + \deg(\bar{g}) + 1$ is produced only when a maximal quotient of degree 2 or more is found. For example, if $d = 2$, we have that $\deg(\overline{m}) = 2$ and $0 \leq \deg(u) \leq 1$. So, if $\deg(u) = 0$, a maximal quotient of degree 2 can be found, and a univariate rational function satisfying the degree requirement $0 \leq \deg(\bar{f}) + \deg(\bar{g}) \leq 1$ will be produced.

Thus, Algorithm 10 will always attempt to find the total degrees $\deg(f)$ and $\deg(g)$ with high probability as long as the number of evaluation points used is 2 or more ($d \geq 2$). However, Algorithm 10 may fail by returning the wrong numerator and denominator total degrees for f and g in $A = f/g$ from the wrong univariate rational function $h(z)$, when a quotient of degree 2 or more occurs for $d < \deg(f) + \deg(g) + 2$ during the call to MQRFR algorithm. Therefore, we need to find a bound for the failure probability of Algorithm 10. First, we give an example to illustrate the failure.

Example 4.11. Consider the bivariate rational function

$$A = f/g = \frac{y_2^2 + 2y_1}{y_1 + 2y_2}.$$

Suppose we want to determine the total degrees $\deg(f) = 2$ and $\deg(g) = 1$ using Algorithm 10. Algorithm 10 will attempt to find the total degrees of f and g using $d = 2, 3, 4, 5, \ldots,$ evaluation points. If everything goes right, Algorithm 10 terminates at $d = 5$, and outputs the correct total degrees of f and g. But at $d = 2$ or 3 or 4, the wrong total degree may be returned. We consider the following case when $d = 4$ in which this error can happen using the notation of Algorithm 10.

Let $p = 3137$ and let the evaluation points chosen for z be

$$\theta_1 = 2909, \theta_2 = 2799, \theta_3 = 1325, \theta_4 = 2016.$$

Suppose $\beta = (27, 1445) \in \mathbb{Z}_p^2$ and let $\alpha = (3, 6) \in \mathbb{Z}_p^2$ as defined in Line 1 of Algorithm 10. We have that the images a_i obtained in Line 6 will be

$$a_1 = 1551, a_2 = 1550, a_3 = 2753, a_4 = 1460.$$

The product polynomial in Line 10 will be

$$\overline{m}(z) = \prod_{i=1}^{4}(z - \theta_i) = z^4 + 362z^3 + 857z^2 + 1679z + 607 \in \mathbb{Z}_p[z]$$

and in Line 9, the interpolating polynomial

$$u(z) = 1165z^3 + 382z^2 + 2710z + 2246$$

which is obtained using the points $\{(\theta_i, a_i) : 1 \leq i \leq 4\}$. We present the intermediate computations that occur when we input both $\overline{m}$ and u as inputs to the MQRFR algorithm in Line 11 of Algorithm 10.

Table 4.1: MQRFR (Algorithm 8) intermediate computations for input polynomials $\overline{m}$ and u

i	q_i	r_i	t_i
0	–	$z^4 + 362z^3 + 857z^2 + 1679z + 607$	0
1	$517z + 749$	$1165z^3 + 382z^2 + 2710z + 2246$	1
2	$2750z + 755$	$1375z^2 + 1036z + 2922$	$2388 + 2620z$
3	$1512z^2 + 2483z + 1680$	1447	$689z^2 + 88z + 836$
4	–	0	$2853z^4 + 713z^3 + 1298z^2 + 3125z + 147$

Notice that the maximal quotient occurs in row 3 as highlighted in Table 4.1. So, Algorithm 10 outputs the wrong degrees 0 and 2 from

$$r_3/t_3 = \frac{1447}{689z^2 + 88z + 836} \equiv \frac{767}{z^2 + 1794z + 1358}$$

for f and g in f/g as highlighted in Table 4.1 when $d = 4$.

Remark 4.12. Suppose

$$A(z, y_1, \ldots, y_m) = \frac{f(y_1 z + \alpha_1, \ldots, y_m z + \alpha_m)}{g(y_1 z + \alpha_1, \ldots, y_m z + \alpha_m)} = \frac{\sum_{k=0}^{\deg(f)} M_k(y_1, \ldots, y_m) z^k}{\sum_{k=0}^{\deg(g)} N_k(y_1, \ldots, y_m) z^k}.$$

Let $d = \deg(f) + \deg(g) + 1$ and let $\beta \in \mathbb{Z}_p^m$. Let

$$A(z, \beta) = \frac{\hat{f}(z)}{\hat{g}(z)} \in \mathbb{Z}_p(z).$$

We know that there exist polynomials u and $\overline{m}$ satisfying $\deg(\overline{m}) > \deg(u)$ such that

$$\frac{\hat{f}(z)}{\hat{g}(z)} \equiv u(z) \bmod \overline{m}(z),$$

where $u(\theta_j) = A(\theta_j, \beta)$ for $1 \leq j \leq d$ and $\overline{m}(z) = \prod_{j=1}^{d+1}(z - \theta_j)$. Now let

$$\hat{W}_j = A(\theta_j, y_1, y_2, \ldots, y_m) = \frac{f(y_1\theta_j + \alpha_1, \ldots, y_m\theta_j + \alpha_m)}{g(y_1\theta_j + \alpha_1, \ldots, y_m\theta_j + \alpha_m)} \text{ for } 1 \leq j \leq d.$$

Using polynomial interpolation, we know there exists a unique function $H \in \mathbb{Z}_p(y_1, \ldots, y_m)[z]$ of degree at most d in z such that $H(\theta_j, y_1, y_2, \ldots, y_m) = \hat{W}_j$. Let

$$H(z, y_1, \ldots, y_m) = \sum_{k=0}^{d-1} \frac{N_k(y_1, \ldots, y_m)}{D_k(y_1, \ldots, y_m)} z^k \in \mathbb{Z}_p(y_1, \ldots, y_m)[z]$$

and let $L = \text{LCM}\{D_k \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m] : 0 \leq k \leq d-1\}$. It is not hard to show that $\deg(N_k) \leq \deg(f) + (d-1)\deg(g)$ and $\deg(D_k) \leq d\deg(g)$. Clearing fractions of H, we have that $\hat{H} = HL$. Note that the total degree of $\hat{H}$ in $y_1, y_2, \cdots y_m$ is at most

$$\deg(f) + 2d^2 \deg(g). \tag{4.4}$$

Thus $\hat{H}(z, y_1, \ldots, y_m) \equiv L(y_1, \ldots, y_m)A(z, y_1, \ldots, y_m) \bmod (z - \theta_j)$ which implies that $L(y_1, y_2, \cdots y_m)A(z, y_1, y_2, \cdots y_m) \equiv \hat{H}(z, y_1, y_2, \cdots y_m) \bmod \overline{m}(z)$. Hence,

$$L(\beta)\frac{\hat{f}(z)}{\hat{g}(z)} \equiv \hat{H}(z, \beta) \bmod \overline{m} \implies \hat{H}(z, \beta) = \hat{k}u(z) \text{ for some } \hat{k} \in \mathbb{Z}_p.$$

Thus, inputting $\overline{m}(z)$ and $u(z)$ to the MQRFR algorithm is equivalent to calling the MQRFR algorithm with inputs $m(z)$ and $\hat{H} \in \mathbb{Z}_p[y_1, \ldots, y_m][z]$, and then evaluating the intermediate remainders which are rational functions at $(y_1, y_2, \cdots y_m) = (\beta_1, \beta_2, \ldots, \beta_m)$.

Example 4.13 (Continuation of Example 4.11)**.** We found out that inputting

$$\begin{aligned}
\hat{H} = &\left(4y_1^3 + 3121y_1^2y_2 + 3100y_1\,y_2^2 + 22y_2^3\right)z^3 \\
&+ \left(1448y_1^3 + 482y_1^2y_2 + 2291y_1\,y_2^2 + 1690y_2^3 + 3077y_1^2 + 360y_1y_2 + 2972y_2^2\right)z^2 \\
&+ (1889y_1^3y_2^2 + 1923y_1^2y_2^3 + 709y_1\,y_2^4 + 2564y_2^5 + 2121y_1^2y_2^2 + 2210y_1\,y_2^3 + 2210y_2^4 \\
&+ 291y_1^3 + 1973y_1^2y_2 + 1263y_1\,y_2^2 + 99y_2^3 + 239y_1^2 + 1703y_1y_2 + 2450y_2^2 + 900y_1 + 2211y_2)z \\
&+ 641y_1^4 + 1418y_1^3y_2 + 2433y_1^2y_2^2 + 680y_1\,y_2^3 + 2127y_2^4 + 1547y_1^3 + 3008y_1^2y_2 + 2879y_1\,y_2^2 \\
&+ 2965y_2^3 + 1976y_1^2 + 1630y_1y_2 + 1630y_2^2 + 1569y_1 + y_2 + 2942.
\end{aligned}$$

and the product polynomial

$$\overline{m}(z) = \prod_{i=1}^{4}(z-\theta_i) = z^4 + 362z^3 + 857z^2 + 1679z + 607 \in \mathbb{Z}_p[z]$$

to the MQRFR algorithm produces a remainder $r_3^* \in \mathbb{Z}_p(y_1, y_2)[z]$ whose degree in z is one and $\text{LC}(r_3^*) = f_1 f_2$ where

$$\begin{aligned} f_1 &= 3059y_1\,y_2^6 + 2981y_2^7 + 1505y_2^6 + 2567y_1\,y_2^4 + 1423y_2^5 + 2956y_1^2y_2^2 + 362y_1\,y_2^3 + 1584y_2^4 \\ &\quad + y_1^3 + 4y_1^2y_2 + 740y_1\,y_2^2 + 1522y_2^3, \\ f_2 &= \frac{16(y_1+2y_2)}{y_2^4}. \end{aligned}$$

However, computing $\text{LC}(r_3^*)(y_1 = \beta_1, y_2 = \beta_2) = 0$ implying that the remainder r_3^* is a constant. The evaluation points β caused the $\deg(r_3, z)$ to drop by 1. This is why r_3 in Table 4.1 is a constant and not a degree 1 polynomial in z. We are now ready to give a failure probability bound for Algorithm 10.

Proposition 4.14. Let $A = f/g \in \mathbb{Z}(y_1, y_2, \ldots, y_m)$ where $f = \sum_{i=0}^{\deg(f)} f_i$ and $g = \sum_{i=0}^{\deg(g)} g_i$ such that $i = \deg(f_i) = \deg(g_i)$ and $\deg(f) \geq \deg(g)$. Let p be a prime such that p does not divide $f_{\deg(f)}$ and $g_{\deg(g)}$. If no evaluation point causes the black box for $A = f/g$ to output a division by 0, and the random evaluation point $\beta \neq 0$ selected in Line 1 of Algorithm 10 does not cause both $f_{\deg(f)}$ and $g_{\deg(g)}$ to vanish then

$$\Pr[\text{Algorithm 10 returns one or more wrong degrees}] \leq \frac{529\deg(f)^7}{p}.$$

Proof. Let $D = \deg(f) + \deg(g) + 1$ and let $2 \leq d \leq D+1$. Let $\overline{m}(z)$ and $u(z)$ be as constructed in Lines 10-9 of Algorithm 10 where $\deg(\overline{m}(z)) = d$ and $\deg(u(z)) = d-1$. For simplicity, suppose $r_0^d = \overline{m}(z)$ and let $r_1^d = u(z)$. Recall that the number of division steps $l \leq \deg(u) + 1 = d$ [von zur Gathen and Gerhard, 2013].

Let $\{r_i^d, 2 \leq i \leq d\}$ be the respective remainder sequence generated at each step d in Line 11 of Algorithm 10 with the assumption that $r_2^2 \neq 0$ when the MQRFR algorithm is called with inputs $r_0^d = m(z)$ and $r_1^d = u(z)$. Notice that we exclude the final remainder r_d^d in the MQRFR algorithm from $\{r_i^d, 2 \leq i \leq d\}$ because $r_d^d = 0$ for $2 < d \leq D+1$. Lines $1-3$ of the MQRFR algorithm takes care of this special case when a zero rational function can be returned. Thus, we cannot output a zero rational function at step d of Algorithm 10 whenever $d \geq 2$ and $\deg u \geq 0$. Let $\dfrac{N_{i,d}(y_1, y_2, \ldots, y_m)}{M_{i,d}(y_1, y_2, \ldots, y_m)}$ be the leading coefficients of the remainders r_i^d and let

$$h(z) = \frac{r_i^d(z)}{t_i^d(z)} = \frac{\bar{f}(z)}{\bar{g}(z)}$$

with $\deg(\bar{f}) = \deg(r_i^d(z))$ and $\deg(\bar{g}) = \deg(t_i^d(z))$, where the polynomials r_i^d, t_i^d are generated from the MQRFR algorithm for each step d in Algorithm 10 for $2 \leq i \leq d$. Note that a quotient of degree 2 or more is generated when the degree of one of the remainders in the remainder sequence $\{r_i^d, 2 \leq i \leq d\}$ has degree 1 or more lower than its actual degree, which happens when their leading coefficient vanishes. Clearly, $\frac{N_{i,d}(y_1,y_2,\ldots,y_m)}{M_{i,d}(y_1,y_2,\ldots,y_m)}$ vanishes when $N_{i,d}(\beta) = 0$ and $M_{i,d}(\beta) \neq 0$.

If we define $\Delta_d = \prod_{i=2}^{d} N_{i,d}(x_1, x_2, \ldots, x_n)$, then $\Delta_d(\beta) = 0$ whenever $N_{i,d}(\beta) = 0$ for some i. Let $\Delta = \prod_{d=1}^{D} \Delta_{d+1}$. Using Theorem B.2 and (4.4), we have that

$$\begin{aligned}\deg(N_{i,d}) &\leq i(\deg(\overline{m}, z) + \deg(u, z)) \max\left(\deg(\overline{m}, \{y_1, y_2, \cdots y_m\}), \deg(u, \{y_1, y_2, \cdots y_m\})\right) \\ &\leq i(2d-1)\left(\deg(f) + 2d^2 \deg(g)\right).\end{aligned}$$

Therefore, using the Schwartz-Zippel Lemma, it follows that

$$\begin{aligned}&\Pr[\text{Algorithm 10 returns one or more wrong degrees}] = \Pr[\Delta(\beta) = 0] \leq \frac{\deg(\Delta)}{p} \\ &\leq \frac{\sum_{d=2}^{D+1} \sum_{i=2}^{d} \deg(N_{i,d})}{p} \\ &\leq \frac{\sum_{d=2}^{D+1} \sum_{i=2}^{d} i(2d-1)\left(\deg(f) + 2d^2 \deg(g)\right)}{p} \\ &\leq \frac{529 \deg(f)^7}{p}.\end{aligned}$$

□

Chapter 5

The Dixon Resultant Algorithm

5.1 Summary of Contributions

All the materials presented in this chapter are new. The main contribution of this chapter is the design and implementation of a new Dixon resultant algorithm for solving parametric polynomial systems over $\mathbb{Q}$. A preliminary version of our Dixon resultant algorithm with benchmarks has been published in the Proceedings of CASC '22 [Jinadu and Monagan, 2022a]. The algorithm was also presented at Maple conference '22 and at the ISSAC '22 poster session (a 4-page extended abstract was published [Jinadu and Monagan, 2022b]). A journal paper presenting the improved version of the algorithm with new benchmark polynomial systems, including its failure probability analysis and complexity analysis is currently being prepared.

The improved version of our new algorithm with the benchmarks (real parametric polynomial systems) will be presented in this chapter, and we compare our new Dixon resultant algorithm with a Maple implementation of the Gentleman & Johnson minor expansion algorithm [Monagan, 2023b], a Maple implementation of the Dixon-EDF algorithm [Monagan, 2023b] and a hybrid Maple+C implementation of Zippel's sparse interpolation algorithm for interpolating the Dixon resultant R. The failure probability analysis and the complexity analysis of our new Dixon resultant algorithm will be presented in Chapter 6.

5.2 Introduction

To avoid ambiguity, any sub-matrix M of a Dixon matrix D such that $\text{rank}(M) = \text{rank}(D)$ will always be referred to as a Dixon matrix.

Let M be a Dixon matrix with polynomial entries in $x_1, y_1, y_2, \cdots y_m$ and let R be the Dixon resultant $R = \det(M)$. As we have discussed in the introductory chapter of this thesis, our goal is to get rid of the unwanted repeated factors and the polynomial content of R, by interpolating the monic square-free factors R_j of the Dixon resultant R and not R directly. We remind the reader that the monic square-free factorization of the Dixon

resultant R is a unique factorization of the form $\hat{r}\prod_{j=1}^{l} R_j^j$ where

$$R_j = x_1^{d_{T_j}} + \sum_{k=0}^{T_j-1} \frac{f_{jk}(y_1, y_2, \ldots, y_m)}{g_{jk}(y_1, y_2, \ldots, y_m)} x_1^{d_{j_k}} \in \mathbb{Q}(y_1, y_2, \ldots, y_m)[x_1] \tag{5.1}$$

for non-zero $f_{jk}, g_{jk} \in \mathbb{Q}[y_1, y_2, \ldots, y_m]$ such that

(1) $\hat{r} = C/L$ for some $L \in \mathbb{Q}[Y]$,

(2) each R_j is monic and square-free in $\mathbb{Q}(Y)[x_1]$,

(3) $\gcd(R_i, R_j) = 1$ for $i \neq j$.

(4) $\gcd(f_{jk}, g_{jk}) = 1$ for all $0 \leq k \leq T_j - 1$ and

(5) $\mathrm{LC}(g_{jk}) = 1$.

Let M be a Dixon matrix of polynomials in $\mathbb{Z}[x_1, y_1, \ldots, y_m]$. For our Dixon resultant algorithm, our black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ is a program that takes an evaluation point $\alpha \in \mathbb{Z}_p^{m+1}$ and a prime p as inputs and outputs $\det(M(\alpha)) \bmod p$.

We have designed our new Dixon resultant algorithm to probe the black box representing $R = \det(M)$ in order to interpolate the monic square-free factors R_j of R from monic univariate polynomial images of R in x_1, using our new sparse multivariate rational function interpolation from Chapter 4 to recover the coefficients of R_j in $\mathbb{Q}(Y)$ modulo primes, and then use Chinese remaindering and rational number reconstruction to recover their rational coefficients.

For the purpose of a simpler description, in this thesis we assume that there is one monic square-free factor to be interpolated. That is, our Dixon resultant algorithm is presented to interpolate only one square-free factor R_j. However, our implementation of our Dixon resultant algorithm handles more than one monic square-free factor. Let

$$S = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)} x_1^{d_k} \tag{5.2}$$

be the one monic square-free factor to be interpolated where the polynomials f_k and g_k can be expressed in the form

$$f_k = \sum_{i=0}^{\deg(f_k)} f_{i,k}(y_1, \ldots, y_m) \text{ and } g_k = \sum_{j=0}^{\deg(g_k)} g_{j,k}(y_1, \ldots, y_m) \tag{5.3}$$

such that $f_{i,k}$ and $g_{j,k}$ are homogeneous polynomials and $\deg(f_{i,k}) = i$ and $\deg(g_{j,k}) = j$.

Based on our new sparse rational function interpolation method, we remind the reader that the interpolation of the multivariate rational function coefficients $\dfrac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)}$ of the

monic square-free factor S is reduced to the interpolation of univariate rational functions

$$K_r(f_k/g_k) = \frac{f_k(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}{g_k(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})} \in \mathbb{Q}(y)$$

in our Dixon resultant algorithm, where $r = (r_1, r_2, \ldots, r_{m-1}) \in \mathbb{Z}^{m-1}$ with

$$r_i > \max_{k=0}^{T-1}(\max(\deg(f_k, y_i), \deg(g_k, y_i)),$$

and K_r is the Kronecker substitution and T is given in 5.2. The corresponding auxiliary rational function with a Kronecker substitution K_r is

$$F_k(y, z, \beta) := \frac{f_k^{\beta}(y, z)}{g_k^{\beta}(y, z)} = \frac{f_k(zy + \beta_1, zy^{(r_1)} + \beta_2, \ldots, zy^{(r_1 r_2 \cdots r_{m-1})} + \beta_m)}{g_k(zy + \beta_1, zy^{(r_1)^i} + \beta_2, \ldots, zy^{(r_1 r_2 \cdots r_{m-1})} + \beta_m)}. \tag{5.4}$$

The implication of the black box representation of the Dixon resultant $R = \det(M)$ is that vital information such as the number of terms and the variable degrees are unknown. We need to know

(i) $\deg(R, x_1)$ needed for univariate polynomial interpolation,

(ii) the degrees $[d_0, \ldots, d_{T-1}]$ as defined in (5.2), to ensure that there are no missing terms in the interpolated univariate monic polynomial images of R,

(iii) the total degrees $\deg(f_k)$ and $\deg(g_k)$ for $0 \leq k \leq T-1$, which are needed to densely interpolate the univariate auxiliary rational functions $F_k(\alpha^{\hat{s}+i}, z, \beta)$ for $= 0, 1, \ldots,$

(iv) the maximum partial degrees $\max_{k=0}^{T-1}(\max(\deg(f_k, y_i), \deg(g_k, y_i))$ of S with respect to each variable y_i which are needed to apply a Kronecker substitution, and

(v) the total degrees of the polynomials $f_{i,k}$ and $g_{i,k}$.

We can discover $(i) - (v)$ before the start of our Dixon resultant algorithm with high probability, but we cannot discover $\#f_{i,k}$ and $\#g_{i,k}$ in advance. They are discovered while the algorithm is executing. Thus, the first step in our Dixon resultant algorithm is to discover these degrees.

5.2.1 Degree bounds

The first degrees that must be discovered are $\deg(R, x_1)$ and the list of degrees $[d_0, \ldots, d_T]$ as defined in (5.2). Let p be a large prime and let $\alpha \in \mathbb{Z}_p^m$ be a random evaluation point. We have designed Algorithm 12 to discover $\deg(R, x_1)$ and degrees $[d_0, \ldots, d_T]$ by interpolating a univariate polynomial $g(x_1) \in \mathbb{Z}_p[x_1]$ from $g(\delta_i) := R(\delta_i, \alpha)$ via black box calls to **BB** for $i = 1, 2, 3, \ldots$. The algorithm stops when $\deg(g) = \deg(R, x_1)$ and the support of

$g(x_1) \in \mathbb{Z}_p[x_1]$ yields $[d_0, \ldots, d_T]$. Algorithm 12 then uses one extra random point from $\mathbb{Z}_p$ to check if the degrees are correct with high probability.

Algorithm 12: ABound

Input: A prime p and a black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for the Dixon resultant $R \in \mathbb{Z}_p[x_1, y_1, y_2, \ldots, y_m]$.

Output: The list of degrees $[d_0, \ldots, d_T]$ as defined in (5.2) and $\deg(R, x_1)$.

1 Pick $\alpha \in \mathbb{Z}_p \setminus \{0\}^m$ at random.
2 $\hat{D} \leftarrow -1$
3 **for** $t = 1, 2, 3, \cdots$ **do**
4 Pick $\delta_t \in \mathbb{Z}_p$ at random such that $\delta_t \notin \{\delta_1, \delta_2, \ldots, \delta_{t-1}\}$.
5 $H_t \leftarrow \mathbf{BB}\,((\delta_t, \alpha), p)$//probes to the black box
6 Interpolate the polynomial $g \in \mathbb{Z}_p[x_1]$ of degree $< t$ with points (δ_i, H_i) for $1 \le i \le t$.
7 **if** $\hat{D} = \deg(g, x_1)$ **then**
8 // Break out of the main for loop (terminate) when degree does not change.
9 **Probabilistic check** : Pick $\theta \in \mathbb{Z}_p$ at random. // An extra point.
10 **if** $\mathbf{BB}\,((\theta, \alpha), p) = g(\theta)$ **then**
11 $g \leftarrow g/\mathrm{lc}(g, x_1)$ // Make g monic.
12 $\hat{f} \leftarrow g/\gcd(g, \frac{dg}{dx_1})$ // $\hat{f}$ is the monic square-free part of g.
13 Let $\hat{f} = x_1^{d_T} + \sum_{k=0}^{T-1} a_k x_1^{d_k}$ with $a_k \neq 0$.
14 **return** $[d_0, \ldots, d_T]$ and $\hat{D}$.
15 **else**
16 Go to Line 1 // restart the algorithm
17 **end**
18 **else**
19 $\hat{D} \leftarrow \deg(g, x_1)$
20 **end**
21 **end**

Note that the returned output by Algorithm 12 might be incorrect. We give the following three examples to illustrate how this can happen.

Example 5.1. Let the Dixon resultant

$$R = (y_1 - a_1)(y_2 - a_2)x_1^4 + 5x_1^2 + 7.$$

If the random point α selected in Line 1 is $\alpha = (a_1, a_2)$, then

$$\mathrm{LC}(R)(\alpha) = 0 \text{ and } g(x_1) = 5x_1^2 + 7 \Longrightarrow \deg(g) = 2 < \deg(R, x_1) = 4.$$

Example 5.2. Let the Dixon resultant

$$R = x_1^2 + \sum_{j=1}^{2} y_j^4 + \prod_{j=1}^{4} (x_1 - \delta_j).$$

Since $\deg(R, x_1) = 4$, we need 5 points to interpolate a univariate polynomial $g \in \mathbb{Z}_p[x_1]$ of degree 4, plus one extra point to confirm that $\deg(g) = \deg(R, x_1)$ with high probability.

However, at $t = 3$, suppose the first three random points picked by Algorithm 12 in Line 4 are $\delta_1, \delta_2, \delta_3$. Then the interpolated polynomial produced in Line 6 is

$$g(x_1) = x_1^2 + \sum_{j=1}^{2} \alpha_j^4.$$

Algorithm 12 is going to pick another point to check if it has the correct answer. At $t = 4$, suppose the next random point picked is δ_4, so the four successive random points to be used are $\delta_1, \delta_2, \delta_3, \delta_4$. Again, Algorithm 12 interpolates

$$g(x_1) = x_1^2 + \sum_{j=1}^{2} \alpha_j^4$$

and it terminates since $\deg(g)$ has stabilized. But $\deg(g) = 2 < \deg(R, x_1) = 4$ which implies that Algorithm 12 terminated too early. This early termination problem led us to add a probabilistic check in Lines 9-10 to ensure that the returned degrees are correct with high probability. However, it is also possible that our probabilistic check fails by confirming that we have obtained the correct degree even though $\deg(g) \neq \deg(R, x_1)$. To see this, let $\theta \in \mathbb{Z}_p$ be selected at random as in Line 9. Observe that $g(\theta) = \theta^2 + \sum_{j=1}^{2} \alpha_j^4$ and

$$\mathbf{BB}\left((\theta, \alpha), p\right) = R(\theta, \alpha) = \theta^2 + \sum_{j=1}^{2} \alpha_j^4 + \prod_{j=1}^{4} (\theta - \delta_j).$$

Thus, if $\theta = \delta_j$ for any j such that $1 \leq j \leq 4$, then $g(\theta) = \mathbf{BB}\left((\theta, \alpha), p\right)$, which implies that the check will confirm that we have the correct degree even though it is wrong.

Example 5.3. Let the Dixon resultant $R = (x_1 + y_1 - a)^2$. If the random point α selected in Line 1 of Algorithm 12 is $\alpha = a$ then the interpolated polynomial $g(x_1) = x_1^2$. Notice that this yields the correct degree of R in x_1 but the monic square-free part $\hat{f} \in \mathbb{Z}_p[x_1]$ of $g(x_1)$ in Line 12 is x_1 which implies that $[d_0, d_1] = [1]$. However, the correct list of degrees $[d_0, d_1] = [0, 1]$. Therefore, Algorithm 12 will return the incorrect list of degrees.

Theorem 5.4. Let $R \in \mathbb{Z}[x_1, y_1, \ldots, y_m]$ be the Dixon resultant and let $S \in \mathbb{Z}(y_1, \ldots, y_m)[x_1]$ be its only monic square-free factor where $\|S\|_\infty$ and $\|R\|_\infty$ denote the largest integer coefficient of S and R respectively. If a prime p is chosen at random from a list of pre-computed N primes $P = \{p_1, p_2, \ldots, p_N\}$ such that $p_{\min} = \min(P)$ then the probability that Algorithm 12 returns the wrong degrees is at most

$$\frac{3 \deg(R) \deg(R, x_1)}{p} + \frac{\deg(R, x_1) \left(\log_{p_{\min}} \|S\|_\infty + \log_{p_{\min}} \|R\|_\infty\right)}{N}.$$

Proof. Let $g = R(x_1, \alpha)$ be the interpolated polynomial as defined in Line 6 where $\alpha \in \mathbb{Z}_p^m$ is the random evaluation point picked in Line 1 of Algorithm 12. Let $\hat{f} = x_1^{d_T} + \sum_{k=0}^{T-1} a_k x_1^{d_k}$ be the monic square-free part of g as defined in Line 12 of Algorithm 12. Clearly, Algorithm 12 returns the wrong degrees if

1. $\deg(g) < \deg(R, x_1)$ or
2. the coefficients $a_k = 0$ for any k (the monic square-free part $\hat{f}$ loses its support) or
3. $g(x_1) \neq R(x_1, \alpha)$ but $\mathbf{BB}\left((\theta, \alpha), p\right) - g(\theta) = 0$.

We obtain the failure probability bounds for each of the cases as follows.

Case 1: Clearly, $\deg(g) < \deg(R, x_1)$ if $p|\mathrm{LC}(R)$ or $\mathrm{LC}(R)(\alpha) = 0$. Therefore,

$$\begin{aligned}\Pr[\deg(g) < \deg(R, x_1)] &= \Pr[p \text{ divides one term of } \mathrm{LC}(R)] + \Pr[\mathrm{LC}(R)(\alpha) = 0] \\ &\leq \frac{\deg(\mathrm{LC}(R))}{p} + \frac{\log_{p_{\min}} \|R\|}{N} \leq \frac{\deg(R)}{p} + \frac{\log_{p_{\min}} \|R\|}{N}. \end{aligned} \tag{5.5}$$

Case 2 : Since R is assumed to have only one square free factor S, we can write $R = \hat{w} S^k$ for some $k \geq 1$. Thus, the coefficients a_k of $\hat{f}$ vanishes for any k, if $f_k(\alpha) = 0$ and $g_k(\alpha) \neq 0$ or $p|f_k$ and $p \nmid g_k$ where f_k and g_k are the coefficients of S (as defined in (5.3)). Therefore,

$$\begin{aligned}&\Pr[\text{the coefficients } a_k = 0 \text{ for any } k] \\ &\leq \Pr[\prod_{k=0}^{T-1} f_k(\alpha) = 0] + \Pr[p \text{ divides one coefficient of } f_k \text{ for any } k] \\ &\leq \frac{\sum_{k=0}^{T-1} \deg(f_k)}{p} + \frac{T \log_{p_{\min}} \|f_k\|_\infty}{N} \\ &\leq \frac{T \deg(R)}{p} + \frac{T \log_{p_{\min}} \|f_k\|_\infty}{N} \\ &\leq \frac{\deg(R, x_1) \deg(R)}{p} + \frac{\deg(R, x_1) \log_{p_{\min}} \|S\|}{N}. \end{aligned} \tag{5.6}$$

Case 3 : Finally, if $g(x_1) \neq R(x_1, \alpha)$ then

$$\Pr[\mathbf{BB}\left((\theta, \alpha), p\right) - g(\theta) = 0] = \Pr[R(\theta, \alpha) - g(\theta)) = 0] \leq \frac{\deg(R, x_1)}{p}. \tag{5.7}$$

Adding (5.5), (5.6) and (5.7) proves our claim. □

Remark 5.5. The reader should note that bounds for $\|S\|_\infty$ and $\|R\|_\infty$ are provided in Theorem 6.5 of Chapter 6.

Next, we compute $\deg(f_k)$ and $\deg(g_k)$ for $0 \leq k \leq T-1$ using Algorithm 13. The reader should note that Lines 4-16 of Algorithm 13 were added to emphasize that, at step

Algorithm 13: TotalDegrees

Input: A prime p and a black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for the Dixon resultant $R \in \mathbb{Z}_p[x_1, y_1, y_2, \ldots, y_m]$, degree $\hat{D} = \deg(R, x_1)$, the list of degrees $[d_0, \ldots, d_T]$ and T.

Output: $(\deg(f_k), \deg(g_k) : 0 \le k \le T-1)$ with high probability where f_k and g_k are as defined in (5.2)

Pick $\beta \in \mathbb{Z}_p \setminus \{0\}^m, \delta \in \mathbb{Z}_p^{\hat{D}+1}$ and $\alpha \in (\mathbb{Z}_p \setminus \{0\})^m$ at random.
$(\overline{m}, t, r) \leftarrow (1, 0, 0)$
$h(z) \leftarrow$ FAIL
for $k = 0, 1, 2, \ldots, T-1$ **do**
  if $t \ge 2$ and $h(z) \ne$ FAIL **then**
    Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
    Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le i \le t)$
    $h(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u)$.
    if $h(z) \ne$ FAIL **then**
      Let $h(z) = \frac{a(z)}{b(z)} \in \mathbb{Z}_p(z)$ with $\gcd(a, b) = 1$.
      $(\deg(\bar{f}_k), \deg(\bar{g}_k)) \leftarrow (\deg(a), \deg(b))$
    end
  else
    for $t = r+1, r+2, \cdots$ **do**
      Pick $\theta_t \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\theta_t \notin \{\theta_1, \theta_2, \ldots, \theta_{t-1}\}$.
      for j from 0 to $\hat{D}$ **do**
        $W_j \leftarrow \mathbf{BB}\,((\delta_j, \theta_t\alpha_1 + \beta_1, \theta_t\alpha_2 + \beta_2, \cdots, \theta_t\alpha_m + \beta_m), p)$ //probes to the black box
      end
      Interpolate $B_t \in \mathbb{Z}_p[x_1]$ using points $(\delta_i, W_i : 0 \le i \le \hat{D})$; ..... $O(\hat{D}^2)$
      if $\deg(B_t, x_1) \ne \hat{D}$ **then return FAIL end**
      $B_t \leftarrow B_t/\text{lc}(B_t, x_1)$. // Make B_t monic.
      Compute $G_t = B_t/\gcd(B_t, \frac{dB_t}{dx_1})$; ..... $O(\hat{D}^2)$
      if $t \ge 2$ **then**
        Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
        Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le j \le t)$ ..... $O(t^2)$
        $\overline{m}(z) \leftarrow \overline{m} \times (z - \theta_t)$
        $h(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u) \ge 0$ ..... $O(t^2)$
        if $h(z) \ne$ FAIL **then**
          Let $h(z) = \frac{a(z)}{b(z)} \in \mathbb{Z}_p(z)$ with $\gcd(a, b) = 1$.
          $(\deg(\bar{f}_k), \deg(\bar{g}_k)) \leftarrow (\deg(a), \deg(b))$
          $r \leftarrow t$.
          break;
        end
      end
    end
  end
end
return $(\deg(\bar{f}_k), \deg(\bar{g}_k))$ for $0 \le k \le T-1$.

Algorithm 14: MaxPartialDegrees

Input: A prime p and a black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for the Dixon resultant $R \in \mathbb{Z}_p[x_1, y_1, y_2, \ldots, y_m]$, degree $\hat{D} = \deg(R, x_1)$, number of terms T, the list of degrees $[d_0, \ldots, d_T]$ and i satisfying $1 \le i \le m$.

Output: The maximum partial degree $\max_{k=0}^{T-1}(\max(\deg(f_k, y_i), \deg(g_k, y_i))$ of S with high probability where f_k, g_k and S are as defined in (5.2).

Pick $\beta \in \mathbb{Z}_p \setminus \{0\}, \delta \in \mathbb{Z}_p^{\hat{D}+1}$ and $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_m) \in (\mathbb{Z}_p \setminus \{0\})^{m-1}$ at random.
$(\overline{m}, t, r) \leftarrow (1, 0, 0)$
$h(z) \leftarrow$ FAIL
for $k = 0, 1, 2, \ldots, T-1$ **do**
  if $t \ge 2$ and $h(z) \ne$ FAIL **then**
    Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
    Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le i \le t)$
    $h(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u)$.
    if $h(z) \ne$ FAIL **then**
      Let $h(z) = \frac{a(z)}{b(z)} \in \mathbb{Z}_p(z)$ with $\gcd(a, b) = 1$.
      $e_k \leftarrow \max\{\deg(a), \deg(b)\}$.
    end
  else
    for $t = r+1, r+2, \cdots$ **do**
      Pick $\theta_t \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\theta_t \notin \{\theta_1, \theta_2, \ldots, \theta_{t-1}\}$.
      for j from 0 to $\hat{D}$ **do**
        $W_j \leftarrow \mathbf{BB}\,((\delta_j, \alpha_1, \alpha_2, \cdots, \alpha_{i-1}, \beta\theta_t, \alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_m), p)$. //probes to the black box
      end
      Interpolate $B_t \in \mathbb{Z}_p[x_1]$ using points $(\delta_i, W_i : 0 \le i \le \hat{D})$; ..... $O(\hat{D}^2)$
      if $\deg(B_t, x_1) \ne \hat{D}$ **then return FAIL end**
      $B_t \leftarrow B_t/\text{lc}(B_t, x_1)$. // Make B_t monic.
      Compute $G_t = B_t/\gcd(B_t, \frac{dB_t}{dx_1})$; ..... $O(\hat{D}^2)$
      if $t \ge 2$ **then**
        Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
        Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le j \le t)$ ..... $O(t^2)$
        $\overline{m}(z) \leftarrow \overline{m} \times (z - \theta_t)$
        $h(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u) \ge 0$ ..... $O(t^2)$
        if $h(z) \ne$ FAIL **then**
          Let $h(z) = \frac{a(z)}{b(z)} \in \mathbb{Z}_p(z)$ with $\gcd(a, b) = 1$.
          $e_k \leftarrow \max\{\deg(a), \deg(b)\}$.
          $r \leftarrow t$.
          break;
        end
      end
    end
  end
end
return $\max_{k=0}^{T-1}(e_k)$

Algorithm 15: MonoTotalDegrees

Input: A prime p and a black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for the Dixon resultant $R \in \mathbb{Z}_p[x_1, y_1, y_2, \ldots, y_m]$, degree $\hat{D} = \deg(R, x_1)$, number of terms T, list of degrees $[d_0, \ldots, d_T]$ and the total degrees $(\deg(f_k), \deg(g_k) : 0 \le k \le T-1)$.

Output: The list of degrees $[(E_{f_k}, E_{f_k}) : 0 \le k \le T-1)]$ with high probability where $E_{f_k} = [\deg(f_{i,k}) : 0 \le i \le \deg(f_k))]$ and $E_{g_k} = [\deg(g_{i,k}) : 0 \le i \le \deg(g_k))]$ where the homogeneous polynomials $f_{i,k}$ and $g_{i,k}$ are as defined in (5.3).

Pick $\delta \in \mathbb{Z}_p^{\hat{D}+1}$ and $\alpha \in (\mathbb{Z}_p \setminus \{0\})^m$ at random.
Initialize $(\overline{m}, t, r) \leftarrow (1, 0, 0)$ and initialize $h_{-1}(z) \leftarrow$ FAIL.
for $k = 0, 1, 2, \ldots, T-1$
  if $t \ge 2$ and $h_{k-1}(z) \ne$ FAIL **then**
    Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
    Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le i \le t)$
    $h_k(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u)$.
    if $h_k(z) \ne$ FAIL **then**
      Let $h_k(z) = \frac{a_k(z)}{b_k(z)}$ such that $\gcd(a_k, b_k) = 1$ where $a_k = \sum_{i=0}^{\deg(a_k)} \bar{a}_{i,k}$ and $b_k = \sum_{i=0}^{\deg(b_k)} \bar{b}_{i,k}$ and $\deg(a_{i,k}) = \deg(b_{i,k}) = i$.
      $(e_{k_1}, e_{k_2}) \leftarrow (\deg(f_k) - \deg(a_k), \deg(g) - \deg(b_k))$
      if $e_{k_1} = e_{k_2}$ **then return** (A_k, B_k) for $0 \le k \le T-1$ where
        $A_k = [\deg(a_{i,k}) + e_{k_1} : 0 \le i \le \deg(f_k))]$ // $\deg(a_{i,k}) = \deg(f_{i,k})$.
        $B_k = [\deg(b_{i,k}) + e_{k_1} : 0 \le i \le \deg(g_k))]$ // $\deg(b_{i,k}) = \deg(g_{i,k})$.
      else return FAIL;
    end
  end
  else
    for $t = r+1, r+2, \cdots$ **do**
      Pick $\theta_t \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\theta_t \notin \{\theta_1, \theta_2, \ldots, \theta_{t-1}\}$.
      $W_j \leftarrow \mathbf{BB}\,((\delta_j, \alpha_1\theta_t, \alpha_2\theta_t \cdots, \alpha_m\theta_t), p)$ for $0 \le j \le \hat{D}$. //probes to **BB**
      Interpolate $B_t \in \mathbb{Z}_p[x_1]$ using points $(\delta_i, W_i : 0 \le i \le \hat{D})$; ..... $O(\hat{D}^2)$
      if $\deg(B_t, x_1) \ne \hat{D}$ **then** return **FAIL end**
      $B_t \leftarrow B_t/\text{lc}(B_t, x_1)$. // Make B_t monic.
      Compute $G_t = B_t/\gcd(B_t, \frac{dB_t}{dx_1})$; ..... $O(\hat{D}^2)$
      if $t \ge 2$ **then**
        Let $E = \{\text{Coeff}(G_j, x_1^{d_k}) : 1 \le j \le t\}$
        Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, E_i : 1 \le j \le t)$ ..... $O(t^2)$
        $\overline{m}(z) \leftarrow \overline{m} \times (z - \theta_t)$
        $h_k(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$ //$\deg(\overline{m}) > \deg(u) \ge 0$ ..... $O(t^2)$
        if $h_k(z) \ne$ FAIL **then**
          Let $h_k(z) = \frac{a_k(z)}{b_k(z)}$ such that $\gcd(a_k, b_k) = 1$ where $a_k = \sum_{i=0}^{\deg(a_k)} \bar{a}_{i,k}$ and $b_k = \sum_{i=0}^{\deg(b_k)} \bar{b}_{i,k}$ and $\deg(a_{i,k}) = \deg(b_{i,k}) = i$.
          $(e_{k_1}, e_{k_2}) \leftarrow (\deg(f_k) - \deg(a_k), \deg(g) - \deg(b_k))$
          if $e_{k_1} = e_{k_2}$ **then return** (A_k, B_k) for $0 \le k \le T-1$ where
            $A_k = [\deg(a_{i,k}) + e_{k_1} : 0 \le i \le \deg(f_k))]$ // $\deg(a_{i,k}) = \deg(f_{i,k})$.
            $B_k = [\deg(b_{i,k}) + e_{k_1} : 0 \le i \le \deg(g_k))]$ // $\deg(b_{i,k}) = \deg(g_{i,k})$.
          else return FAIL;
          end
          $r \leftarrow t$.
          break;
        end
      end
    end
  end
end
return $[(A_k, B_k) : 0 \le k \le T-1]$

k of the main for loop, the coefficients of the previous computed polynomial images G_t at step $k-1$ can be re-used if sufficient. More probes to the black box **BB** are only necessary when Algorithm 13 needs more polynomial images of R to discover the degrees.

We remark that it is possible that our algorithms for pre-computing the needed degrees may output a smaller degree value but never a larger degree value. Finally, we compute the maximum partial degrees $\max_{k=0}^{T-1}(\max(\deg(f_k, y_i), \deg(g_k, y_i))$ of S as defined in (5.2) with respect to each variable y_i for $1 \leq i \leq m$ and the the total degrees of the homogeneous polynomials $f_{i,k}$ and $g_{i,k}$ using Algorithms 14 and 15 respectively.

5.3 Algorithm DixonRes

We now present our Dixon resultant algorithm labelled as Algorithm DixonRes (Algorithm 19). It calls Algorithms NewPrime (Algorithm 20) and MQRFR, and Subroutines Poly-Interp, RatFun, Remove-Shift, VandermondeSolver and BMStep. The MQRFR algorithm (Algorithm 8) is the Maximal Quotient Rational Function Reconstruction algorithm. Algorithm 19 to interpolate S involves eight major steps, namely:

1. The computation of the degrees $[d_0, \ldots, d_T]$ as defined in (5.2) by Algorithm 12, the total degrees of the polynomials f_k, g_k by Algorithm 13, the maximum partial degrees $D_{y_i} = \max(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i)))$ of S as defined in (5.2) for $1 \leq i \leq m$ and the total degrees of the homogeneous polynomials f_k, g_k by Algorithm 15 in Lines 1-5 of Algorithm 19.

2. The use of a Kronecker substitution (Lines 5-7) to reduce the interpolation of the multivariate rational function coefficients $\frac{f_k}{g_k}$ of S to a univariate rational function interpolation. This consequently leads to a reduction in the size of the prime needed by our Dixon resultant algorithm, because the prime p needed by our algorithm must satisfy $p > \prod_{i=1}^{m}(D_{y_i}+1)$, which is typically much smaller than the prime required for the Ben-Or/Twari sparse interpolation algorithm to work in practice.

3. The selection of a basis shift $\beta \neq 0 \in \mathbb{Z}_p^m$ if needed by our Dixon resultant algorithm in Lines 8-13. A non-zero basis shift may not be needed, if there is a non-zero integer in the leading coefficient of the Dixon resultant $R = \det(M)$ in x_1, that the input prime does not divide. With high probability, we are assured of the presence of a constant term in the denominator polynomials g_k, which is needed for normalizing the corresponding auxiliary rational functions. We detect if a basis shift is needed at the start of the algorithm by checking if the degree of $R(x_1, \beta)$ where $\beta = (0, 0, \cdots 0) \in \mathbb{Z}_p^m$ is equal to $\deg(R, x_1)$ using random evaluation points for x_1 via the black box representing R. If both degrees are the same then our algorithm does not need a basis shift.

Example 5.6. If $R = (y_1 - y_2)x_1^2 + x_1 + 5$ then we need a basis shift since $\deg(R(x_1, 0, 0), x_1) = 1 < \deg(R, x_1)$. But if $R = (y_1 - y_2 + 7)x_1^2 + x_1 + 5$, and the input prime $p \neq 7$ then we do not need a basis shift since $\deg(R(x_1, 0, 0), x_1) = \deg(R, x_1)$.

4. The interpolation of many univariate monic square-free polynomial images H_i in x_1 of S via probes to the black box **BB**. This is done by calls to Subroutine PolyInterp in Line 20. We remark that H_i in Line 20 is a list of $e_{\max}$ monic polynomial images in x_1, since we need at most $e_{\max} = 2 + \max_{k=0}^{T-1} \deg(f_k) + \deg(g_k)$ points to produce an auxiliary rational function $A_j = \frac{N_j}{\hat{N}_j} \in \mathbb{Z}_p(z)$ where $\deg(N_j) = \deg(f_k)$ and $\deg(\hat{N}_j) = \deg(g_k)$ with high probability using Algorithm 8.

5. The dense interpolation of auxiliary univariate rational functions A_j (Think of $A_j = F_k(\alpha^{s+j}, z, \beta)$ where F_k is as defined in (5.4)) using the coefficients of the monic images H_i. These univariate rational functions are the intermediate functions whose coefficients are used to interpolate the rational function coefficients of S, and they are produced in Line 24 via calls to Subroutine Ratfun which uses Algorithm 8. By design, these univariate rational functions must have a constant term in their denominator, so a basis shift β may be needed to force the production of a constant term (See Lines 8-13).

6. The discovery of the number of terms in the rational function coefficients of S using the Berlekamp Massey algorithm (BMA). By design, the leading term polynomials $f_{\deg(f_k),k}$ and $g_{\deg(g_k),k}$, referred to as F_k and G_k, respectively, in Lines 28-29 are interpolated first by calls to Subroutine BMStep, before the lower total degree polynomial terms can be interpolated. In Subroutine BMStep, the size of the supports $\#F_k$ and $\#G_k$ are discovered with high probability when the BMA returns the feedback polynomials, say $\lambda_1, \lambda_2 \in \mathbb{Z}_p[z]$ respectively. The roots of λ_1 and λ_2, determine the supports of $K_r(F_k)$ and $K_r(G_k)$ in y. The univariate functions $K_r(F_k)$ and $K_r(G_k)$ in y are the mapped images of F_k and G_k since a Kronecker substitution K_r was used.

Note that Subroutine BMStep generates a feedback polynomial $\lambda(z)$ using an input P, a sequence of coefficients of length i, collected from the coefficients of the auxiliary rational functions A_j. Line 2 of Subroutine BMStep will not cause the algorithm to return FAIL if $\deg(\lambda, z) < \frac{i}{2}$. The condition $\deg(\lambda, z) < \frac{i}{2}$ ensures that $\lambda(z)$ is correct with high probability. Otherwise, it returns FAIL indicating that we do not have the correct term bound, so more univariate polynomial images and auxiliary rational functions are needed. Algorithm 19 then computes more polynomial images and auxiliary rational functions, so that the process is repeated until a new term bound is found.

The next step is to assemble polynomials F_k and G_k by solving for their coefficients using Subroutine VandermondeSolver (Section 4.3.3), an algorithm that solves shifted transposed Vandermonde systems using Zippel's quadratic algorithm.

7. The interpolation of the lower total degree polynomials $f_{i,k}$ and $g_{i,k}$ in Lines 31-32 by calls to Subroutine RemoveShift. Before the polynomials $f_{i,k}$ and $g_{i,k}$ can be interpolated, Subroutine RemoveShift adjusts the coefficients of the auxiliary rational functions in order to remove the effect of the basis shift β that was contributed by the higher total degree polynomials $f_{j,k}$ and $g_{j,k}$ for $j > i$, whenever $\beta \neq 0$.

8. Performing sparse interpolation using additional primes in Line 41 whenever the rational number reconstruction process fails on the integer coefficients of the first image of S for the first prime. Algorithm 20 (an algorithm similar to Algorithm 19 which does not use a Kronecker substitution K_r since the first image of S has been found) uses the support obtained for the first prime to get more images if additional primes are needed to recover S. We remark that one 62 bit prime was often enough to interpolate the R_j's in our benchmark systems. Finally, we check if the returned answer is correct using a probabilistic approach.

Subroutine 16: PolyInterp

Input: A prime p and the black box $\mathbf{BB} : \left(\mathbb{Z}_p^{m+1}, p\right) \rightarrow \mathbb{Z}_p$ for the Dixon resultant $R = \det(M)$ with $m \geq 1$, a list of $m-$ tuple evaluation points $Z = [Z_j \in \mathbb{Z}_p^m : 1 \leq j \leq e_{\max}]$, degree $e_{\max} \geq 2$, list of degrees $d = [d_0, \ldots, d_T]$ as defined in (5.2) and degree $\hat{D} = \deg(R, x_1)$.

Output: A list of $e_{\max}$ monic univariate polynomials $H = [\text{monic}(H_j) \in \mathbb{Z}_p[x_1] : 1 \leq j \leq e_{\max}]$ or **FAIL**.

Pick $\delta \in \mathbb{Z}_p^{\hat{D}+1}$ at random with $\delta_i \neq \delta_j$ for $i \neq j$.
for $j = 1, 2, \ldots, e_{max}$ **do**
  Compute $G_j = (\mathbf{BB}((\delta_i, Z_j), p) : 1 \leq i \leq \hat{D}+1)$. // $\hat{D}+1$ probes to the black box **BB**.
  Interpolate $B_j \in \mathbb{Z}_p[x_1]$ using points $(\delta_i, G_{j,i} : 1 \leq i \leq \hat{D}+1)$; ..... $O(\hat{D}^2)$
  if $\deg(B_j, x_1) < \hat{D}$ **then** return **FAIL end**
  Compute the square-free part $H_j = B_j / \gcd(B_j, \frac{dB_j}{dx_1})$; ..... $O(\hat{D}^2)$
  if $\text{supp}(H_j) \neq d$ **then** return **FAIL end**
end
return $[\text{monic}(H_1), \ldots, \text{monic}(H_{e_{\max}})]$.

Subroutine 17: BMStep

Input: A list of points $P = [P_j \in \mathbb{Z}_p : 1 \leq j \leq i \text{ and } i \text{ is even}]$, a generator α for $\mathbb{Z}_p^*$, a random shift $\hat{s} \in [0, p-2]$ and r which defines the Kronecker substitution K_r.

Output: A non-zero multivariate polynomial $\bar{F} \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m]$ or **FAIL**.

Run the Berlekamp-Massey algorithm [Atti et al., 2006] on P to obtain $\lambda(z) \in \mathbb{Z}_p[z]$; $O(i^2)$
if $\deg(\lambda, z) = \frac{i}{2}$ **then** return **FAIL end** // More images are needed
Compute the roots of λ in $\mathbb{Z}_p[z]$ to obtain the monomial evaluations $\hat{m}_i$. Let $\hat{m} \subset \mathbb{Z}_p$ be the set of monomial evaluations $\hat{m}_i$ and let $t = |\hat{m}|$; ..... $O(t^2 \log p)$
if $t \neq \deg(\lambda, z)$ **then** return **FAIL end** // $\lambda(z)$ is wrong.
Solve $\alpha^{e_i} = \hat{m}_i$ for e_i with $e_i \in [0, p-2]$ // The exponents are found here.
Let $\hat{M} = [\, y^{e_i} : i = 1, 2 \cdots, t\,]$ // These are the monomials
$F \leftarrow$VandermondeSolver $(\hat{m}, [P_1, \cdots P_t], \hat{s}, \hat{M})$ // $F \in \mathbb{Z}_p[y]$; ..... $O(t^2)$
$\bar{F} \leftarrow K_r^{-1}(F) \in \mathbb{Z}_p[y_1, \ldots, y_m]$.// Invert the Kronecker map K_r.
return $\bar{F}$

Algorithm 19: DixonRes

Input: The black box $\mathbf{BB} : \left(\mathbb{Z}_p^{m+1}, p\right) \to \mathbb{Z}_p$ for the Dixon resultant $R = \det(M) \in \mathbb{Q}[x_1, y_1, y_2, \ldots, y_m]$ with $m \geq 1$.

Output: The monic square-free factor $S \in \mathbb{Q}(y_1, \ldots, y_m)[x_1]$ of R where S is as defined in (5.2) or **FAIL**.

Compute $d = [d_0, \ldots, d_T]$ as defined in (5.2) and $\hat{D} = \deg(\det(M), x_1)$ using Algorithm 12.
Compute $\deg(f_k)$ and $\deg(g_k)$ for $0 \leq k \leq T-1$ as defined in (5.2) using Algorithm 13.
Let $e_{\max} = \max_{k=0}^{T-1} e_k$ where $e_k = \deg(f_k) + \deg(g_k) + 2$ and assume $e_0 \geq e_1, \cdots \geq e_{T-1}$.
Compute $D_{y_i} = \max_{k=0}^{T-1} (\max(\deg(f_k, y_i), \deg(g_k, y_i)))$ for $1 \leq i \leq m$ using Algorithm 14.
Initialize $r_i = D_{y_i} + 1$ for $1 \leq i \leq m$.
Compute $[(E_{f_k}, E_{f_k} : 0 \leq k \leq T-1)]$ using Algorithm 15 where $E_{f_k} = [\deg(f_{i,k}) : 0 \leq i \leq \deg(f_k))]$ and $E_{g_k} = [\deg(g_{i,k}) : 0 \leq i \leq \deg(g_k))]$ where the homogeneous polynomials $f_{i,k}$ and $g_{i,k}$ are as defined in (5.3).
Pick a random smooth prime $p > \prod_{j=1}^{m} r_i$. // p is the prime to be used by **BB**.
Let $K_r : \mathbb{Z}_p(y_1, \ldots, y_m)[x_1] \to \mathbb{Z}_p(y)[x_1]$ be the Kronecker substitution $K_r(S)$ where S is as defined in (5.2)) and $r = (r_1, r_2, \ldots, r_{m-1})$.
Let $\beta = (0, 0, \ldots, 0) \in \mathbb{Z}_p^m$.
Interpolate $G = R(x_1, \beta)$ using $\hat{D} + 1$ points for x_1 via black box **BB**; ..... $O(\hat{D}^2)$
while $\deg(G) < \hat{D}$ **do**
    Choose a random basis shift $\beta \in \mathbb{Z}_p^m$.
    Interpolate $G = R(x_1, \beta)$ using $\hat{D} + 1$ points for x_1 via **BB**; ..... $O(\hat{D}^2)$
end
Pick a random shift $\hat{s} \in [0, p-2]$ and any generator α for $\mathbb{Z}_p^*$.
Pick $\theta \in \mathbb{Z}_p^{e_{\max}}$ at random with $\theta_i \neq \theta_j$ for $i \neq j$.
Let z be the homogenizing variable and initialize $k = 0$.
for $i = 1, 2, \cdots$ **while** $k \leq T-1$
    $\hat{Y}_i \leftarrow (\alpha^{\hat{s}+i-1}, \alpha^{(\hat{s}+i-1)r_1}, \ldots, \alpha^{(\hat{s}+i-1)(r_1 r_2 \cdots r_{m-1})})$. // Implements K_r
    Let $Z = [\hat{Y}_i \theta_j + \beta \in \mathbb{Z}_p^m : 1 \leq j \leq e_{\max}]$ be the evaluation points.
    // Compute the monic univariate images $H_i \in \mathbb{Z}_p[x_1]$ where $|H_i| = e_{\max}$.
    $H_i \leftarrow \text{PolyInterp}\left(\mathbf{BB}, (Z, p), e_{\max}, d, \hat{D}\right)$ ..... $O(e_{\max}\hat{D}^2)$
    if $H_i =$ **FAIL then** return **FAIL end**
    if $i \in \{2, 4, 8, 16, 32, \cdots\}$ **then**
        for $j = 1, 2, \ldots, i$ **do**
            // Compute the auxiliary rational functions $A_j(\alpha^{\hat{s}+j-1}, z) = \frac{N_j}{\hat{N}_j} \in \mathbb{Z}_p(z)$
            $A_j \leftarrow \text{RatFun}(H_j, \theta, d_k, e_k, p)$; ..... $O(e_k^2)$
            if $\deg(N_j, z) \neq \deg(f_k)$ or $\deg(\hat{N}_j, z) \neq \deg(g_k)$ **then** return **FAIL end**
            // $\alpha^{\hat{s}+j-1}$ is a bad evaluation point or β is a bad basis shift or prime p is unlucky (see Definitions 6.21 & 6.22.)
        end
        Let $F_k = f_{\deg(f_k),k}$ and $G_k = g_{\deg(g_k),k}$ as defined in (5.3).
        $F_k \leftarrow \text{BMStep}([\text{coeff}(N_j, z^{\deg(f_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r)$; ..... $O(i^2 + \#F_k^2 \log p)$
        $G_k \leftarrow \text{BMStep}([\text{coeff}(\hat{N}_j, z^{\deg(g_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r)$; ..... $O(i^2 + \#G_k^2 \log p)$
        if $F_k \neq$ **FAIL** and $G_k \neq$ **FAIL then**
            $f_k \leftarrow \text{RemoveShift}(F_k, \beta, E_{f_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [N_1, \ldots, N_i], r)$
            $g_k \leftarrow \text{RemoveShift}(G_k, \beta, E_{g_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [\hat{N}_1, \ldots, \hat{N}_i], r)$
            if $f_k \neq$ **FAIL** and $g_k \neq$ **FAIL then**
                $k \leftarrow k + 1$// The k-th rational function coefficient of S is interpolated.
            end
        end
    end
end
$\hat{S} \leftarrow x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)} x_1^{d_k}$ // Assemble $\hat{S} = S \bmod p$ where S is as defined in (5.2)
Apply rational number reconstruction on the coefficients of $\hat{S} \bmod p$ to get S
if $S =$ FAIL **then** $S \leftarrow$NewPrime($\mathbf{BB}, \hat{S}, d, \hat{D}, p$) **else return** S **end**

Subroutine 18: RemoveShift

Input: A multivariate polynomial $F_k \in \mathbb{Z}_p[y_1, \ldots, y_m]$, a basis shift $\beta \in \mathbb{Z}_p^m$, list of degrees E_{f_k}, a random shift $\hat{s} \in [0, p-2]$, a generator α for $\mathbb{Z}_p^*$, a list of m-tuple evaluation points $[\hat{Y}_j \in \mathbb{Z}_p^m : 1 \le j \le i]$, a list of univariate polynomials $[N_j \in \mathbb{Z}_p[z] : 1 \le j \le i]$ and r which defines the Kronecker substitution K_r.

Output: A polynomial $f_k \in \mathbb{Z}_p[y_1, \ldots, y_m]$ where f_k is as defined in (5.3) or **FAIL**

$(\overline{A}, f_k, d) \leftarrow (F_k, F_k, \deg(F_k))$
Initialize $H_j = 0$ for $1 \le j \le i$.
for $\bar{d} \in E_{f_k}$ **do**
  if $\beta \neq 0$ **then**
    Pick $\theta \in \mathbb{Z}_p^{d+1}$ at random.
    for $j = 1, 2, \cdots, i$ **do**
      for $t = 1, 2, \ldots, d+1$ **do**
        Let $Z_{j,t} = \overline{A}(y_1 = \hat{Y}_{j,1}\theta_t + \beta_1, \ldots, y_m = \hat{Y}_{j,m}\theta_t + \beta_m)$ be the polynomial evaluations of $\overline{A}$ ..... $O(md\#\overline{A})$.
      end
      Interpolate $\overline{W}_j \in \mathbb{Z}_p[z]$ using points $(\theta_t, Z_{j,t} : 1 \le t \le d+1)$; ..... $O(d^2)$
      $H_j \leftarrow H_j + \overline{W}_j$; ..... $O(d)$
    end
  end
  if $\bar{d} \neq 0$ **then**
    $P \leftarrow \left[\text{coeff}(N_j, z^{\bar{d}} : 1 \le j \le i\right]$.
    if $\beta \neq 0$ **then**
      for $j = 1, 2, \cdots, i$ **do**
        $P_j \leftarrow P_j - \text{coeff}(H_j, z^{\bar{d}})$
        // The P_j's are adjusted to remove the effect of the basis shift β.
      end
    end
    if $[P_j = 0 : 1 \le j \le i]$ **then**
      $\overline{A} \leftarrow 0$ // There is no monomial of total degree $\hat{d}$.
    else
      $\overline{A} \leftarrow \text{BMStep}([P_1, \ldots, P_i], \alpha, \hat{s}, r)$; ..... $O(i^2 + \#\overline{A}^2 \log p)$
      if $\overline{A} =$ **FAIL then return FAIL end** // More P_j's are needed.
    end
  else
    $\overline{A} \leftarrow \text{coeff}(N_1, z^0)$ // We get the constant term.
    if $\beta \neq 0$ then $\overline{A} \leftarrow \overline{A} - \text{coeff}(\Gamma_1, z^0)$ **end**
  end
  $(f_k, d) \leftarrow (f_k + \overline{A}, \hat{d} + 1)$.
end
return f_k

Algorithm 20: NewPrime

Input: The black box $\mathbf{BB} : \left(\mathbb{Z}_q^{m+1}, q\right) \to \mathbb{Z}_q$ for the Dixon resultant R, the first image $\hat{S} = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1,y_2,\ldots,y_m)}{g_k(y_1,y_2,\ldots,y_m)} x_1^{d_k} \in \mathbb{Z}_p(y_1,\ldots,y_m)[x_1]$ of S obtained from Algorithm 19 with respect to a smooth prime p where S is as defined in (5.2), the list of degrees $d = \{d_0, d_1, \ldots, d_T\}$ and $\hat{D} = \deg(R, x_1) > 0$.

Remark : Additional primes are required to interpolate the monic square-free factor of R.

Output: The monic square-free factor $\bar{F} \in \mathbb{Q}(y_1,\ldots,y_m)[x_1]$ of R or **FAIL**.

Let $B_1 = [f_{\deg(f_k)-1,k}, \ldots, f_{0,k}]$ and $B_2 = [g_{\deg(g_k)-1,k}, \ldots, g_{0,k}]$ where $f_{i,k}, g_{i,k}$ are as defined in (5.3).
$e_k \leftarrow \deg(f_k) + \deg(g_k) + 2$ for $0 \le k \le T-1$.
Let $e_{\max} = \max_{k=0}^{T-1} e_k$ and let $P \leftarrow p$.
Let $N_{\max} = \max_{k=0}^{T-1} \left\{ \max_{0 \le i \le \deg(f_k)} \{\# f_{i,k}\}, \max_{0 \le i \le \deg(g_k)} \{\# g_{i,k}\} \right\}$.
do
  Get a new 62 bit prime $q \neq p$. // The black box **BB** uses a new prime q.
  Let $\beta = (0, 0, \ldots, 0) \in \mathbb{Z}_q^m$.
  Interpolate $G = R(x_1, \beta)$ using $\hat{D}+1$ points for x_1 via black box **BB**; ..... $O(\hat{D}^2)$
  while $\deg(G) < \hat{D}$ **do**
    Choose a random basis shift $\beta \in \mathbb{Z}_q^m$.
    Interpolate $G = R(x_1, \beta)$ using $\hat{D}+1$ points for x_1 via **BB**; ..... $O(\hat{D}^2)$
  end
  Pick $\alpha \in (\mathbb{Z}_q \setminus \{0\})^m, \theta \in \mathbb{Z}_q^{e_{\max}}$ and shift $\hat{s} \in [0, q-2]$ at random.
  for $i = 1, 2, \ldots, N_{\max}$ **do**
    Let $\hat{Y}_i = (\alpha_1^{\hat{s}+i-1}, \alpha_2^{\hat{s}+i-1} \cdots, \alpha_m^{\hat{s}+i-1})$. // No Kronecker substitution is required.
    Let $Z = [\theta_j \hat{Y}_i + \beta \in \mathbb{Z}_q^m : 1 \le j \le e_{\max}]$ be the evaluation points.
    $H \leftarrow \text{PolyInterp}\left(\mathbf{BB}\left(Z, q\right), d, e_{\max}, \hat{D}\right)$ //$|H| = e_{\max}$; ..... $O(e_{\max}\hat{D}^2)$
    if $H =$ **FAIL** then return **FAIL end**
  end
  for $k = 0, 1, \ldots, T-1$ **do**
    $(\hat{n}, \hat{M}) \leftarrow (\# f_{\deg(f_k),k}, \text{supp}(f_{\deg(f_k),k}))$
    $(\bar{n}, \bar{M}) \leftarrow (\# g_{\deg(g_k),k}, \text{supp}(g_{\deg(g_k),k}))$
    $(\hat{m}, \bar{m}) \leftarrow ([\hat{M}_i(\alpha) : 1 \le i \le \hat{n}], [\bar{M}_i(\alpha) : 1 \le i \le \bar{n}])$; .... $O(m(\deg(f_k)\hat{n} + \deg(g_k)\bar{n}))$
    if the monomial evaluations $\hat{m}_i = \hat{m}_j$ or $\bar{m}_i = \bar{m}_j$ **then** return **FAIL end.**
    for $j = 1, 2, \ldots, N_{\max}$ **do**
      $B_j \leftarrow \text{RatFun}(H_j, \theta, d_k, e_k, q)$// Let $B_j = N_j(z)/\hat{N}_j(z) \in \mathbb{Z}_q(z)$. ..... $O(e_k^2)$
      if $\deg(N_j, z) \neq \deg(f_k)$ or $\deg(\hat{N}_j, z) \neq \deg(g_k)$ **then** return **FAIL**
    end
    Let $a_i = \text{LC}(N_j, z)$ and let $b_i = \text{LC}(\hat{N}_j, z)$ for $1 \le i \le N_{\max}$.
    $F_k \leftarrow$VandermondeSolver$(\hat{m}, [a_1, \ldots, a_{\hat{n}}], \hat{s}, \hat{M})$; ..... $O(\hat{n}^2)$
    $G_k \leftarrow$VandermondeSolver$(\bar{m}, [b_1, \ldots, b_{\bar{n}}], \hat{s}, \bar{M})$; ..... $O(\bar{n}^2)$
    $F_k \leftarrow \text{GetTerms}(F_k, \alpha, \beta, \hat{s}, B_1, \hat{Y}_1, \ldots, \hat{Y}_{N_{\max}}], [N_1, \ldots, N_{N_{\max}}], q)$
    $G_k \leftarrow \text{GetTerms}(G_k, \alpha, \beta, \hat{s}, B_2, [\hat{Y}_1, \ldots, \hat{Y}_{N_{\max}}], [\hat{N}_1, \ldots, \hat{N}_{N_{\max}}], q)$
    if $F_k =$ **FAIL** or $G_k =$ **FAIL then** return **FAIL end**
  end
  $\hat{T} \leftarrow x_1^{d_T} + \sum_{k=0}^{T-1} \frac{F_k(y_1,y_2,\ldots,y_m)}{G_k(y_1,y_2,\ldots,y_m)} x_1^{d_k} \in \mathbb{Z}_q(y_1,\ldots,y_m)[x_1]$
  Solve $\left\{ \hat{F} \equiv \hat{S} \bmod P \text{ and } \hat{F} \equiv \hat{T} \bmod q \right\}$ using Chinese remaindering.
  Set $P = P \times q$. // Product of primes
  Apply rational number reconstruction on the coefficients of $\hat{F}$ mod P to get $\overline{F}$.
  if $\overline{F} \neq$ **FAIL then** return $\overline{F}$ **else** $(\hat{S}, p) \leftarrow (\hat{F}, q)$ **end**
end

Subroutine 21: GetTerms

Input: A multivariate polynomial $F_k \in \mathbb{Z}_q[y_1, \ldots, y_m]$, evaluation points $\alpha \in (\mathbb{Z}_q \setminus \{0\})^m$, $\beta \in \mathbb{Z}_q^m$, a random shift $\hat{s} \in [0, q-2]$, list of lower total degree polynomials $B_1 = [f_{\deg(f_k)-1,k}, \ldots, f_{0,k}]$ obtained using the first prime from Algorithm 19, a list of $m-$ tuple evaluation points $[\hat{Y}_j \in \mathbb{Z}_q^m : 1 \le j \le N_{\max}]$, a list of univariate polynomials $[N_j \in \mathbb{Z}_q[z] : 1 \le j \le N_{\max}]$ and a prime q.

Output: A polynomial $\overline{f}_k = f_k \bmod q$ where f_k is as defined in (5.3) or **FAIL**.

$(\overline{A}, \overline{f}_k, \hat{d}) \leftarrow (F_k, F_k, \deg(F_k))$.
Set $H = (0, 0, , \ldots, 0) \in \mathbb{Z}_q^{N_{\max}}$.
$\overline{D} \leftarrow [\deg(e) : e \in B_1]$, $\quad \hat{M} \leftarrow [\text{supp}(e) : e \in B_1]$ // supp means support.
for $h = 1, 2, \ldots, |\overline{D}|$ **do**
  $d \leftarrow \overline{D}_h$
  if $\beta \neq 0$ **then**
    Pick $\theta \in \mathbb{Z}_q^{\hat{d}+1}$ at random.
    for $j = 1, 2, \cdots, N_{\max}$ **do**
      for $t = 1, 2, \ldots, \hat{d}+1$ **do**
        $Z_{j,t} \leftarrow \overline{A}(y_1 = \hat{Y}_{j,1}\theta_t + \beta_1, \ldots, y_m = \hat{Y}_{j,m}\theta_t + \beta_m)$; ..... $O(m\hat{d}\#\overline{A})$
      end
      Interpolate $\overline{W}_j \in \mathbb{Z}_q[z]$ using points $(\theta_t, Z_{j,t} : 1 \le t \le \hat{d}+1)$; ..... $O(\hat{d}^2)$
      $H_j \leftarrow H_j + \overline{W}_j$; ..... $O(\hat{d})$
    end
  end
  if $d \neq 0$ **then**
    $P \leftarrow \left[\text{coeff}(N_j, z^d) : 1 \le j \le N_{\max}\right]$
    if $\beta \neq 0$ **then**
      for $j = 1, 2, \ldots, N_{\max}$ **do**
        $P_j \leftarrow P_j - \text{coeff}(H_j, z^d)$
      end
    end
    $\hat{m} \leftarrow [\hat{M}_i(\alpha) : 1 \le i \le \hat{n}]$ where $\hat{n} = \#\hat{M}_h$; ..... $O(m\hat{n}\hat{d})$
    if any monomial evaluations $\hat{m}_i = \hat{m}_j$ **then** return **FAIL end.**
    $\overline{A} \leftarrow \text{VandermondeSolver}(\hat{m}, P, \hat{s}, \hat{M}_h)$; ..... $O(\hat{n}^2)$
  else
    $\overline{A} \leftarrow \text{coeff}(N_1, z^0)$ // We use only one point to get the constant term
    if $\beta \neq 0$ **then**
      $\overline{A} \leftarrow \overline{A} - \text{coeff}(\Gamma_1, z^0)$
    end
    $(\overline{f}_k, \hat{d}) \leftarrow (\overline{f}_k + \overline{A}, \deg(\overline{A}) + 1)$.
  end
end
return $\overline{f}_k$.

Subroutine 22: RatFun

Input: A prime p and a list of univariate polynomials $H = [H_j \in \mathbb{Z}_p[x_1] : 1 \leq j \leq e_{\max}]$, an evaluation point $\theta \in \mathbb{Z}_p^{e_{\max}}$, degree d_k and degree e_k such that $2 \leq e_k \leq e_{\max}$.

Output: A univariate rational function $A(z) = \frac{N(z)}{\hat{N}(z)} \in \mathbb{Z}_p(z)$.

$\overline{m}(z) \leftarrow \prod_{i=1}^{e_k}(z - \theta_i) \in \mathbb{Z}_p[z]$; ..... $O(e_k)$

Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, \text{coeff}(H_i, x_1^{d_k}) : 1 \leq i \leq e_k)$; ..... $O(e_k^2)$

$A(z) \leftarrow \text{MQRFR}(\overline{m}, u, p)$; ..... $O(e_k^2)$

Let $A(z) = \frac{N(z)}{\hat{N}(z)} \in \mathbb{Z}_p(z)$ and normalize $A(z)$ s.t. $\hat{N}(z) = 1 + \sum_{j=1}^{\deg(\hat{N},z)} a_j z^j$.

return $A(z)$.

Subroutine 23: VandermondeSolver

Input: Vectors $\hat{m}, v \in \mathbb{Z}_p^t$, a random shift $\hat{s} \in [0, p-2]$ and monomials $[M_1, \ldots, M_t]$

Output: A non-zero multivariate polynomial $F = \sum_{i=1}^t a_i M_i \in \mathbb{Z}_p[y_1, \ldots, y_m]$.

Let $V_{ij} = \hat{m}_i^{\hat{s}+j-1}$ for $1 \leq i, j \leq t$. // A shifted transposed Vandermonde matrix

Solve the shifted transposed Vandermonde system $Va = v$ using Zippel's $O(t^2)$ algorithm.

Compute $a_i = \frac{a_i}{\hat{m}_i^{\hat{s}}}$ for $1 \leq i \leq t$.

return $F = \sum_{i=1}^t a_i M_i$

5.3.1 Probabilistic Test

The following algorithm (Algorithm 24) uses a probabilistic strategy to determine if the returned R_j's by our Dixon resultant algorithm are correct with high probability.

Algorithm 24: CheckResultant

Input: The black box $\mathbf{BB} : \left(\mathbb{Z}_q^{m+1}, q\right) \to \mathbb{Z}_q$ for the Dixon resultant $R \in \mathbb{Z}_q[x_1, y_1, y_2, \ldots, y_m]$, $\deg(R, x_1)$ and the monic square-free factors $R_j \in \mathbb{Q}(y_1, y_2, \ldots, y_m)[x_1]$ as defined in (5.1) such that $\deg(R_j, x_1) > 0$ for $1 \leq j \leq l$.

Output: true (if all the R_j's are correct) or false otherwise.

repeat
  Pick a new 62 bit prime q at random.
  Pick $\alpha \in \mathbb{Z}_q^m$ at random.
  Pick $\beta \in \mathbb{Z}_q^{\deg(R,x_1)+1}$ at random.
  for $i = 1, 2, \ldots, \deg(R, x_1) + 1$ **do**
    $\delta_i \leftarrow \mathbf{BB}\left((\beta_i, \alpha), q\right)$ // probes to the black box.
  end
  Interpolate the unique polynomial $F \in \mathbb{Z}_q[x_1]$ using $\{(\beta_i, \delta_i) : 1 \leq i \leq \deg(R, x_1) + 1\}$.
until $g_{jk} \neq 0$ for all j and k where g_{jk} is as defined in (5.1) and $\deg(F) = \deg(R, x_1)$.
Compute the monic square-free factorization $\hat{w} \prod_{j=1}^l F_j^j$ of F in $\mathbb{Z}_q[x_1]$ where $\hat{w} \in \mathbb{Z}_q$.
for $j = 1, 2, \ldots, l$ **do**
  if $F_j \neq R_j(x_1, \alpha) \in \mathbb{Z}_q[x_1]$ **then return** false **end**
end
return true

Example 5.7. Let $p = 3137$ be the input prime in our Dixon resultant algorithm and let

$$R = (y_1 + y_2)x_1^2 + y_1 + 3138y_2$$

be the Dixon resultant. The only monic square-free factor is

$$S = x_1^2 + \frac{y_1 + 3138y_2}{y_1 + y_2}.$$

At termination, our Dixon resultant algorithm will return the incorrect answer $x_1^2 + 1$ as the output for S because the rational number reconstruction process will succeed with respect to this input prime p. Using Algorithm 24, we detect that the returned answer is wrong as follows. Suppose we pick a new prime $q = 2^{31} - 1$, and evaluation points $\alpha = (2, 7) \in \mathbb{Z}_q^2$ and $\beta = (1351727965, 581869303, 1742863087) \in \mathbb{Z}_q^3$ at random. Probing the black box representing R to interpolate the monic square-free polynomial $F_1 \in \mathbb{Z}_q[x_1]$, we get

$$F_1 = x_1^2 + 238611735 \neq S(x_1, \alpha) = x_1^2 + 1 \in \mathbb{Z}_q[x_1],$$

which confirms that we have the incorrect answer from our Dixon resultant algorithm.

Note that it is possible that the R_j's are correct, but our probabilistic test fails, that is, it wrongly verifies that we have an incorrect answer. Similarly, it is possible that the R_j's are incorrect, but our probabilistic test wrongly verifies that we have a correct answer. We give the following example to illustrate this.

Example 5.8. Let p and q be primes such that $p, q \neq 2$ and let $R = (y_1+1)x_1+(pq+y_1+2)$ be the Dixon resultant. Clearly, the only monic square-free factor of R is

$$S = x_1 + \frac{(pq + y_1 + 2)}{y_1 + 1}.$$

Let p be the first prime used in our Dixon resultant algorithm and suppose that the rational number reconstruction process succeeds on the coefficient of the returned answer with respect to p. Then the output of Dixon resultant algorithm will be

$$x_1 + \frac{(y_1 + 2)}{y_1 + 1}.$$

For simplicity, let $\alpha = 0$ be the evaluation point selected at random in Line 3. If the random prime picked in Line 2 of Algorithm 24 is q then Algorithm 24 will wrongly verify that we have the correct answer. That is, in Line 12, we will have

$$F_1 = x_1 + 2 = S(x_1, \alpha)$$

because q divides an integer coefficient of S.

We will give a failure probability bound for Algorithm 24 in Theorem 6.39 when we perform the failure probability analysis of our Dixon resultant algorithm.

5.3.2 Identifying the Extraneous factors

After checking that the monic square-free factors obtained by our Dixon resultant algorithm are correct with high probability using Algorithm 24, the extraneous factors are identified using the following algorithm.

Algorithm 25: ExtraneousFactors

Input: A parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Q}[x_1, \ldots, x_n][y_1, \ldots, y_m]$ and the monic square-free factors $R_j \in \mathbb{Q}(y_1, y_2, \ldots, y_m)[x_1]$ as defined in (5.1) such that the R_j's are relatively prime and $\deg(R_j, x_1) > 0$ for $1 \leq j \leq l$.

Output: The set of factors H such that

1. for each $h \in H$, h is monic and irreducible in $\mathbb{Q}[y_1, \ldots, y_m][x_1]$,
2. for each $h \in H$, h divides one of the R_j's and
3. $\prod_{h \in H} h = g$ where g is the generator of $\left\langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \right\rangle \cap \mathbb{Q}[y_1, \ldots, y_m][x_1]$.

Remark : Conditions 2 and 3 hold with high probability.

1 Let $E_j = R_j \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1]$ for $1 \leq j \leq l$ // We clear the fractions in the R_j's.
2 **repeat**
3 Pick a random 62 bit prime p and an evaluation point $\alpha \in \mathbb{Z}_p^m$ at random.
4 Evaluate the input system $\mathcal{F}$ at $y_1 = \alpha_1, y_2 = \alpha_2, \cdots, y_m = \alpha_m$ to obtain a new polynomial system $G = \{\hat{g}_1, \hat{g}_2 \cdots, \hat{g}_n\} \subset \mathbb{Z}_p[x_1, x_2, \cdots x_n]$.
5 Compute the unique polynomial $\overline{R} \in \langle g_1, g_2, \ldots, g_n \rangle \cap \mathbb{Z}_p[x_1]$ using a Gröbner basis.
6 **until** $\mathrm{LC}(E_j, x_1) \neq 0$ for $1 \leq j \leq l$ and $\deg(g_i, x_1) = \deg(\hat{f}_i, x_1)$ for $1 \leq i \leq n$, and $\deg(\overline{R}) > 0$.
7 Factor E_j over $\mathbb{Q}$ for $1 \leq j \leq l$ and let A be the set all of irreducible polynomial factors obtained.
8 Let H be an empty set.
9 **for** $h \in A$ **do**
10 **if** $h(x_1, \alpha)$ divides $\overline{R}$ over $\mathbb{Z}_p$ **then**
11 $H \leftarrow H \cup h$ // h divides g with high probability.
12 **else**
13 // h does not divide g with high probability.
14 **end**
15 **end**
16 **return** H

Notice that Algorithm 25 continues to pick a new random prime p and a new random evaluation point $\alpha \in \mathbb{Z}_p^m$ in Lines 2-6 until $\mathrm{LC}(E, x_1) \neq 0$ where $E \in \prod_{j=1}^{l} R_j \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1]$, and until we obtain a new polynomial system $G = \{\hat{g}_1, \hat{g}_2 \cdots, \hat{g}_n\} \subset \mathbb{Z}_p[x_1, x_2, \cdots x_n]$ such that $\deg(g_j, x_1) = \deg(\hat{f}_j, x_1)$ for $1 \leq j \leq n$ when the input para-

metric polynomial system $\mathcal{F}$ is evaluated at $y_1 = \alpha_1, y_2 = \alpha_2, \cdots, y_m = \alpha_m$, and until $\deg(\overline{R}) > 0$ where $\overline{R}$ is the unique polynomial in $\langle g_1, g_2, \ldots, g_n \rangle \cap \mathbb{Z}_p[x_1]$ computed in Line 5 using a Gröbner basis. Otherwise, the evaluation point $\alpha \in \mathbb{Z}_p^m$ or the prime p picked at random in Line 3 of Algorithm 25 may cause the algorithm to wrongly identify a correct irreducible polynomial factor of R_j over $\mathbb{Q}$ as an extraneous factor, or it may cause the algorithm to wrongly identify all the irreducible factors of the R_j's over $\mathbb{Q}$ as the correct irreducible factors. We illustrate these potential problems with the following two examples.

Example 5.9. Let

$$\mathcal{F} = \{\hat{f}_1, \hat{f}_2\} = \{(y_2 - y_1)x_1^2 + 1, x_2^2 - 1\}$$

and let $\alpha = (10, 10)$ be an evaluation point. Then the new polynomial system without parameters obtained when $\mathcal{F}$ is evaluated at α over $\mathbb{Z}_{3137}$ is

$$G = \{\hat{g}_1, \hat{g}_2\} = \{1, x_2^2 + 3136\}.$$

Notice that $\deg(\hat{g}_1, x_1) < \deg(\hat{f}_1, x_1)$ and the Gröbner basis for $< \hat{g}_1, \hat{g}_2 >$ is $\{1\}$ so $\overline{R} = 1$. Thus, all the irreducible factors of the R_j's with positive degree in x_1 will be identified as an extraneous factor.

Example 5.10. Let $p \neq 2$ be a prime and let

$$\mathcal{F} = \{\hat{f}_1, \hat{f}_2\} = \{x_1^2 + ax_1 + x_2y_1 + 2, x_1^2 + x_2y_1 + 2\}$$

be a parametric polynomial system. If the random prime p chosen in Line 3 of Algorithm 25 divides the integer coefficient a of $\hat{f}_1 \in \mathcal{F}$ then the unique generator $\overline{R}$ computed in Line 5 of Algorithm 25 will be the zero polynomial. This will cause all the irreducible polynomial factors of the R_j's of $\mathcal{F}$ with positive degree in x_1 to be identified as the correct factors.

5.4 Implementation Notes and Benchmarks

We have implemented our new Dixon resultant algorithm in Maple. To improve the overall efficiency, we have implemented in C, major subroutines such as evaluating a Dixon matrix at integer points modulo prime p, computing the determinant of an integer matrix over $\mathbb{Z}_p$, solving a $t \times t$ shifted Vandermonde system and performing dense rational function interpolation using the MQRFR algorithm modulo a prime. Thus, each probe to the black box is computed using C code. Our C code supports primes up to 63 bits in length.

5.4.1 Speeding up evaluation of the Dixon matrix

In our experiments, the most expensive step in our algorithm was, and still is, evaluating the Dixon matrix M at α modulo a prime. Let p be a prime and let M be a $t \times t$ matrix of

polynomials in $\mathbb{Z}[z_1, ..., z_n]$. We need to compute $\det(M(\alpha)) \bmod p$ for many $\alpha \in \mathbb{Z}_p^n$. Often, over 80% of the time is spent computing $M(\alpha) \bmod p$. The Maple command

```
> Eval(M,{seq(z[i]=alpha[i]}) mod p;
```

does what we want, however, because we want our implementation to handle many variables and fail with low probability, we want to use the largest primes the hardware can support which are 63 bit primes if we use signed 64 bit integers. Unfortunately, `Eval` uses hardware arithmetic for $p < 2^{31}$, otherwise, it uses software arithmetic which is relatively very slow. Also, `Eval` evaluates each polynomial in M independently, that is, if $M_{1,1} = 2z_1^3 z_2$ and $M_{2,2} = z_1^3 + 5z_3$ say, `Eval` computes α_1^3 twice. To speed up evaluations, we have written a C program to compute $M(\alpha)$ for $p < 2^{63}$ using hardware arithmetic. In Maple, we first precompute a vector of degrees

$$D = \left[\max_{1 \le i,j \le t} \deg(M_{ij}, z_k) \ : \ 1 \le k \le n \right].$$

For each $\alpha \in \mathbb{Z}_p^n$ we call our C program from Maple with inputs M, α, D, p. To save multiplications, our C program first computes power arrays

$$P_k = \left[\alpha_k^i \ : \ 0 \le i \le D_k \right] \quad \text{for} \quad 1 \le k \le n$$

then uses these P_k to evaluate $M_{i,j}(\alpha)$ for $1 \le i, j \le t$. Maple uses two data structures for polynomials, the SUM-OF-PROD data structure and the POLY data structure. POLY was added to Maple in 2013 by Monagan and Pearce [Monagan and Pearce, 2015] to speed up polynomial arithmetic. Figure 1 shows the POLY data structure for the polynomial $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$. Figure 2 shows how the same polynomial is represented in the SUM-OF-PROD data structure. All boxes in Figures 1 and 2 represent arrays. The first entry in each box is a header word; it encodes the object type and the array length.

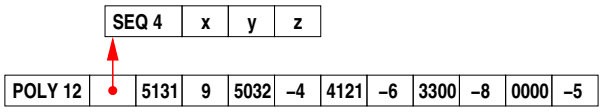

Figure 5.1: Maple's POLY representation for $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$.

In POLY, if $M = z_1^{d_1} z_2^{d_2} \cdots z_n^{d_n}$ is a monomial in f, then M is encoded as the integer $d2^{nb} + \sum_{i=0}^{n-1} 2^{ib} d_i$ where $d = \deg(M) = \sum_{i=1}^{n} d_i$ and $b = \lfloor 64/(n+1) \rfloor$. For example, the monomial xy^3z with $d = 5, b = 16, n = 3$ is encoded as the integer $5 \cdot 2^{48} + 2^{32} + 3 \cdot 2^{16} + 1$. This is depicted as 5131 in Figure 1. This encoding allows Maple to compare two monomials in the graded monomial ordering using a single 64 bit integer comparison. Also, provided overflow does not occur, Maple can multiply two monomials using a single 64 bit integer addition.

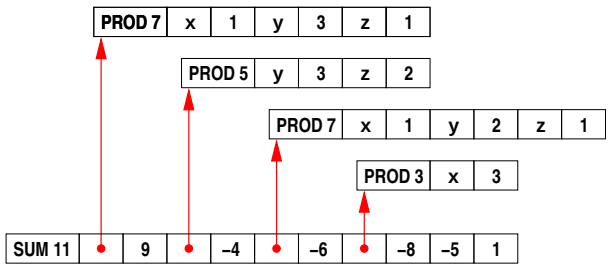

Figure 5.2: Maple's SUM-OF-PROD representation for $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$.

When does Maple use POLY instead of SUM-OF-PROD? If a polynomial f has (i) all integer coefficients, (ii) more than one term, (iii) is not linear, and (iv) all monomials in f can be encoded in a 64 bit integer using B bits for d_i and $64 - nB$ bits for d, then it is encoded using POLY otherwise the SUM-OF-PROD representation is used. In a typical Dixon matrix both representations are used so we have to handle both and we need to know the details of both representations.

Also important for efficiency is how to multiply in $\mathbb{Z}_p$. We do not use the hardware division instruction which is very slow. Instead, we use Roman Pearce's assembler implementation of Möller and Granlund [Möller and Granlund, 2010] which replaces division with two multiplications and other cheap operations.

Table 5.1: Timings showing improvements for Heron5d and Tot systems

System	**`Eval`**	Determinant	Total	C-Eval	New Total
Heron5d	70.17s (66.2%)	9.74s (9.18%)	106.07s	18.02s (3.89x)	42.82s (2.48x)
Tot	635.75s (83.3%)	37.66s (4.9%)	763.2s	32.36s (19.64x)	150s (5.08x)

This experiment was performed on an Intel Xeon E5-2680 v2 processor using 1 core (cecm `jude` server).

Table 5.1 shows the improvement obtained using our C code for evaluating a Dixon matrix M at integer points modulo a prime for the Tot and Heron5d systems. Column `Eval` contains the timings obtained using Maple's `Eval` command, and column C-Eval represents the timings obtained when our C code was used. Column Determinant indicates the amount of time spent computing the determinant of integer matrices modulo a prime. Column Total contains the total CPU timings using `Eval`, and column New Total is the new total CPU timings for both polynomial systems when our C code for performing matrix evaluations was used.

5.4.2 Pre-computing $\deg(f_{i,k})$ and $\deg(g_{i,k})$

We did not pre-compute the total degrees $\deg(f_{i,k})$ and $\deg(g_{i,k})$ of the lower degree homogeneous polynomials $f_{i,k}, g_{i,k}$ in our old benchmarks when our CASC paper [Jinadu and

Monagan, 2022a,b] was accepted for publication. Since then, we have re-designed our Dixon resultant algorithm to pre-compute these degrees. The timings reported in Table 5.2 show the improvement when these total degrees were not precomputed (row Before), and the new timings (row After) obtained when the degrees $\deg(f_{i,k})$ and $\deg(g_{i,k)}$ were precomputed in our Dixon resultant algorithm.

Table 5.2: Timings showing improvements when $\deg(f_{i,k})$ and $\deg(g_{i,k)}$ are pre-computed

	Robot-t_2	Robot-b_1	Robot-b_2	Tot	Flex-v1	Flex-v2	Pose	Perimeter
Before	316.99s	27.78s	241.61s	82.11s	201s	461.4s	461.4s	49.97s
After	222.60s	18.33s	171.97s	49.04s	100.99s	154.20s	243.88s	18.99s

This experiment was performed on an Intel Xeon E5-2680 v2 processor using 1 core (cecm `jude` server).

5.4.3 Timings

We present two tables (Tables 5.3 and 5.4) for our Dixon resultant algorithm. Table 5.3 contains basic information about our benchmark polynomial systems. This includes the names of the real parametric polynomial systems (Column **System**) on which we tested our code, the number of equations in each parametric polynomial system (Column #**Eq**), the number of variables n and the number of parameters m (Column n/m), the dimension of the Dixon matrix D obtained, and the rank of a sub-matrix M of D such that $\text{rank}(M) = \text{rank}(D)$ (Column $\dim(D)/$**Rank**).

The timings for comparing our new Dixon resultant algorithm with three other methods for computing R are also reported in Table 5.3. In particular, we report the timings for our new Dixon resultant algorithm in Column **DixonRes**, the timings of an efficient Maple implementation of the Gentleman & Johnson minor expansion method in Column **Minor**, the timings of a hybrid implementation of Zippel's sparse interpolation algorithm in Maple + C in Column **Zippel** and the timings of a Maple implementation of the Dixon-EDF algorithm in Column **EDF**. The implementation of the Dixon-EDF algorithm involves sorting the matrix M by placing the sparsest columns at the left of the matrix, removing the gcd of each row before starting the elimination and it has a pivot selection algorithm. We remind the reader that the Dixon-EDF algorithm was introduced by Lewis in [Lewis, 2017].

To make the comparison between our Dixon resultant algorithm and Zippel's sparse interpolation algorithm for computing R fair, we have implemented the most expensive part of Zippel's algorithm, which is the routine that solves for the coefficients of the Dixon resultant R in C (Subroutine VandermonderSolver). We use our C code for evaluating matrix of polynomials at α modulo a prime p.

Table 5.3 also contains the number of terms in the product of all the monic square-free factors in expanded form after clearing the denominators (Column $\#S$). Additionally, it includes the number of terms in the Dixon resultant R in expanded form (Column $\#R$). In

Column 6 labelled $t_{\max} = \max(\#f_{jk}, \#g_{jk})$, we report the number of terms present in the largest polynomial coefficient of an R_j to be interpolated by our Dixon resultant algorithm. The number of monic square-free factors with respect to each Dixon resultant R is reported in Column # **of R_j's.** All our experiments were performed on a 24 core Intel Gold 6342 processor with 256 gigabytes of RAM using only 1 core (cecm `maple` server) running at 2.8GHz (base) and 3.5GHz (turbo) and the first smooth prime used in our code is the 62 bit prime $p = (2^{50})(61)(67) + 1$.

As the reader can see in Columns $8, 10, 11$ and 12, our new Dixon resultant (DixonRes) algorithm outperforms Zippel's sparse interpolation and the Gentleman & Johnson algorithm on most of our benchmark systems. This was expected because $\#R$ is much larger than $t_{\max}$. Another reason why this is the case is because more primes are needed to recover integer coefficients in R compared to the R_j's. Our algorithm is not always faster than the Dixon-EDF algorithm. The evaluation cost of the Dixon matrix is still the bottleneck of our algorithm while the determinant computation takes typically 10% of the total time.

5.4.4 Optimization

In our experiments, we found out some Dixon matrices have a block diagonal form and often, the determinant of all the blocks produce the same Dixon resultant R. For the timings recorded in Tables 5.3 and 5.4, we always compute the determinant of the smallest block after confirming that all the blocks produce the same Dixon resultant. We give the following example to illustrate how we do this.

Example 5.11 (Continuation of Example 2.15)**.** The Dixon matrix D from Example 2.15 is rectangular so extracting its sub-matrix of maximal rank yields a M, a matrix of rank $13 = \text{rank}(D)$. The matrix M is diagonally decomposable , so we obtain the block decomposition form of M using Maple's command `StronglyConnectedBlocks` which yields the form

$$M = \left[\begin{array}{c|c} B_1 & \mathbf{0} \\ \hline \mathbf{0} & B_2 \end{array}\right]$$

such that

$$B_1 = \begin{pmatrix} -A & -B & 0 & 0 & -C & 0 \\ 0 & -A & -B & K & 0 & 0 \\ 0 & 0 & -A & -C & 0 & 0 \\ B & E & 0 & 0 & F & G \\ 0 & 0 & B & F & A & 0 \\ 0 & 0 & 0 & G & 0 & A \end{pmatrix}$$

and

$$B_2 = \begin{pmatrix} -A & -C & 0 & 0 & 0 & 0 & 0 \\ 0 & E & B & -B & F & G & 0 \\ 0 & -B & -A & A & -C & 0 & 0 \\ 0 & 0 & A & 0 & C & 0 & 0 \\ B & F & C & 0 & K & 0 & 0 \\ 0 & G & 0 & 0 & 0 & K & C \\ 0 & 0 & 0 & 0 & 0 & C & A \end{pmatrix}$$

where $A, B, C, E, F, G, K \in \mathbb{Q}[y_1, y_2, y_3 \cdots, y_6][x_1]$.

Our idea to check if one block B_i is enough to compute the Dixon resultant R is as follows. We choose a prime p, say $p = 3137$ and random points

$$\{y_1 = 2372, y_2 = 2491, y_3 = 2088, y_4 = 1484, y_5 = 107, y_6 = 2780\}$$

from $\mathbb{Z}_p^6$. Then we interpolate

$$\det(B_1, x_1) = 936x_1^2 + 1801 \text{ and } \det(B_2, x_1) = 2827x_1^2 + 2507.$$

Making both $\det(B_1, x_1)$ and $\det(B_2, x_1)$ monic in $\mathbb{Z}_p[x_1]$ yields

$$x_1^2 + 508,$$

which means that one block is sufficient to compute R with high probability.

Thus, the number of terms in S and R recorded in Tables 5.3 and 5.4 are obtained using the smallest block obtained from the block decomposition of a sub-matrix of maximal rank. Details about the block structure of all the Dixon matrices for our benchmark systems are provided in Table 5.4. The details include block sizes of each Dixon matrix M and the number of black box probes required by our Dixon resultant algorithm to successfully interpolate the R_j's.

The number of black box probes done to obtain all degree bounds needed, and the number of black box probes needed to get the first image of the R_j's by our Dixon resultant algorithm are reported in Columns **degree** and **image-1** respectively. If the rational number reconstruction process fails on the first image, then more primes are needed. The number of black box probes used for each subsequent prime is reported in Column **image**-2. The number of primes used to interpolate the monic square-free factors is labelled as $\#p_i$.

As the reader can see in column $\#p_i$, one 62 bit prime is typically enough to recover the R_j's. In Table 5.4, the number of black box probes used by Zippel's algorithm to interpolate R is denoted by Z-probes.

Table 5.3: Systems Information for our Dixon matrices and timings for DixonRes versus Minor Expansion, Dixon-EDF and Zippel's Interpolation

System	**#Eq**	n/m	$\dim D$/**Rank**	$\#S$	$t_{\max}$	$\#R$	**# of R_j's**	**DixonRes**	**Minor**	**Zippel**	**EDF**
Robot-x_1	4	4/7	$(32 \times 32)/16$	450	14	6924715	**3**	3.53s	1442.45s	$> 10^5$s	962.79s
Robot-x_2	4	4/7	$(32 \times 48)/12$	13016	691	16963876	**3**	130.90s	!	$> 10^5$s	$> 10^5$s
Robot-x_3	4	4/7	$(32 \times 32)/16$	334	85	6385205	**2**	10.77s	168.88s	$> 10^5$s	25.60s
Robot-x_4	4	4/7	$(32 \times 48)/12$	11737	624	16801877	**3**	101.28s	!	$> 10^5$s	$> 10^5$s
Tot	4	4/5	$(85 \times 94)/56$	8930	348	52982	**2**	26.02s	!	284.83s	5146.35s
Storti	6	5/2	$(24 \times 113)/20$	12	4	32	**2**	0.074s	75.24s	0.013s	0.098s
Allie-2	3	2/2	$(13 \times 13)/13$	40	3	204	**2**	0.073s	1.06s	0.028s	0.089s
Allie-3	4	3/2	$(63 \times 63)/55$	222	7	4923	**4**	3.21s	$> 10^4$s	12.06s	36.06s
Allie-4	5	4/2	$(313 \times 313)/237$	614	8	-	**9**	367. 80s	NA	NA	$> 10^5$s
Allie-5	6	5/2	$(1563 \times 1563)/967$	2100	10	-	**12**	46914.83s	NA	NA	NA
Laconelli	5	5/6	$(28 \times 21)/11$	20	5	912	**2**	0.058s	8.02s	0.285s	0.072s
Auto	5	5/3	$(32 \times 18)/18$	23	6	666	**3**	0.063s	15.99s	0.578s	0.135s
Circle	9	8/5	$(88 \times 58)/43$	180085	3731	12296628	**6**	3359.49s	$>10^5$s	10^5s	23724.30s
Hairer	11	11/2	$(96 \times 85)/40$	398	17	1592	**3**	0.195s	$> 10^4$s	2.10s	2.57s
Pizza-Roll	7	6/2	$(288 \times 1008)/264$	1655	33	7322	**3**	294.96s	NA	547.68s	3662.40s
Toothy	7	6/2	$(798 \times 2092)/544$	1694	48	10462	**5**	4086.33s	$> 10^4$s	4851.60s	$> 10^5$s
Heron4d	10	10/10	$(103 \times 75)/63$	131	130	1471	**1**	0.548s	2.67s	7.90s	0.087s
Heron5d	15	14/16	$(414 \times 707)/313$	823	822	460599	**1**	4.49s	!	$> 10^5$s	0.431s
Heron6d	21	21/21	$(4981 \times 2573)/1797$	6203	6202	-	**1**	99.29s	NA	NA	32.97s
Heron7d	28	28/28	$(35461 \times 16306)/10343$	52553	52552	-	**1**	6715.20s	!	NA	!
Pendulum	3	2/3	$(40 \times 40)/33$	4667	243	19899	**3**	22.55s	1195.25s	36.84s	2.74s
Flex-v1	3	3/15	$(8 \times 8)/8$	5685	2481	45773	**2**	56.85s	2.28s	1784.27s	0.751s
Flex-v2	3	3/15	$(8 \times 8)/8$	12101	2517	45773	**2**	85.08s	2.29s	3156.24s	0.753s
Perimeter	6	6/4	$(16 \times 16)/16$	1980	303	9698	**1**	10.50s	8.23s	35.53s	0.1s
Lee	4	3/3	$(28 \times 28)/22$	2925	325	2925	**1**	17.14s	$> 10^4$s	6.28s	1.69s
Hawes1	4	3/2	$(58 \times 54)/46$	78	3	230	**2**	1.14s	$> 10^3$s	0.591s	1.91s
Bisector	3	3/3	$(12 \times 11)/11$	136	31	136	**1**	1.29s	0.132s	0.066s	0.187s
Sift-Ex	4	4/11	$(8 \times 9)/8$	16614	1362	223806	**3**	37.68s	60.32s	$> 10^4$s	12.40s
3dconic	4	2/13	$(4 \times 4)/4$	4474	243	17430	**2**	2.35s	0.074s	115.56s	0.020s
Morley	4	4/4	$(35 \times 35)/35$	179	23	179	**1**	2.00s	$> 10^5$s	1.31s	0.370s
Geddes2	4	3/2	$(36 \times 34)/24$	1425	27	1533	**3**	6.18s	$> 10^5$s	0.743s	1.90s
Geddes3	4	4/8	$(26 \times 26)/22$	2415	302	4501	**2**	5.54s	0.009s	16.44s	0.042s

! = ran out of memory, NA = Not Attempted

Table 5.4: Block structure and # of probes used by Algorithm DixonRes and Zippel's interpolation

System	**Block Structure**	**degree**	**image**-1	**image**-2	$\#p_i$	Z-probes
Robot-x_1	$[8, 8]$	4096	13000	-	1	DNF
Robot-x_2	$[12]$	6224	705796	-	1	DNF
Robot-x_3	$[8, 8]$	4356	91000	-	1	DNF
Robot-x_4	$[12]$	6028	529984	-	1	DNF
Heron4d	[18, 17, 14, 14]	244	9360	-	1	104465
Heron5d	[35, 34, 47, 44, 33, 41, 36, 43]	346	62928	-	1	DNF
Heron6d	"x"	322	294360	-	1	DNF
Pendulum	$[17, 16]$	13261	114920	53040	2	128229
Tot	$[31, 25]$	5071	264000	-	1	742099
Flex-v1	$[8]$	2044	637632	-	1	2005023
Flex-v2	$[8]$	5116	2664948	-	1	3310871
Perimeter	$[16]$	1342	225828	-	1	221075
Storti	$[20]$	426	816	-	1	279
Allie-2	$[13]$	476	900	-	1	851
Allie-3	$[55]$	4441	8658	-	1	38723
Allie-4	$[237]$	20563	40230	-	1	DNF
Manocha	$[5]$	586	209700	-	1	111716
Laconelli	$[11]$	197	280	-	1	5745
Lee	$[22]$	2626	37700	16250	11	52378
Hawes1	$[46]$	2497	4896	816	2	4160
Auto	$[18]$	477	952	-	1	13280
Hermert	$[14]$	449	1280	-	1	141123
Vanabuel	$[7, 7, 7, 7]$	900	1740	-	1	1226665
Storti2	$[272]$	126872	253164	42194	11	261463
Circle	$[43]$	58411	4583592	1762920	2	DNF
Conic	$[5]$	586	209700	-	1	111716
Ellipse	$[272, 272]$	126433	252288	42048	6	308698
Alie-5	$[967]$	84477	165504	27584	2	DNF

"x" $= [100, 98, 93, 141, 133, 89, 124, 94, 131, 129, 107, 124, 93, 100, 123, 118]$ and DNF= Did Not Finish

Table 5.3 Continued: Systems Information for our Dixon matrices and timings for DixonRes versus Minor Expansion, Dixon-EDF and Zippel's Interpolation

System	**#Eq**	n/m	$\dim D$/**Rank**	$\#S$	$t_{\max}$	$\#R$	# of R_j's	**DixonRes**	**Minor**	**Zippel**	**EDF**
Hawes4	6	6/3	$(117 \times 154)/60$	26894	1974	37301	**2**	254.58s	$> 10^4$s	399.72s	1533.60
Datum	7	6/19	$(4 \times 4)/4$	345001	203924	60254646	**1**	21157.34s	1052.48s	$> 10^5$s	$> 10^5$s
Ellipse	7	6/2	$(800 \times 2184)/544$	1350	35	5265	**2**	7144.59s	$> 10^5$s	3771.81s	19944.01s
Image3d	10	10/9	$(178 \times 152)/130$	130	84	1456	**1**	0.535s	0.491s	1.33s	0.102s
Topo	5	5/6	$(6 \times 6)/6$	66	21	150	**1**	0.33s	0.03	0.13s	0.014s
Enneper	3	3/2	$(11 \times 11)/9$	23	11	257	**1**	0.089s	0.025s	0.007s	0.013s
Cyclo	3	3/3	$(8 \times 8)/8$	313	44	698	**2**	0.705s	0.054s	0.104s	0.025s
Basepoint	3	3/2	$(12 \times 12)/5$	51	6	51	**1**	0.097s	0.002s	0.006s	0.007s
Wolfie	4	4/8	$(13 \times 13)/12$	24068	5482	24068	**1**	139.39s	2.10s	244.80s	41.02s
Nachtwey	6	6/5	$(11 \times 18)/11$	244	106	244	**1**	2.14s	0.292s	0.531s	0.087s
Pavelle	4	4/19	$(5 \times 5)/14$	89	35	89	**2**	0.492s	0.002s	0.041s	0.027s
Wein1	3	3/18	$(5 \times 5)/5$	21894	10603	80538	**1**	183.22s	0.322s	3792.60s	1.99s
Wein2	3	3/18	$(5 \times 5)/5$	80538	10603	80538	**1**	1135.68s	0.307s	3851.40s	1.99s
Heron3d	6	6/6	$(16 \times 14)/13$	23	22	23	**1**	0.099s	0.004s	0.036s	0.044s
Hawes2	4	4/5	$(9 \times 8)/8$	671	80	671	**1**	0.910s	0.018s	0.294s	0.072s
Hermert	14	14/12	$(20 \times 31)/14$	112	8	5976	**2**	0.114s	1.62s	19.25s	0.088s
Vanaubel	9	9/5	$(28 \times 28)/28$	11	4	32166	**1**	0.187s	0.816s	739.20s	0.013s
Storti2	5	4/2	$(400 \times 400)/272$	1350	35	5265	**2**	2658.42s	$> 10^5$s	937.83s	42984.92s
Conic	3	3/12	$(5 \times 5)/4$	2424	912	6548	**1**	9.44s	0.005s	14.74s	0.114s
Heron2d	3	3/3	$(3 \times 3)/3$	7	6	7	**1**	0.043s	0s	0.005s	0.26s
Bricard	6	6/11	$(41 \times 44)/29$	312783	38986	1111775	**2**	5491.80s	226.20s	$> 10^4$s	624.60s
Geddes4	4	4/8	$(26 \times 26)/22$	57252	5540	87244	**3**	457.74s	$> 10^5$s	$> 10^4$s	416.40s

! = ran out of memory, NA= Not Attempted

Table 5.4 Continued: Block structure and # of probes used by Algorithm DixonRes and Zippel's interpolation

System	**Block Structure**	**degree**	**image**-1	**image**-2	$\#p_i$	Z-probes
Heron2d	[3]	94	288	-	1	119
Nachtwey	[11]	611	39780	18020	2	12739
Cyclo	[8]	1684	15708	6930	2	3293
Basepoint	[5]	161	480	-	1	231
Wolfie	[12]	2772	2322540	-	1	550149
Pavelle	[5]	901	12768	-	1	1731
Wein1	[5]	786	2558160	-	1	2718978
Wein2	[5]	1051	11906676	-	1	2740218
Datum	[4]	652	29731968	-	1	DNF
Heron3d	[6, 7]	160	1392	-	1	757
Hawes2	[8]	1626	5100	-	1	10887
Hawes4	[60]	9226	295000	118000	4	473470
Image3d	[13, 14, 14, 15, 18, 19, 18, 19]	436	12320	-	1	27959
Topo	[6]	610	7392	-	1	4547
Enneper	[9]	295	616	-	1	257
Heron7d	"y"	412	3233868	-	1	-
Bisector	[11]	1420	36784	-	1	2255
Sift-Ex	[8]	1525	819984	-	1	DNF
Toothy	[272, 272]	91451	182280	30380	4	412206
3dconic	[4]	1418	48672	-	1	DNF
Morley	[35]	3837	20608	9016	2	16351
Geddes2	[24]	13601	27030	4505	2	7018
Geddes3	[5]	1000	146124	-	1	149543
Geddes4	[22]	8971	2361600	1017600	2	DNF
Hairer	[40]	552	1520	-	1	22107
Pizza-Roll	[132, 132]	30099	59898	9983	4	234839
Bricard	[17, 12]	3091	16046640	-	1	DNF

DNF= Did Not Finish

"y" = [343, 338, 375, 319, 299, 305, 324, 303, 320, 329, 353, 318, 329, 353, 374, 343, 318, 342, 324, 299, 303, 342, 320, 305, 287, 302, 315, 338, 319, 315, 287, 302]

Chapter 6

Failure Probability and Complexity Analysis

6.1 Summary of Contributions

In this chapter, we present the failure probability analysis and the complexity analysis of our new Dixon resultant algorithm. Our main contribution includes the classification of several causes of failure in our Dixon resultant algorithm with new failure probability bounds. New bounds for the Dixon resultant R and bounds for its monic square-free factors R_j which are potentially useful in other applications are obtained. Constructed examples are also given to illustrate possible failure scenarios. A journal paper containing the results of this chapter is currently being prepared. All the results in this chapter are new unless explicitly cited.

6.2 Introduction

6.2.1 Two Useful Results

Definition 6.1. Let $f = \sum_{i=1}^{t} a_i N_i \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$ where $a_i \neq 0, t = \#f \geq 1$ and N_i is a monomial in variables $y_1, y_2, \ldots, y_m$. The height of f denoted by $\|f\|_\infty$ is defined as $\|f\|_\infty = \max_{i=1}^{t} |a_i|$. We also define $\|H\|_\infty = \max_{k=0}^{d_T} (\|f_k\|_\infty, \|g_k\|_\infty)$ where $H = \sum_{k=0}^{d_T} \frac{f_k}{g_k} x_1^k$ and $f_k, g_k \in \mathbb{Z}[y_1, y_2, \ldots, y_n]$ with $\gcd(f_k, g_k) = 1$.

We state the following two important results for later use.

Theorem 6.2. [Hu and Monagan, 2021, Proposition 2] Let A be a $s \times s$ matrix with $A_{ij} \in \mathbb{Z}[y_1, \ldots, y_m]$, satisfying the term bound $\#A_{ij} \leq t$ and the coefficient bound $\|A_{ij}\|_\infty \leq h$. Then

$$\|\det(A)\|_\infty < s^{\frac{s}{2}} t^s h^s.$$

Lemma 6.3. [Gelfond, 2015, Lemma 2, page 135] Let $f, g \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$. If $g | f$ then

$$\|g\|_\infty \leq \mathrm{e}^{\sum_{i=1}^{m} \deg(f, y_i)} \|f\|_\infty$$

where e ≈ 2.718 is the Euler number.

6.2.2 Important Notations and Bounds

The failure probability analysis of our Dixon resultant algorithm will depend on the input parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \cdots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$. Thus, our first goal is to obtain height and degree bounds for the Dixon resultant R and its monic square-free factor R_j. For simplicity, the following notation (Notations 6.4) will remain unchanged and will be used consistently throughout this chapter.

Notations 6.4. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \cdots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$ be a parametric polynomial system. For convenience, let the Dixon resultant

$$R = \sum_{k=0}^{\hat{d}} \bar{r}_k(y_1, \ldots, y_m) x_1^k \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1]$$

and let the monic square-free factors

$$R_j = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \frac{f_{jk}(y_1, y_2, \ldots, y_m)}{g_{jk}(y_1, y_2, \ldots, y_m)} x_1^{d_{j_k}} \in \mathbb{Z}(y_1, y_2, \ldots, y_m)[x_1]$$

for $f_{jk}, g_{jk} \neq 0$ in $\mathbb{Z}[y_1, y_2, \ldots, y_m]$ where $\gcd(f_{jk}, g_{jk}) = 1$ and $\hat{d} = \deg(R, x_1) > 0$. Let

$$S = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)} x_1^{d_k} \in \mathbb{Z}(y_1, y_2, \cdots, y_m)[x_1]$$

be the one monic square-free factor to be interpolated by our Dixon resultant algorithm (Algorithm 19) such that $\gcd(f_k, g_k) = 1$, so $S := R_1$. Furthermore, let

1. $H = \max_{\hat{f} \in \mathcal{F}} \|\hat{f}\|_\infty$,
2. $N_{\max} = \max_{\hat{f} \in \mathcal{F}} \#\hat{f}$,
3. $d_{\max,i} = \max_{\hat{f} \in \mathcal{F}} \deg(\hat{f}, x_i)$ for $1 \leq i \leq n$,
4. $D_{\max,k} = \max_{\hat{f} \in \mathcal{F}} \deg(\hat{f}, y_k)$ for $1 \leq k \leq m$.
5. $d_{\max} = \max_{i=1}^{n} d_{\max,i}$,
6. $D_{\max} = \max_{i=1}^{m} D_{\max,i}$,
7. $D_{R_y} = \sum_{k=1}^{m} \deg(R, y_k)$,
8. M be a $s \times s$ Dixon matrix where $s = (n-1)! \prod_{i=2}^{n} d_{\max,i}$ and
9. $T_{\max} = \max_{i,j=1}^{s} \#M_{ij}$.

Theorem 6.5. We have

(i) $\deg(R, x_1) \leq nsd_{\max,1}$.

(ii) $\deg(R, y_k) \leq nsD_{\max,k}$ for $1 \leq k \leq m$.

(iii) $\|\Delta_{X_e}\|_\infty \leq n^{\frac{n}{2}} H^n N_{\max}^n$ where Δ_{X_e} is the Dixon polynomial (Definition 2.2).

(iv) $\|R\|_\infty \leq T_{\max}^s n^{\frac{n}{2}} (HN_{\max})^{ns} s^{\frac{s}{2}}$.

(v) $\|R_j\|_\infty \leq \mathrm{e}^{nsd_{\max,1}+2D_{R_y}} \|R\|_\infty$ where $\mathrm{e} \approx 2.718$ is the Euler number.

Proof. For claim (i), we have

$$\deg(R, x_1) \leq s \times \max_{1 \leq i,j \leq s} \{\deg(M_{ij}, x_1)\} \leq s \times \deg(\Delta_{X_e}, x_1) \leq nsd_{\max,1}.$$

Similarly, for claim (ii), we have

$$\deg(R, y_k) \leq \ s \times \max_{1 \leq i,j \leq s} \{\deg(M_{ij}, y_k)\} \leq s \times \deg(\Delta_{X_e}, y_k) \leq nsD_{\max,k}.$$

We prove clam (iii) as follows. Recall Definition 2.2, which says that the Dixon polynomial

$$\Delta_{X_e} = \frac{\det(\mathcal{C})}{\prod_{j=2}^n (x_j - \bar{x}_n)}$$

where $\mathcal{C}$ is the old cancellation matrix. Let

$$B = \sum_{j=1}^n \deg(\det(\mathcal{C}), x_i) + \sum_{j=2}^n \deg(\det(\mathcal{C}), \bar{x}_j) + \sum_{k=1}^m \deg(\det(\mathcal{C}), y_k).$$

Since $d_{\max,1} \leq d_{\max}$, it follows that

$$B \leq nd_{\max,1} + d_{\max} \sum_{i=2}^n (i-1) + d_{\max} \sum_{i=2}^n (n-i+1) + n \sum_{k=1}^m D_{\max,k} \leq n^2 d_{\max} + n \sum_{k=1}^m D_{\max,k}.$$

Now let $P = \prod_{i=2}^n (x_i - \bar{x}_i)$. Since $\|P\|_\infty = 1$ and $P \,|\, \det(\mathcal{C})$ in (2.3), using Lemma 6.3, we have that one possible bound for $\|\Delta_{X_e}\|_\infty$, namely

$$\|\Delta_{X_e}\|_\infty \ \leq \ \mathrm{e}^B \ n^{\frac{n}{2}} H^n N_{\max}^n \leq \mathrm{e}^{n^2 d_{\max} + n \sum_{k=1}^m D_{\max,k}} \, n^{\frac{n}{2}} H^n N_{\max}^n. \tag{6.1}$$

However, the above bound is too large. Instead, we obtain a tighter bound for $\|\Delta_{X_e}\|_\infty$ using the formula we derived for creating the new cancellation matrix $\hat{\mathcal{C}}$ in (2.6), which was used in our algorithm for constructing a Dixon matrix in order to avoid expression swell. Recall

$$\hat{\mathcal{C}}_{j,k} = \sum_{i=0}^{d_{\max,j}-1} \left(\sum_{u=i}^{d_{\max,j}-1} \tilde{f}_{u+1,j,k} \ x_j^{u-i} \right) \bar{x}_j^{\,i}$$

where

$$\tilde{f}_{u,j,k} \in \mathbb{Z}[x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_{j+1}, \ldots, x_n]$$

for $u \neq 0$ and

$$\tilde{f}_{0,j,k} \in \mathbb{Z}[x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n].$$

Since $\tilde{f}_{u,j,k}$ does not contain variables x_j and $\bar{x}_j$ for $u \neq 0$, we get

$$\|\hat{\mathcal{C}}_{j,k}\|_\infty \leq \|\tilde{f}_{u,j,k}\|_\infty \leq \|\hat{f}_k\|_\infty \leq H. \tag{6.2}$$

Now, using Theorem 6.2, we have

$$\|\Delta_{X_e}\|_\infty \leq \|\det(\hat{\mathcal{C}})\|_\infty \leq n^{\frac{n}{2}} \|\hat{\mathcal{C}}_{ij}\|_\infty^n N_{\max}^n \leq n^{\frac{n}{2}} H^n N_{\max}^n, \tag{6.3}$$

which is tighter than the bound obtained for $\|\Delta_{X_e}\|_\infty$ in (6.1). Therefore,

$$\|R\|_\infty \ \leq \left(T_{\max} \|M_{ij}\|_\infty \sqrt{s}\right)^s \leq T_{\max}^s n^{\frac{n}{2}} (H N_{\max})^{ns} s^{\frac{s}{2}},$$

by Theorem 6.2, since $R = \det(M)$ and

$$\|M_{ij}\|_\infty \ \leq \|\Delta_{X_e}\|_\infty.$$

This proves claim (iv). Finally, we prove claim (v). Let

$$L = \text{LCM}\{g_{jk} \in \mathbb{Z}[x_1, y_1, \ldots, y_m] : 0 \leq k \leq T_j - 1\}$$

(clearing fractions of R_j in $y_1, y_2, \ldots, y_m$) and let

$$H_j = LR_j \in \mathbb{Z}[x_1, y_1, \ldots, y_m]$$

where $R_j = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \frac{f_{jk}(y_1, y_2, \ldots, y_m)}{g_{jk}(y_1, y_2, \ldots, y_m)} x_1^{d_{j_k}}$. Since $H_j | R$, by Lemma 6.3, we have that

$$\|H_j\|_\infty \ \leq \mathbf{e}^{\deg(R, x_1) + D_{R_y}} \|R\|_\infty \ \leq \mathbf{e}^{nsd_{\max,1} + D_{R_y}} \|R\|_\infty.$$

Let $H_j = \sum_{k=0}^{T_j} a_k(y_1, \ldots, y_m) x_1^{d_k}$. Thus, we have

$$\frac{H_j}{L} = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \frac{a_k(y_1, \ldots, y_m)}{L} x_1^{d_k}.$$

Let $h_k = \gcd(a_k, L)$. Observe that

$$\frac{a_k / h_k}{L / h_k} = \frac{f_{jk}}{g_{jk}}.$$

So, $f_{jk}|a_k$ and $g_{jk}|L$, which implies that $g_{jk}|\mathrm{LC}(R)$. Using Lemma 6.3, we get

$$\|f_{jk}\|_\infty \ \leq \mathbf{e}^{D_{R_y}}\|a_k\|_\infty \ \leq \mathbf{e}^{D_{R_y}}\|H_j\|_\infty \ \leq \mathbf{e}^{nsd_{\max,1}+2D_{R_y}}\|R\|_\infty$$

and

$$\|g_{j,k}\|_\infty \ \leq \mathbf{e}^{D_{R_y}}\|\mathrm{LC}(R)\|_\infty \ \leq \mathbf{e}^{D_{R_y}}\|R\|_\infty.$$

Therefore,

$$\|R_j\|_\infty \leq \max_{k=0}^{T_j-1}(\max(\,\|f_{jk}\|_\infty, \|g_{jk}\|_\infty)) \leq \mathbf{e}^{nsd_{\max,1}+2D_{R_y}}\|R\|_\infty.$$

□

Note that the height bound $\|R_j\|_\infty$ obtained in Theorem 6.5(v) is a worst case because $\|R_j\|_\infty$ is always smaller than $\|R\|_\infty$ in our experiments. It is very rare for factors of R to have larger coefficients than R.

Remark 6.6. Recall that in (2.8), we first derived

$$\hat{\mathcal{C}}_{j,k} = \sum_{u=1}^{d_{\max,j}} \tilde{f}_{u,j,k} \underbrace{\left(\sum_{i=0}^{u-1} x_j^i \ \bar{x}_j^{\,u-i-1}\right)}_{W_u}.$$

Clearly

$$\|W_u\|_\infty = 1 \text{ and } \#W_u \leq u \leq d_{\max,j} \leq d_{\max},$$

so

$$\|W_u\tilde{f}_{u,j,k}\|_\infty \leq \min(\#W_u, \#\tilde{f}_{u,j,k})\|W_u\|_\infty\|\tilde{f}_{u,j,k}\|_\infty \leq d_{\max}\|\tilde{f}_{u,j,k}\|_\infty \leq Hd_{max}.$$

Hence, we have

$$\|\hat{\mathcal{C}}_{j,k}\|_\infty \leq \|\tilde{f}_{u,j,k}\|_\infty \sum_{u=1}^{d_{\max,j}} u \leq Hd_{\max}^2 \tag{6.4}$$

which is still large when compared to (6.2). Thus, (2.8) had to be simplified further in order to get a better bound. Using (6.4) would mean that

$$\|R\|_\infty \ \leq \left(s^{\frac{1}{2n}} T_{\max}^{\frac{1}{n}} n^{\frac{1}{2}} H d_{\max}^2 N_{\max}\right)^{ns}$$

which is worse than the bound obtained for $\|R\|_\infty$ in Theorem 6.5(iv) by a factor of $d_{\max}^{2ns}$.

Theorem 6.7. We have

$$D_{R_y} \leq nmsD_{\max}.$$

Proof. This follows from Theorem 6.5(ii) since

$$D_{R_y} = \sum_{k=1}^{m} \deg(R, y_k)$$

and $D_{\max} = \max_{i=1}^{m} D_{\max,i}$.

□

6.3 Problems

Having obtained all the needed degree and height bounds, we proceed with the classification of several causes of failure in our Dixon resultant algorithm.

6.3.1 Evaluation Points

The first step in our Dixon resultant algorithm is to interpolate many monic univariate polynomial images of the Dixon resultant R in x_1, and then we compute their monic square-free factorization using Subroutine PolyInterp (Subroutine 16). To ensure that our monic square-free factors are consistent from one image to the next with high probability, it is important that we avoid using some evaluation points.

Definition 6.8. Let p be a prime that does not divide any integer coefficient of R and R_j. Let $\alpha \in \mathbb{Z}_p^m$ be an evaluation point. We say that $\alpha \in \mathbb{Z}_p^m$ is **bad** if $\mathrm{LC}(R)(\alpha) = 0$. We also refer to $\alpha \in \mathbb{Z}_p^m$ as an evaluation point that **causes missing terms** if any numerator coefficient of an R_j vanishes. That is $f_{jk}(\alpha) = 0$ and $g_{jk}(\alpha) \neq 0$ for some j and k.

Example 6.9. Let the Dixon resultant

$$R = (y_1 - a)x_1^2 + y_2(y_1 - b)x_1 + (c - y_2).$$

Since R has only one monic square-free factor, we have

$$S := R_1 = x_1^2 + \frac{y_2(y_1 - b)}{y_1 - a}x_1 + \frac{(c - y_2)}{y_1 - a}.$$

Let p be any prime such that $p \nmid a \Rightarrow p \nmid \mathrm{LC}(R)$. By inspection, one sees that the evaluation points $\{(\alpha_1, \alpha_2) \in \mathbb{Z}_p^2 : \alpha_1 = a, \text{ and } \alpha_2 \in \mathbb{Z}_p\}$ are **bad** and $\{(\alpha_1, \alpha_2) \in \mathbb{Z}_p^2 : \alpha_1 = b \text{ or } \alpha_2 = c\}$ **cause missing terms**.

Lemma 6.10. Let p be a prime that does not divide any integer coefficient of R and R_j. If an evaluation point $\alpha \in \mathbb{Z}_p^m$ is chosen at random then

$$\Pr[\,\alpha \text{ is bad or causes missing terms}\,] \leq \frac{2nsD_{R_y}d_{\max,1}}{p}. \tag{6.5}$$

Proof. Using Lemma 2.17 (the Schwartz-Zippel Lemma), we have that

$$\Pr[\,\alpha \text{ is bad}\,] = \Pr[\mathrm{LC}(R)(\alpha) = 0] \leq \frac{\deg(\mathrm{LC}(R))}{p} \leq \frac{D_{R_y}}{p}. \tag{6.6}$$

Now we address the case when the evaluation point α causes missing terms. Let $l = \#R_j$'s to be interpolated and let

$$\Delta(y_1, \ldots, y_m) = \prod_{j=1}^{l} \prod_{k=0}^{T_j - 1} f_{jk}.$$

Since T_j is the number of the rational function coefficients in each R_j, we have

$$\left(\sum_{j=1}^{l} T_j \right) \leq \left(\sum_{j=1}^{l} d_{T_j} \right) \leq \sum_{j=1}^{l} \deg(R_j, x_1) \leq \deg(R, x_1) \leq nsd_{\max,1}.$$

Therefore,

$$\deg(\Delta) = \sum_{j=1}^{l} \sum_{k=0}^{T_j - 1} \deg(f_{jk}) \leq \sum_{j=1}^{l} T_j \sum_{i=1}^{m} \deg(R_j, y_i) \leq nsD_{R_y} d_{\max,1}.$$

Thus, by Lemma 2.17,

$$\Pr[\,\alpha \text{ causes missing terms}] = \Pr[\Delta(\alpha) = 0] \leq \frac{\deg(\Delta)}{p} \leq \frac{nsD_{R_y} d_{\max,1}}{p}. \tag{6.7}$$

Adding (6.6) and (6.7) completes our proof. □

6.3.2 Primes

Let $\phi_p : \mathbb{Z}[y_1, \ldots, y_m][x_1] \to \mathbb{Z}_p(y_1, \ldots, y_m)[x_1]$ be the modular mapping $\phi_p(R_j) = R_j \bmod p$. For the remainder of this chapter, let $P_s = \{p_{N_1}, p_{N_2}, \ldots, p_{N_s}\}$ be the list of pre-computed smooth primes to be used in Algorithm 19 such that $p_{s_{\min}} = \min_{i=1}^{N_s}\{p_{N_i}\}$ and $N_s = |P_s|$, and let $P = \{p_1, p_2, \ldots, p_N\}$ be the list of pre-computed primes (not necessarily smooth) to be used in Algorithm 20 such that $p_{\min} = \min_{i=1}^{N}\{p_i\}$, where $N = |P|$. For simplicity, we will also assume that $N \geq N_s$ and $p_{\min} \geq p_{s_{\min}}$.

Generating random primes

For efficiency purposes, we used 62 bit primes in our Dixon resultant algorithm because of our hybrid Maple+C implementation of our algorithm. Thus, the list of primes P and P_s both contain 62 bit primes with $p_{\min}, p_{s_{\min}} > 2^{61}$.

In order to obtain a low failure probability for our Dixon resultant algorithm, we want $N, N_s \geq 10^9$. However, it is not efficient to generate the lists of primes P and P_s with more

than a billion primes as this computation will take a very long time. We discuss how to generate a random prime from the lists of primes P and P_s without creating the lists.

A random 62 bit prime from $[2^{61}, 2^{62}]$ can be generated by first choosing a random integer $c \in [2^{61}, 2^{62}]$, then we pick the prime before or after the random integer c. Unfortunately, we do not currently have an algorithm that can generate a 62 bit smooth prime uniformly at random without creating the list of smooth primes P_s. We believe we have enough smooth primes for our Dixon resultant algorithm to fail with low probability because Monagan [Monagan, 2023a] estimates that there are about 10^{10} smooth primes in $(2^{60}, 2^{63})$ based on Algorithm RandomSmoothPrime which generates a random smooth prime (not uniformly) $p \in (2^{60}, 2^{63})$.

Algorithm RandomSmoothPrime

repeat

Pick $q_i \in [750, 2500]$ for $1 \leq i \leq 6$ at random.

$p \leftarrow 1 + \prod_{i=1}^{6} q_i$.

until $p \in (2^{60}, 2^{63})$ and p is prime.

Now we characterize the primes that must be avoided in our Dixon resultant algorithm.

Definition 6.11. We say a prime p is **bad** if p divides $\mathrm{LC}(R)$. We also say a prime p **causes missing terms** if p divides any integer coefficient of R_j.

Example 6.12. Suppose the Dixon resultant

$$R = 15y_2x_1^2 + (7y_1 - 49)x_1 + 7$$

and let

$$S = x_1^2 + \frac{(7y_1 - 49)}{15y_2}x_1 + \frac{7}{15y_2}.$$

The primes $3, 5$ are bad and prime 7 causes missing terms.

Example 6.13. Suppose the Dixon resultant

$$R = (3137y_2 + 3)x_1^2 + (7y_1 + 1)x_1 + 7$$

and let

$$S := R_1 = x_1^2 + \frac{(7y_1 + 1)}{3137y_2 + 3}x_1 + \frac{7}{3137y_2 + 3}.$$

Notice that $\phi_{3137}(\mathrm{LC}(R)) = 3 \neq 0$, which means the image

$$\phi_{3137}(S) = x_1^2 + 1046(7y_1 + 1)x_1 + 1048.$$

Clearly, there are missing terms in the denominators of $\phi_{3137}(S)$.

By design, our Dixon resultant algorithm returns an answer when the rational number reconstruction process succeeds on the R_j's for the first prime (See Lines 40-41 of Algorithm 19). If the rational number reconstruction process does not succeed with the first prime, then more primes are used to recover the R_j's. Our algorithm is designed this way because we do not know the number of primes needed a priori since R is represented by black box **BB**, and we want to use the fewest number of primes possible. Therefore, in Example 6.13, if $\phi_{3137}(S)$ is the first image obtained, the rational number reconstruction process will succeed with the input prime $p = 3137$ and Algorithm 19 will return

$$S = x_1^2 + \left(\frac{7y_1}{3} + \frac{1}{3}\right) x_1 + \frac{7}{3}$$

as the answer even though it is wrong. We have already discussed how to check probabilistically if the returned R_j's are correct in Subsection 5.3.1 (Algorithm 24).

We now bound the failure probability of a prime p is bad or p causes missing terms.

Proposition 6.14. Let P be the list of primes such that $|P| = N$ and $p_{\min} = \min(P)$. If p is chosen at random from P then the probability that p is bad or p causes missing terms in the monic square-free factor R_j

$$\leq \frac{\log_{p_{\min}}(\#R_j \|R_j\|_\infty \|R\|_\infty)}{N}.$$

Proof. Let c be an integer coefficient of an R_j. The number of primes p that can divide c from the list of primes P is at most $\lfloor \log_{p_{\min}} c \rfloor$. So

$$\Pr[p \text{ divides } c] \leq \frac{\log_{p_{\min}} c}{N}.$$

By definition 6.11, p is bad $\iff p | \mathrm{LC}(R) \implies p$ divides one term in $\mathrm{LC}(R)$. So

$$\Pr[p \text{ is bad}] = \Pr[p \text{ divides } \mathrm{LC}(R)] \leq \Pr[p \text{ divides one term in } \mathrm{LC}(R)] \leq \frac{\log_{p_{\min}} \|R\|_\infty}{N}. \tag{6.8}$$

Furthermore, the probability that p causes missing terms (See Definition 6.11) is at most

$$\frac{\log_{p_{\min}}(\#R_j \|R_j\|_\infty)}{N}. \tag{6.9}$$

Adding (6.8) and (6.9) completes our proof. □

Corollary 6.15. If p is chosen at random from the list of primes P and p is not bad then

$$\Pr[\mathrm{supp}(\phi_p(R_j)) \neq \mathrm{supp}(R_j)] \leq \frac{\log_{p_{\min}}(\#R_j \|R_j\|_\infty)}{N}.$$

Proof. This follows from the above proposition. □

6.3.3 Monic Univariate Polynomial Images of R

Recall that Subroutine 16 (Subroutine PolyInterp) interpolates monic polynomial images of R in x_1 with high probability for one monic square-free factor S. The integer coefficients of these monic univariate polynomial images are what we use to interpolate the monic square-free factor S. Therefore, we must avoid evaluation points and primes that are bad. We must also avoid evaluation points and primes that could cause these monic univariate polynomial images of R in x_1 to lose their support (the univariate monomials in x_1 disappear).

A bad evaluation point can be detected in the same way as a bad prime. This is detected with high probability in Subroutine 16 by checking that the degree of the interpolated univariate monic polynomial images of R is the same as the degree of R in x_1. Line 5 of Subroutine 16 detects the occurrence of a bad evaluation point or a bad prime.

Similarly, an evaluation point that causes the supports of the interpolated monic polynomial images of R in x_1 to disappear can be detected in the same way as a prime that causes the supports of these images to vanish. If the set of degrees $d = \{d_0, \ldots, d_T\}$ as defined in (5.2) are the same as the degrees of the support of the interpolated monic polynomial images H in Subroutine 16, then we know we have a correct univariate monic polynomial image of R with high probability. We detect this in Line 7 of Subroutine 16. Otherwise, we interpolate a monic square-free factor R_j that have missing terms. We now find the probability that Subroutine 16 returns FAIL.

Lemma 6.16. Assume the degrees $[d_0, \ldots, d_T]$ as defined in (5.2) are correct. Let $e_{\max} = 2 + \max_{k=0}^{T-1} (\deg(f_k) + \deg(g_k))$. If prime p is chosen at random from the list of primes P such that $|P| = N$ and $p_{\min} = \min(P)$ then the probability that Subroutine 16 returns FAIL

$$\leq e_{\max} \left(\frac{2nsD_{R_y} d_{\max,1}}{p} + \frac{\log_{p_{\min}} (\|S\|_\infty \|R\|_\infty nsd_{\max,1})}{N} \right).$$

Proof. There are two sources of failure in Subroutine 16. First, if an input evaluation point $Z_j \in \mathbb{Z}_p^m$ is bad for any $1 \leq j \leq e_{\max}$ or an input prime p is bad then the degree of the interpolated monic polynomial images B_j of R in x_1 denoted by $\deg(B_j, x_1) < \hat{D} = \deg(R, x_1)$ in Line 5. Thus,

$$\begin{aligned}
&\Pr[\text{prime } p \text{ or evalaution point } Z_j \text{ is bad in Line 5 of Subroutine 16}] \\
&\leq \Pr[p \text{ divides LC}(R)] + \Pr[\text{LC}(R)(Z_j) = 0] \\
&\leq \Pr[p \text{ divides 1 term in } \text{LC}(R)] + \Pr[\text{LC}(R)(Z_j) = 0] \\
&\leq \frac{\log_{p_{\min}} \|\text{LC}(R)\|_\infty}{N} + \frac{\deg(\text{LC}(R))}{p} \\
&\leq \frac{\log_{p_{\min}} \|R\|_\infty}{N} + \underbrace{\frac{D_{R_y}}{p}}_{\text{by (6.6)}}.
\end{aligned} \tag{6.10}$$

Now we assume that prime p and Z_j are not bad. If $Z_j \in \mathbb{Z}_p^m$ or p causes missing terms, then $\text{supp}(H_j) \neq [d_0, \ldots, d_T]$ in Line 7 of Subroutine 16 where H_j is the monic square-free part of the univariate polynomial B_j. Since T is the number of numerator polynomials f_k in S, we have that $T \leq \deg(R, x_1) \leq nsd_{\max,1}$. Thus

$$\begin{aligned}
&\Pr[\, Z_j \text{ or } p \text{ causes missing terms in Line 7 of Subroutine 16}] \\
&\leq \Pr[\text{Any } f_k(Z_j) = 0 \text{ for } 0 \leq k \leq T-1] + \Pr[\, p \,\text{divides any } f_k \text{ in } S \text{ for } 0 \leq k \leq T-1] \\
&\leq \Pr[\prod_{k=1}^{T-1} f_k(Z_j) = 0] + \Pr[p \,\text{divides one term of } f_k \text{ in } S \text{ for } 0 \leq k \leq T-1] \\
&\leq \Pr[\prod_{k=0}^{T-1} f_k(Z_j) = 0] + \frac{\log_{p_{\min}}(\|S\|_\infty T)}{N} \\
&\leq \underbrace{\frac{nsD_{R_y} d_{\max,1}}{p}}_{by\ (6.7)} + \frac{\log_{p_{\min}}(\|S\|_\infty nsd_{\max,1})}{N}. \qquad (6.11)
\end{aligned}$$

Hence Pr[Subroutine 16 returns FAIL] $\leq e_{\max}((6.10) + (6.11))$. □

Remark 6.17. If Algorithm 19 calls Subroutine 16 then the list of primes P is replaced with the list of smooth primes P_s. Thus, the above theorem works accordingly.

6.3.4 Unlucky Content

Definition 6.18. Let the Dixon resultant

$$R = \sum_{k=0}^{\hat{d}} \bar{r}_k(y_1, \ldots, y_m) x_1^k \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1]$$

where $\bar{r}_k \neq 0$ and $\hat{d} = \deg(R, x_1) > 0$. Let p be a prime such that $\phi_p(\bar{r}_{\hat{i}}) \neq 0$ for all i. Let the polynomial content of R be denoted by $C = \gcd(\bar{r}_0, \bar{r}_1 \cdots, \bar{r}_{\hat{d}})$. We say p **causes unlucky content** if $\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \phi_p\left(\frac{\bar{r}_1}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{\hat{d}}}{C}\right)\right) \neq 1$.

Example 6.19. Let prime $p \neq 2$ and let the Dixon resultant

$$R = (2y_1^2 + 2y_1 + 2py_1)x_1 + (2y_1^2 + 2y_1).$$

Notice that the polynomial content $C = 2y_1$, but

$$\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \phi_p\left(\frac{\bar{r}_1}{C}\right)\right) = y_1 + 1 \neq 1.$$

Thus, our Dixon resultant algorithm (Algorithm 19) will output

$$S = x_1 + 1,$$

instead of the correct answer

$$S = x_1 + \frac{y_1 + 1}{y_1 + p + 1},$$

because p has caused an unlucky content.

Unfortunately, we cannot detect in advance the occurrence of an unlucky content in our algorithm because of the black box representation of R. But it can be detected after our algorithm terminates. We have already discussed how to detect this in Subsection 5.3.1.

Theorem 6.20. Let $R = \sum_{k=0}^{\hat{d}} \bar{r}_k(y_1, \ldots, y_m)x_1^k$ such that $\bar{r}_k \neq 0$ and $\bar{r}_k \in \mathbb{Z}[y_2, y_3, \ldots, y_m][y_1]$. Let p be a prime chosen at random from the list of primes P where $p_{\min} = \min(P)$. If $\phi_p(\mathrm{LC}(\bar{r}_i)) \neq 0$ for all i, then the probability that p causes unlucky content

$$\leq \frac{2nmsD_{\max} \log_{p_{\min}}((1 + 2nsD_{\max})\|R\|_\infty)}{N}.$$

Proof. Clearly,

$$\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{\hat{d}}}{C}\right)\right)\right) > 0 \Longrightarrow \deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_i}{C}\right), \phi_p\left(\frac{\bar{r}_j}{C}\right)\right)\right) > 0$$

for any $0 \leq i \neq j \leq \hat{d}$ which implies that

$$\deg\left(\gcd(\phi_p(\bar{r}_i), \phi_p(\bar{r}_j))\right) > 0 \Longrightarrow \deg\left(\gcd(\phi_p(\bar{r}_i), \phi_p(\bar{r}_j)), y_k\right) > 0$$

for at least one k, say $k = 1$. So prime p causes unlucky content if

$$\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{\hat{d}}}{C}\right)\right)\right) > 0 \Longrightarrow \deg(\gcd(\phi_p(\bar{r}_0), \phi_p(\bar{r}_1)), y_1) > 0 \iff \phi_p(\overline{R}_{0,1}) = 0$$

where $\overline{R}_{0,1}$ is the Sylvester resultant of $\bar{r}_0, \bar{r}_1$ in y_1. Therefore,

$$\Pr[\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{\hat{d}}}{C}\right)\right)\right) > 0] \leq \Pr[\phi_p(\overline{R}_{0,1}) = 0] \leq \frac{\log_{p_{\min}}\|\overline{R}_{0,1}\|_\infty}{N}.$$

To complete our proof, we need a bound for $\|\overline{R}_{0,1}\|_\infty$. We obtain a bound $\|\overline{R}_{0,1}\|_\infty$ as follows. Let $\hat{S}$ be the Sylvester matrix whose entries are coefficients of $\bar{r}_0$ and $\bar{r}_1$ in y_1. Thus, $\overline{R}_{0,1} = \det(\hat{S})$. The dimension of $\hat{S}$ denoted by

$$\dim(\hat{S}) \leq \deg(\bar{r}_0, y_1) + \deg(\bar{r}_1, y_1) \leq 2\deg(R, y_1) \leq 2nsD_{\max,1} \leq 2nsD_{\max}$$

by Theorem 6.5. Let $t_{\max} = \max_{j,k}\{\#\hat{S}_{j,k}\}$. Clearly,

$$t_{\max} \leq \prod_{k=2}^{m}(1 + \deg(R, y_k)) \leq \prod_{k=2}^{m}(1 + 2nsD_{\max,k}) \leq (1 + 2nsD_{\max})^{m-1}$$

by Theorem 6.5. Using Theorem 6.2, we get

$$\begin{aligned}\|\overline{R}_s\|_\infty &= \|\det(\hat{S})\|_\infty \ < (t_{\max}\ \|\hat{S}_{jk}\|_\infty\ (2nsD_{\max})^{\frac{1}{2}})^{2nsD_{\max}} \\ &< ((1+2nsD_{\max})^{m-1}\ \|R\|_\infty\ 2nsD_{\max})^{2nsD_{\max}} \\ &< (\|R\|_\infty\ (1+2nsD_{\max})^m)^{2nsD_{\max}}\end{aligned}$$

and we are done. □

6.3.5 Auxiliary Univariate Rational Functions

Using a Kronecker substitution, we remind the reader that the interpolation of the multivariate rational function coefficients $\frac{f_k(y_1,\ldots,y_m)}{g_k(y_1,\ldots,y_m)}$ of the R_j's is reduced to a univariate rational function interpolation in our Dixon resultant algorithm. Let $r = (r_1, r_2, \ldots, r_{m-1}) \in \mathbb{Z}^{m-1}$ where $r_i > 0$ and let $K_r : \mathbb{Z}_p(y_1, y_2, \ldots, y_m) \to \mathbb{Z}_p(y)$ be the Kronecker substitution

$$K_r(f_k/g_k) = \frac{f_k(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})}{g_k(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{m-1}})} \in \mathbb{Z}_p(y)$$

where $r_i > \max\left(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i))\right)$, prime $p > \prod_{j=1}^m r_i$ and f_k and g_k are as in (5.2). Let α be a generator for $\mathbb{Z}_p^*$, and let $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ serve as a basis shift, which is chosen at random, as described in Algorithm 19 in Lines 8-13. Let

$$F_k(y, z, \beta) := \frac{f_k^\beta(y,z)}{g_k^\beta(y,z)} = \frac{f_k(zy+\beta_1, zy^{r_1}+\beta_2, \ldots, zy^{(r_1 r_2 \cdots r_{m-1})}+\beta_m)}{g_k(zy+\beta_1, zy^{r_1}+\beta_2, \ldots, zy^{(r_1 r_2 \cdots r_{m-1})}+\beta_m)} \in \mathbb{Z}_p(y)(z).$$

Recall that the introduction of the basis shift β ensures that the functions $F_k(\alpha^i, z, \beta)$ are normalized in Line 4 of Subroutine 22 using the constant term produced by $g_k^\beta(\alpha^i, z)$. If g_k has a constant term, then we use $\beta = (0, \ldots, 0)$ in Line 8 of Algorithm 19.

Definition 6.21. We say that $\alpha \in \mathbb{Z}_p \setminus \{0\}$ is a **bad evaluation point** if $\deg(f_k^\beta(\alpha, z)) < \deg(f_k, z)$ or $\deg(g_k^\beta(\alpha, z)) < \deg(g_k, z)$ for any k. That is, $\mathrm{LC}(f_k^\beta(\alpha, z)) = 0$ or $\mathrm{LC}(g_k^\beta(\alpha, z)) = 0$ We say that $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ is a **bad basis shift** if $\gcd(f_k, g_k) = 1$ but the degree $\deg(\gcd(f_k^\beta(\alpha, z), g_k^\beta(\alpha, z))) > 0$ for any k.

Definition 6.22. We say a prime p is **unlucky** if $p|\mathrm{LC}(f_k^\beta(y, z))$ in z or $p|\mathrm{LC}(g_k^\beta(y, z))$ in z for any k.

Example 6.23. Let

$$f_1/g_1 = \frac{2891y_1 + y_2 + y_3}{y_2^2 + y_1 + y_3} \in \mathbb{Z}_{3137}(y_1, y_2, y_3).$$

Clearly, $\gcd(f, g) = 1$. The rational function f_1/g_1 does not have a constant term in the numerator or denominator so we need a basis shift. Let $\beta = (5, 2, 3) \in \mathbb{Z}_{3137}$ serve as the

basis shift for A. Let $r = (2, 3)$ and let

$$K_r(f_1/g_1) = \frac{f_1(y, y^2, y^6)}{g_1(y, y^2, y^6)} = \frac{y^6 + y^2 + 2891y}{y^6 + y^4 + y}.$$

Then an auxiliary rational function $F_1(y, z, \beta)$ with a Kronecker substitution K_r is

$$F_1(y, z, \beta) = \frac{f_1^\beta(y, z)}{g_1^\beta(y, z)} = \frac{1912 + (y^6 + y^2 + 2891y)z}{12 + y^4z^2 + (y^6 + 4y^2 + y)z} \in \mathbb{Z}_{3137}[y](z).$$

If $\alpha = 3$ is randomly picked in $\mathbb{Z}_{3137}^*$, then the auxiliary rational function

$$F_1(3, z, \beta) = \frac{f_1^\beta(3, z)}{g_1^\beta(3, z)} = \frac{1912}{81z^2 + 768z + 12} \in \mathbb{Z}_{3137}(z).$$

Thus, $\deg(f_1(\alpha, z)) < 1$ because $\mathrm{LC}(f_1^\beta(3, z)) = 9411 = 3 \times 3137 \equiv 0 \pmod p$ which implies that $\alpha = 3$ is a bad evaluation point.

To avoid the occurrence of bad evaluation points with high probability in Algorithm 19, we remind the reader that we interpolate $F_k(\alpha^{\hat{s}+i}, z, \beta)$ for some random $\hat{s} \in [0, p-2]$ instead of $F_k(\alpha^i, z, \beta)$ for $i = 0, 1, 2, \cdots$. This is labelled as A_j in Line 24 of Algorithm 19. Line 25 detects bad evaluation points, a bad basis shift and a prime p that is unlucky.

Example 6.24. Let

$$f_1/g_1 = \frac{y_1 + y_2 + y_3}{py_2^2 + y_1 + y_3} \in \mathbb{Z}(y_1, y_2, y_3).$$

Let p be a prime and let $\beta = (5, 2, 3)$ So

$$F_1(y, z, \beta) = \frac{f_1^\beta(y, z)}{g_1^\beta(y, z)} = \frac{10 + (y^6 + y^2 + y)\, z}{8 + (y^6 + y)\, z} \in \mathbb{Z}_p[y](z).$$

Notice that $\deg(g_1^\beta(\alpha, z)) < 2$ for any α since $p | \mathrm{LC}(g_1^\beta(y, z))$. Thus p is an unlucky prime.

Example 6.25. Let p be a prime and let

$$\frac{f_1}{g_1} = \frac{y_1}{(y_1 + y_3)y_2} \in \mathbb{Z}_p(y_1, y_2, y_3).$$

Observe that the partial degrees $d_i = \max\{\deg(f_1, y_i), \deg(g_1, y_i)\} = 1$ for $1 \le i \le 3$. For the Kronecker map K_r to be invertible, we need $r_i > d_i$, so let $r = (2, 2)$. Thus,

$$K_r(f_1/g_1) = \frac{f(y, y^2, y^4)}{g(y, y^2, y^4)} = \frac{y}{(y + y^4)y^2} = \frac{y}{y^3 + y^6}.$$

Since g_1 has no constant term, we need a non-zero basis shift β. To interpolate $K_r(f_1/g_1)$, we need to densely interpolate $F_1(\alpha^j, z, \beta)$ for $1 \le j \le 4 = 2 \times \#g_1$. Computing $F_1(\alpha, z, \beta)$

directly yields the univariate rational function

$$F_1(\alpha, z, \beta) = \frac{f_1^{\beta}(\alpha, z)}{g_1^{\beta}(\alpha, z)} = \frac{\alpha z + \beta_1}{(z\alpha^4 + z\alpha + \beta_1 + \beta_3)(z\alpha^2 + \beta_2)}.$$

The Sylvester resultant $\mathcal{R} = \text{Res}(f_1^{\beta}(\alpha, z), g_1^{\beta}(\alpha, z), z) = \alpha^2(\alpha^3\beta_1 - \beta_3)(\alpha\beta_1 - \beta_2) \neq 0$ since $\alpha \neq 0$ and $\beta = (\beta_1, \beta_2, \beta_3) \neq (0, 0, 0)$. But, if $\beta_2 = \alpha\beta_1 \neq 0$ or $\beta_3 = \alpha^3\beta_1 \neq 0$ then $\mathcal{R}(\beta) = 0$ which implies that β is a bad basis shift.

Theorem 6.26. Let N_a be the number of auxiliary rational functions needed by Algorithm 19. If a smooth prime p is chosen at random from the list of primes P_s with $|P_s| = N_s$ and $p_{s_{\min}} = \min(P_s)$ then the probability that Algorithm 19 returns FAIL in Line 25 is at most

$$\frac{3N_a D_{R_y}^2 nsd_{\max,1} \prod_{j=1}^{m}(1 + nsD_{\max,j})}{p-1} + \frac{2N_a \log_{p_{s_{\min}}}(\|S\|_{\infty} nsd_{\max,1})}{N_s}.$$

Proof. There are three causes of FAIL in Line 25 of Algorithm 19. They are bad evaluation points, a bad basis shift and an unlucky prime p. We remark that all three failure causes are a direct consequence of our attempt to interpolate auxiliary rational functions A_j in Line 24 of Algorithm 19 when it calls Subroutine Ratfun.

1. Bad evaluation point case: Let

$$\Delta(y) = \prod_{k=0}^{T-1} \text{LC}(f_k^{\beta}(y, z))\text{LC}(g_k^{\beta}(y, z))$$

where the univariate polynomial $\Delta \in \mathbb{Z}_p[y]$. For $0 \leq j \leq N_a - 1$, the evaluation point $\alpha^{\hat{s}+j-1}$ in Line 18 is random on $[1, p)$, since $\hat{s} \in [0, p-2]$ is random. Since a basis shift β does not affect the degree and the leading coefficients of auxiliary rational functions, we have that, if $\alpha^{\hat{s}+j-1}$ is a bad evaluation point then $\Delta(\alpha^{\hat{s}+j-1}) = 0$. Since T is the number of numerator polynomials f_k in S, we have that $T \leq \deg(R, x_1) \leq nsd_{\max,1}$. Also, recall that

$$r_i = 1 + \max\left(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i))\right) \leq 1 + \deg(R, y_i) \leq 1 + nsD_{\max,i}.$$

Thus,

$$\deg(\Delta(y)) = 2T \prod_{j=1}^{m} r_i \leq 2nsd_{\max,1} \prod_{j=1}^{m}(1 + nsD_{\max,j}).$$

Therefore,

$$\begin{aligned} &\Pr[\alpha^{\hat{s}+j-1} \text{ is bad for any } 0 \leq j \leq N_a - 1] \leq \frac{N_a \deg(\Delta)}{p-1} \\ &\leq \frac{2N_a nsd_{\max,1} \prod_{j=1}^{m}(1 + nsD_{\max,j})}{p-1}. \end{aligned} \tag{6.12}$$

Notice that the bound $\prod_{j=1}^{m}(1+nsD_{\max,j})$ is an exponential contribution in (6.12). Furthermore, the bound in (6.12) allows $\Delta(y)$ to have $\deg(\Delta(y))$ roots. Fortunately, in practice, we do not encounter such high degree for $\Delta(y)$ because the average number of roots of a random $\Delta(y)$ over $\mathbb{Z}_p$ is 1 [Schmidt, 2006, Chapter 4].

2. Bad basis shift case: We will now handle the basis shift case. Suppose $\theta_j := \alpha^{\hat{s}+j-1}$ is not a bad evaluation point for $1 \leq j \leq N_a$. Let $w_1, w_2, \cdots w_m$ be new variables and let

$$G_{kj} = \frac{\hat{f}_{k_j}(w_1,\ldots,w_m)}{\hat{g}_{k_j}(w_1,\ldots,w_m)} = \frac{f_k(\theta_j z + w_1,\ldots,z\theta_j^{(r_1 r_2\cdots r_{m-1})} + w_m)}{g_k(\theta_j z + w_1,\ldots,z\theta_j^{(r_1 r_2\cdots r_{m-1})} + w_m)} \in \mathbb{Z}_p(w_1,w_2,\ldots,w_m)(z).$$

Recall that a basis shift $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ does not affect the leading coefficients of auxiliary functions, so $\mathrm{LC}(\hat{f}_{k_j})(\beta) \neq 0$ and $\mathrm{LC}(\hat{g}_{k_j})(\beta) \neq 0$. Let

$$\overline{R}_{kj} = \mathrm{Res}(\hat{f}_{k_j}, \hat{g}_{k_j}, z) \in \mathbb{Z}_p[w_1, w_2, \ldots, w_m]$$

be the Sylvester resultant of $\hat{f}_{k_j}$ and $\hat{g}_{k_j}$ taken in z and let

$$\Delta(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{N_a}\prod_{k=0}^{T-1} \overline{R}_{kj}.$$

Clearly, β is a bad shift if and only if

$$\deg(\gcd(\hat{f}_{k_j}(z,\beta), \hat{g}_{k_j}(z,\beta)) > 0 \iff \Delta(\beta) = 0$$

for any k and j. We now find a bound for $\deg(\Delta)$. Using Bezout's bound, we have

$$\deg(\overline{R}_{kj}) \leq \deg(\hat{f}_{k_j})\deg(\hat{g}_{k_j}) \leq \deg(f_k)\deg(g_k) \leq D_{R_y}^2.$$

Hence,

$$\deg(\Delta) \leq \sum_{k=0}^{T-1}\sum_{j=1}^{N_a}\deg(\overline{R}_{kj}) \leq N_a T\left(\sum_{j=1}^{m}\deg(R, y_k)\right)^2 \leq N_a nsd_{\max,1}D_{R_y}^2.$$

Therefore,

$$\Pr\ [\beta \text{ is bad basis shift}] = \Pr[\Delta(\beta) = 0] \leq \frac{N_a nsd_{\max,1}D_{R_y}^2}{p-1}. \tag{6.13}$$

3. Unlucky Primes: Finally, we handle the case when p is unlucky. That is, prime p causes the degree of a numerator or denominator polynomial in A_j in z to drop lower than

$\deg(f_k)$ or $\deg(g_k)$. Note that

$$\begin{aligned}
&\Pr[p \text{ is unlucky in } A_j] \leq \Pr[\,p \,\text{divides any } f_k \text{ or } g_k \text{ in } S] \\
&\leq \Pr[p \text{ divides one integer coefficient of } f_k \text{ or } g_k \text{ in } S] \\
&\leq \frac{2\log_{p_{s_{\min}}}(T\|S\|_\infty)}{N_s} \leq \frac{2\log_{p_{s_{\min}}}(\|S\|_\infty nsd_{\max,1})}{N_s}
\end{aligned}$$

because there are T numerator and denominator polynomials f_k and g_k in S. Thus,

$$\begin{aligned}
&\Pr[p \text{ is unlucky in } A_j \text{ for any } \ 1 \leq j \leq N_a] \\
&= N_a \times \Pr[p \text{ is unlucky}] \leq \frac{2N_a \log_{p_{s_{\min}}}(\|S\|_\infty nsd_{\max,1})}{N_s}. \qquad (6.14)
\end{aligned}$$

Adding (6.12), (6.13) and (6.14) completes our proof. □

6.3.6 Discovering the supports of the polynomials $f_{i,k}$ and $g_{i,k}$

We would like to determine the probability of failure of finding the correct support for the polynomials $f_{i,k}$ and $g_{i,k}$, and their support sizes (number of terms) in our Dixon resultant algorithm. Subroutine BMStep (Subroutine 17) was designed to get the correct number of terms when it receives an input prime p, and an input array P containing a sequence of coefficients from the univariate auxiliary rational functions in z such that $|P|= i$ and i is even. This subroutine uses the Berlekamp-Massey Algorithm (BMA) to generate a feedback polynomial $\lambda(z) \in \mathbb{Z}_p[z]$ in Line 2 when the condition

$$\deg(\lambda) < \frac{i}{2} \qquad (6.15)$$

is satisfied, and the number of roots of $\lambda(z)$ over $\mathbb{Z}_p$ yields the number of terms in the polynomial to be interpolated ($f_{i,k}$ or $g_{i,k}$) with high probability.

However, it is possible that an incorrect $\lambda(z)$ is produced even if the condition (6.15) is satisfied. Thus, the wrong number of terms, and consequently the wrong $f_{i,k}$ or $g_{i,k}$ are interpolated. To obtain a failure probability bound, we first state the following important result proved by Kaltofen, Lee and Lobo in [Kaltofen et al., 2000].

Theorem 6.27. [Kaltofen et al., 2000, Theorem 3] Let $f(y_1, y_2, \ldots, y_m) \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$ be a polynomial with t terms. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ be chosen uniformly at random from a finite set $\bar{S} \subset \mathbb{Z}$. If we run the BMA on the sequence $[f(\alpha_1^i, \alpha_2^i, \ldots, \alpha_m^i) : i \geq 0]$ then a zero discrepancy is encountered after $2t$ points with probability of at least $1 - \frac{t(t+1)(2t+1)}{6|\bar{S}|}$.

A zero discrepancy means that two consecutive feedback polynomials generated by the BMA would be the same, implying that the correct term bound has been found with high probability. That is, if we compute $\lambda(z)$ for $j = 2, 4, 6, \cdots$ points, we will see that $\deg(\lambda) =$

$1, 2, 3, \ldots, t-2, t-1, t, t, t \ldots$. Thus, the above theorem assures us of obtaining the correct feedback polynomial $\lambda(z)$ (correct number of terms) with high probability whenever we run the BMA on an input of length i containing a sequence of points such that $i > 2t$.

In practice, an input sequence of length $2t$ would yield a feedback polynomial of degree t (t is also the number of terms in f). Therefore, our stopping condition (6.15) definitely obeys Theorem 6.27 because we are using at least two extra points to confirm that the correct $\lambda(z)$ is found each time the BMA is called in Line 2 of Subroutine 17. As we have said earlier, the condition (6.15) may be satisfied, but still the wrong feedback polynomial is obtained because a zero discrepancy is encountered when $i < 2t$.

By design, Line 2 of Subroutine 17 will be able to detect that there is a problem by returning FAIL if $\deg(\lambda) \neq t_r$ where t_r is the number of roots of λ. But it will not be able to detect the case when the feedback polynomial stabilizes too early and the number of roots obtained is equal to the degree of the feedback polynomial. That is, $\deg(\lambda) = t_r \neq t$. This case will only be discovered by our Dixon result algorithm at termination when we check if the returned answer is incorrect.

Unfortunately, we cannot use the failure bound from Theorem 6.27 because $\bar{S} = \mathbb{Z}_p^*$ and $f = K_r(f_{i,k}) \in \mathbb{Z}_p[y]$ in our Dixon resultant algorithm because a Kronecker map K_r has been applied. We remind the reader again that $\#K_r(f_{i,k}) = \#f_{i,k}$. Fortunately, we have the following useful result which was proved by Hu in his PhD thesis [Hu, 2018]. Our Dixon resultant algorithm also checks that # roots of $\lambda(z)$ is equal to the degree of $\lambda(z)$ (See Line 4 of Subroutine 17).

Theorem 6.28. [Hu, 2018, Theorem 2.6] Let f be a univariate polynomial to be interpolated and let $t = \#f$, p be a prime and $p \gg \deg(f)$. Let α be any generator of $\mathbb{Z}_p^*$. Then the number of shift $\hat{s}$ which make the BMA encounter a zero discrepancy on $\left[f(\alpha^{\hat{s}}), f(\alpha^{\hat{s}+1}), \ldots, f(\alpha^{\hat{s}+2t})\right]$ is at most $\frac{t(t+1)\deg(f)}{2}$. Therefore, if $\hat{s}$ is chosen uniformly at random from $[0, p-2]$, then the probability that the BMA encounters a zero discrepancy for the first time at iteration $2t$ is at least $1 - \dfrac{t(t+1)\deg(f)}{2(p-1)}$.

We give the following example to illustrate the above result.

Example 6.29. Let

$$f = y^6 + 40y^5 + 45y^2 + 75y + 1 \in \mathbb{Z}_{103}[y].$$

Suppose we use the generator $\alpha = 87$. Since $t = \#f = 5$, we need $10+2$ points to determine the correct feedback polynomial, with the extra 2 points used for confirmation. Let $\hat{s} = 0$. We obtain

$$v = [59, 84, 8, 0, 64, 0, 96, 64, 94, 76, 85, 88]$$

by computing $v_j = f(\alpha^{\hat{s}+j})$, for $0 \leq j \leq 12$. Running the BMA on input $W_i = [v_1, v_2, \ldots, v_i]$, yields the feedback polynomials $\lambda(z)$ recorded in the following table.

# of points used (i)	$\lambda(z)$	Number of roots of $\lambda(z)$	$\deg(\lambda) < \frac{i}{2}$	$i > 2t$
2	$30 + z$	1	NO	NO
4	$z^2 + 84z + 95$	0	NO	NO
6	$z^2 + 84z + 95$	0	YES	NO
10	$z^5 + 43z^4 + 76z^3 + 93z^2 + 25z + 71$	5	YES	NO
12	$z^5 + 43z^4 + 76z^3 + 93z^2 + 25z + 71$	5	YES	YES

By design, our Dixon resultant algorithm will terminate at 6 points, which means we have an incorrect feedback polynomial. Hu in his PhD thesis [Hu, 2018, Example 2.6] has an example where the feedback polynomial $\lambda(z)$ terminates too early while the number of roots of λ is equal to $\deg(\lambda)$.

We are now ready to give a failure probability bound for Subroutine 17.

Theorem 6.30. Let N_a be the number of auxiliary rational functions needed by Algorithm 19. If a smooth prime p is chosen at random from the list of primes P_s with $|P_s| = N_s$ and $p_{s_{\min}} = \min(P_s)$ then the probability that Algorithm 19 returns FAIL in Lines 28 or 29 or 31 or 32 is at most

$$\frac{2N_a^2 D_{R_y} nsd_{\max,1} \prod_{j=1}^{m}(1 + nsD_{\max,j})}{p-1}.$$

Proof. Since N_a is the required number of auxiliary rational functions needed by Algorithm 19, it follows that Line 2 of Subroutine 17 will never return FAIL. However, the feedback polynomial $\lambda \in \mathbb{Z}_p[z]$ generated to find the number of terms in $f_{i,k}$ or $g_{i,k}$ in Line 4 of Subroutine 17 might be wrong, so it will return FAIL, which causes Algorithm 19 to return FAIL in either Lines 28 or 29 or 31 or 32.

Using Theorem 6.28, we have that the probability of getting the wrong $\#f_{i,k}$ or $\#g_{i,k}$ $\forall i$ and $\forall k$ using the Berlekamp-Massey Algorithm in Subroutine BMStep is at most

$$\sum_{k=0}^{T-1} \frac{\sum_{i=0}^{\deg(f_k)} \#f_{i,k}(\#f_{i,k}+1)\deg(K_r(f_{i,k}))}{2(p-1)} + \frac{\sum_{i=0}^{\deg(g_k)} \#g_{i,k}(\#g_{i,k}+1)\deg(K_r(g_{i,k}))}{2(p-1)}.$$

Since

$$T \leq nsd_{\max,1},$$

$$\#f_{i,k}, \#g_{i,k} \leq N_a,$$

$$\deg(f_{i,k}), \deg(g_{i,k}), \leq \deg(R) \leq D_{R_y},$$

and

$$\deg(K_r(f_{i,k})), \deg(K_r(g_{i,k})) \leq \prod_{j=1}^{m}(1 + nsD_{\max,j}),$$

we have that

$$\begin{aligned}
&\sum_{k=0}^{T-1} \frac{\sum_{i=0}^{\deg(f_k)} N_a(N_a+1)\prod_{j=1}^{m}(1+nsD_{\max,j})}{2(p-1)} + \frac{\sum_{i=0}^{\deg(g_k)} N_a(N_a+1)\prod_{j=1}^{m}(1+nsD_{\max,j})}{2(p-1)} \\
&\leq \frac{\prod_{j=1}^{m}(1+nsD_{\max,j})N_a^2 D_{R_y} T}{p-1} + \frac{\prod_{j=1}^{m}(1+nsD_{\max,j})N_a^2 D_{R_y} T}{p-1} \\
&\leq \frac{2\prod_{j=1}^{m}(1+nsD_{\max,j})N_a^2 D_{R_y} T}{p-1} \\
&\leq \frac{2N_a^2 D_{R_y} nsd_{\max,1}\prod_{j=1}^{m}(1+nsD_{\max,j})}{p-1}.
\end{aligned} \tag{6.16}$$

□

6.3.7 Monomial Evaluations

We have to solve for the coefficients of the polynomials $f_{i,k}$ and $g_{i,k}$ in Algorithm 20, whenever additional primes are required to interpolate the R_j's.

Algorithm 20 uses the support obtained from the first image to solve for the coefficients of a new image of the R_j's. However, it is possible that an evaluation point can cause two distinct monomials to evaluate to the same value in $\mathbb{Z}_p$. Lines 24 and 34 of Algorithm 20 both detect the occurrence of getting the same monomial evaluation. Thus, we need to obtain a failure probability bound for this case.

Lemma 6.31. Let q be an additional prime chosen at random from the list of primes P to be used by Algorithm 20 in order to get a new image of a monic square-free factor S. Let $p_{\min} = \min(P)$ and let $\hat{N}_{\max} = \max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\})$ where $f_{i,k}, g_{i,k}, f_k, g_k$ is as defined in (5.2). We have

$$\Pr[\text{Algorithm 20 returns FAIL in Line 24 or 34 }] \leq \frac{2nsd_{\max,1}\hat{N}_{\max}^2 D_{R_y}^2}{q-1}.$$

Proof. Let the support of $f_{i,k}$ (the monomials of $f_{i,k}$ in $y_1, y_2, \ldots, y_m$) be denoted by

$$\text{supp}(f_{i,k}) = [H_j(y_1, y_2, \ldots, y_m) : 1 \leq j \leq \#f_{i,k} \text{ where } \deg(H_j) = i]\,.$$

Let

$$J = \prod_{1\leq l\neq j\leq \#f_{i,k}} H_l(y_1, y_2, \ldots, y_m) - H_j(y_1, y_2, \ldots, y_m).$$

Let $\hat{m}_j = H_j(\hat{Y}_i)$ be the j-th monomial evaluation where $\hat{Y}_i = (\alpha_1^{\hat{s}+i-1}, \alpha_2^{\hat{s}+i-1}\cdots, \alpha_m^{\hat{s}+i-1})$ is the evaluation point in Line 15 of Algorithm 20 for $1 \leq i \leq \hat{N}_{\max}$, and $\hat{s} \in [0, q-2]$, and

$\alpha = (\alpha_1, \ldots, \alpha_m) \in (\mathbb{Z}_q \setminus \{0\})^m$ is picked at random in Line 13. By Lemma 2.17, we have

$$\Pr[\hat{m}_l = \hat{m}_j : 1 \leq l \neq j \leq \#f_{i,k}] = \Pr[J(\hat{Y}_i) = 0] \leq \frac{\binom{\#f_{i,k}}{2} \deg(f_{i,k})}{q-1}.$$

If the monomial evaluations obtained in Line 24 of Algorithm 20 or the monomial evaluations obtained in Line 24 of Subroutine 21 are not distinct, then

$$\begin{aligned}
&\Pr[\text{Algorithm 20 returns FAIL in Line 24 or 34}] \\
&\leq \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(f_k)} \binom{\#f_{i,k}}{2} \deg(f_{i,k})}{q-1} + \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(g_k)} \binom{\#g_{i,k}}{2} \deg(g_{i,k})}{q-1} \\
&\leq \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(f_k)} \hat{N}_{\max}^2 D_{R_y}}{2(q-1)} + \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(g_k)} \hat{N}_{\max}^2 D_{R_y}}{2(q-1)} \\
&\leq \frac{2T\hat{N}_{\max}^2 D_{R_y}(D_{R_y}+1)}{2(q-1)} \leq \frac{2nsd_{\max,1}\hat{N}_{\max}^2 D_{R_y}^2}{(q-1)}
\end{aligned}$$

since $\deg(f_{i,k}), \deg(g_{i,k}) \leq D_{R_y}$ for $\forall i$ and $\forall k$. □

6.3.8 Univariate Rational Functions without a Kronecker Substitution

Theorem 6.32. Let q be an additional prime chosen at random from the list of primes P to be used by Algorithm 20, in order to get a new image of a monic square-free factor S. Let $p_{\min} = \min(P)$ and let $\hat{N}_{\max} = \max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\})$ where $f_{i,k}, g_{i,k}, f_k, g_k$ is as defined in (5.2). The probability that Algorithm 20 returns FAIL in Line 27 is at most

$$\frac{3\hat{N}_{\max} nsd_{\max,1} D_{R_y}^2}{q-1} + \frac{2\hat{N}_{\max} \log_{p_{\min}}(\|S\|_\infty nsd_{\max,1})}{N}.$$

Proof. Similar to Theorem 6.26, we have three causes of FAIL in Line 27 of Algorithm 20. The failure causes are the presence of bad evaluation points, a bad basis shift and an unlucky prime q. We again remark that all three failure causes are a direct consequence of our attempt to interpolate auxiliary rational functions B_j in Line 26. The reader should note that auxiliary rational functions B_j are different from the A_j's interpolated in Algorithm 19 because a Kronecker substitution map is not used. Let

$$\Delta(y_1, y_2, \ldots, y_m) = \prod_{k=0}^{T-1} \mathrm{LC}(f_k^\beta)\mathrm{LC}(g_k^\beta) \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m]$$

where

$$\frac{f_k^\beta(y_1, y_2, \ldots, y_m, z)}{g_k^\beta(y_1, y_2, \ldots, y_m, z)} = \frac{f_k(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}{g_k(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}$$

Recall that a basis shift β does not affect the degree and the leading coefficients of auxiliary rational functions. Thus, for $0 \leq j \leq \hat{N}_{\max} - 1$, the point $\hat{Y}_j = (\alpha_1^{\hat{s}+j-1}, \alpha_2^{\hat{s}+j-1}, \cdots, \alpha_m^{\hat{s}+j-1})$

in Line 15 is random since $\hat{s} \in [0, q-2]$ is random and $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in (\mathbb{Z}_q \setminus \{0\})^m$ is picked at random in Line 13. Thus, if $\hat{Y}_j$ is a bad evaluation point then $\Delta(\hat{Y}_j) = 0$. Therefore, the probability that $\hat{Y}_j$ is a bad evaluation point for any $0 \leq j \leq \hat{N}_{\max} - 1$

$$\leq \frac{\hat{N}_{\max} \deg(\Delta)}{q-1} \leq \frac{2\hat{N}_{\max} nsd_{\max,1} D_{R_y}}{q-1}$$

where

$$\deg(\Delta, y) = \sum_{k=0}^{T-1} \deg(f_k) + \deg(g_k) = 2T \deg(R) \leq 2nsd_{\max,1} D_{R_y}.$$

Using Theorem 6.26 ((6.13) and (6.14)), it follows that

$$\Pr[\text{ basis shift } \beta \text{ picked at random in Line 10 is bad}] \leq \frac{\hat{N}_{\max} nsd_{\max,1} D_{R_y}^2}{q-1}$$

and

$$\Pr[\text{prime } q \text{ is unlucky for any } \ B_j \text{ where } 1 \leq j \leq \hat{N}_{\max}] \leq \frac{2\hat{N}_{\max} \log_{p_{\min}} (\|S\|_\infty nsd_{\max,1})}{N}.$$

Therefore, the probability that Algorithm 20 returns FAIL in Line 21 is at most

$$\frac{\hat{N}_{\max} nsd_{\max,1} D_{R_y}^2}{q-1} + \frac{2\hat{N}_{\max} nsd_{\max,1} D_{R_y}}{q-1} + \frac{2N_a \log_{p_{\min}} (\|S\|_\infty nsd_{\max,1})}{N}$$
$$\leq \frac{3\hat{N}_{\max} nsd_{\max,1} D_{R_y}^2}{q-1} + \frac{2N_a \log_{p_{\min}} (\|S\|_\infty nsd_{\max,1})}{N}.$$

□

Remark 6.33. The error probability for the rational number reconstruction process when applied on the coefficients of S to reconstruct its rational coefficients using one prime by Algorithm 19 or many subsequent primes in Algorithm 20 will not be accounted for, because Monagan's maximal quotient reconstruction algorithm [Monagan, 2004] is used in our implementation, and it will always succeed with a probability of one when the input prime p or product of the primes $p = \prod_{q \in P} q > 9h^2$ where $h = \max_{k=0}^{T-1} \left(\max_{\frac{n_{k_1}}{d_{k_1}} \in f_k \text{ and } \frac{n_{k_2}}{d_{k_2}} \in g_k} (|n_k d_k|) \right)$.

6.4 Main Results

Our main technical results are presented in this section.

Theorem 6.34. Suppose Algorithm 19 only needs one smooth prime p to interpolate the monic square-free factor S and suppose p is selected at random from the list of N_s smooth primes P_s where $p_{s_{\min}} = \min(P_s)$. Let N_a be the number of auxiliary rational functions

needed to interpolate the monic-square-free factor S. If all the degrees pre-computed in Lines 1-5 are correct then the probability that Algorithm 19 returns FAIL is at most

$$\frac{13N_a^2D_{R_y}^2nsd_{\max,1}\prod_{j=1}^{m}(1+nsD_{\max,j})}{p_{s_{\min}}-1}+\frac{6N_aD_{R_y}\log_{p_{s_{\min}}}(\|S\|_\infty\|R\|_\infty nsd_{\max,1})}{N_s}.$$

Proof. Clearly, the probability that Algorithm 19 returns FAIL is at most

$$\Pr[\text{Algorithm 19 returns FAIL in Lines 21 or 25 or 30 or 33}].$$

Since $e_{\max}=2+\max_{k=0}^{T-1}\{\deg(f_k)+\deg(g_k)\}\leq 4D_{R_y}$, the probability that Algorithm 20 returns FAIL in Lines 21 or 25 or 30 or 33 is at most

$$\leq \underbrace{\frac{8N_ansD_{R_y}^2d_{\max,1}}{p}+\frac{4N_aD_{R_y}\log_{p_{s_{\min}}}(\|S\|_\infty\|R\|_\infty nsd_{\max,1})}{N_s}}_{\text{by Lemma 6.16}}+$$
$$\underbrace{\frac{2N_a^2D_{R_y}nsd_{\max,1}\prod_{j=1}^{m}(1+nsD_{\max,j})}{p-1}}_{\text{by Theorem 6.30}}+$$
$$\underbrace{\frac{3N_aD_{R_y}^2nsd_{\max,1}\prod_{j=1}^{m}(1+nsD_{\max,j})}{p-1}+\frac{2N_a\log_{p_{s_{\min}}}(\|S\|_\infty nsd_{\max,1})}{N_s}}_{\text{by Lemma 6.26}}.$$

The rest of the argument follows by simplifying the above failure probability bound and using the fact that $p\geq p_{s_{\min}}$. □

Theorem 6.35. Suppose Algorithm 19 needs more than one prime p to interpolate the monic square-free factor S. Let q be a new prime selected at random from the list of primes P to be used by Algorithm 20 where $|P|=N$ and $p_{\min}=\min(P)$. Let N_a be the number of auxiliary rational functions needed to interpolate the monic-square-free factor S. Then the probability that Algorithm 20 returns FAIL is at most

$$\frac{6N_aD_{R_y}\log_{p_{\min}}(\|S\|_\infty\|R\|_\infty nsd_{\max,1})}{N}+\frac{13D_{R_y}^2N_a^2nsd_{\max,1}}{(p_{\min}-1)}.$$

Proof. Notice that Algorithm 20 will return FAIL, if it returns FAIL in Lines 17 or 24 or 27 or 34. Since

$$e_{\max}=2+\max_{k=0}^{T-1}\{\deg(f_k)+\deg(g_k)\}\leq 4D_{R_y},$$

and

$$N_a\geq\hat{N}_{\max}=\max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\},\max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\}),$$

the probability that Algorithm 20 returns FAIL in Lines 17 or 24 or 27 or 34 is at most

$$\leq \underbrace{\frac{8N_a nsD_{R_y}^2 d_{\max,1}}{q} + \frac{4N_a D_{R_y} \log_{p_{\min}}(\|S\|_\infty \|R\|_\infty nsd_{\max,1})}{N}}_{\text{by Lemma 6.16}} + \underbrace{\frac{2nsd_{\max,1}N_a^2 D_{R_y}^2}{(q-1)}}_{\text{by Lemma 6.31}} +$$

$$\underbrace{\frac{3N_a nsd_{\max,1}D_{R_y}^2}{q-1} + \frac{2N_a \log_{p_{\min}}(\|S\|_\infty nsd_{\max,1})}{N}}_{\text{by Lemma 6.32}}.$$

Our result follows by simplifying the above failure bounds using the fact that $q \geq p_{\min}$. □

Remark 6.36. The reader should notice that the difference between the failure probability bounds given in Theorems 6.34 and 6.35 is the quantity $\prod_{j=1}^{m}(1 + nsD_{\max,j})$ which arises because a Kronecker substitution was used for the first part of our Dixon resultant algorithm. Again, we note that the average value of this quantity is 1 [Schmidt, 2006, Chapter 4].

Lemma 6.37. The probability that Algorithm 20 returns FAIL is at most

$$\frac{6N_a D_{R_y} \log_{p_{s_{\min}}}(\|S\|_\infty \|R\|_\infty nsd_{\max,1})}{N_s} + \frac{13D_{R_y}^2 N_a^2 nsd_{\max,1}}{(p_{s_{\min}} - 1)}.$$

Proof. Use the fact that $N \geq N_s$ and $p_{\min} \geq p_{s_{\min}}$ in Theorem 6.35. □

Corollary 6.38. Suppose Algorithms 19 and 20 are modified to interpolate l monic square-free factors R_j of the Dixon resultant R. Let N_a be the number of auxiliary rational functions needed to interpolate all the monic square-free factors R_j. Suppose all the smooth primes needed by Algorithm 19 are selected from the list of smooth primes P_s such that $p_{\min} = \min(P)$ and $|P_s| = N_s$, and the primes (not necessarily smooth) needed by Algorithm 20 are selected at random from the list of primes P such that $p_{\min} = \min(P)$ and $p_{\min} \geq p_{s_{\min}}$.

(A) If Algorithm 19 only needs one smooth prime to interpolate all the monic square-free factors R_j then the probability that Algorithm 19 returns FAIL is at most

$$\frac{13N_a^2 D_{R_y}^2 nsd_{\max,1}\prod_{j=1}^{m}(1 + nsD_{\max,j})}{p_{s_{\min}} - 1} + \frac{6N_a D_{R_y} \log_{p_{s_{\min}}}(\|R_j\|_\infty \|R\|_\infty nsd_{\max,1})}{N_s}.$$

(B) Suppose Algorithm 19 needs more than one prime to interpolate all the monic square-free factors R_j. Then the probability that Algorithm 20 returns FAIL is at most

$$\frac{13D_{R_y}^2 N_a^2 nsd_{\max,1}}{(p_{s_{\min}} - 1)} + \frac{6N_a D_{R_y} \log_{p_{s_{\min}}}(\|R_j\|_\infty \|R\|_\infty nsd_{\max,1})}{N_s}.$$

Proof. Recall that the number of the rational function coefficients of S denoted by $T \leq nsd_{\max,1}$. Let T_j be the number of the rational function coefficients in R_j. So modifying Algorithm 19 and 20 to handle l monic square-free factors will not affect our failure probability bounds since

$$\left(\sum_{j=1}^{l} T_j\right) \leq \left(\sum_{j=1}^{l} d_{T_j}\right) \leq \sum_{j=1}^{l} \deg(R_j, x_1) \leq \deg(R, x_1) \leq nsd_{\max,1}.$$

□

We are now ready to give a failure probability bound for our probabilistic test algorithm (Algorithm 24) which verifies if a monic square-free factor is correct.

Theorem 6.39. Let N_a be the number of auxiliary rational functions needed to interpolate the monic-square-free factor S. Let q be a new prime chosen to be used by Algorithm 24 (Our probabilistic test) at random from the list of primes P such that $|P| = N$ and $p_{\min} = \min(P)$ in order to determine if the output of Dixon resultant algorithm is correct. We have

$$\Pr[\text{Our probabilistic test fails}] \leq \frac{\log_{p_{\min}}(2nsd_{\max,1}N_aD_{R_y}\|S\|_\infty)}{N}.$$

Proof. Our probabilistic test (Algorithm 24) will correctly verify that the output of our Dixon resultant algorithm is correct or wrong if the input prime q does not divide any integer coefficient in the monic square free factor $S = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1,\ldots,y_m)}{g_k(y_1,\ldots,y_m)} x_1^{d_k}$. Since

$$N_a \geq \max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\}),$$

and $f_k = \sum_{i=0}^{\deg(f_k)} f_{i,k}$ and $g_k = \sum_{j=0}^{\deg(g_k)} g_{j,k}$, we have that

$$\max(\#f_k, \#g_k) \leq N_a\,(\max(\deg(f_k), \deg(g_k)) \leq N_aD_{R_y}.$$

Thus, $\#S \leq T(\#f_k + \#g_k) \leq 2TN_aD_{R_y} \leq 2nsd_{\max,1}N_aD_{R_y}$. Therefore,

$$\Pr[q \text{ divides any integer coefficient of } S] \leq \frac{\log_{p_{\min}}(2nsd_{\max,1}N_aD_{R_y}\|S\|_\infty)}{N}$$

and our result follows. □

Proposition 6.40. Suppose Algorithm 19 only needs one smooth prime p to interpolate S and p is picked at random from the list of smooth primes P_s where $|P_s| = N_s$ and $p_{s_{\min}} = \min(P_s)$. Let $k \in \mathbb{Z}^+$. If the smallest prime $p_{s_{\min}}$ in P_s satisfies

$$p_{s_{\min}} > 2^{k+1}\left(13N_a^2D_{R_y}^2nsd_{\max,1}\prod_{j=1}^{m}(1 + nsD_{\max,j})\right) + 1,$$

and the number of smooth primes in P_s denoted by N_s satisfies

$$N_s > 2^{k+1}\left(6N_aD_{R_y}\log_{p_{s_{\min}}}(\|S\|_\infty\|R\|_\infty nsd_{\max,1})\right)$$

then $\Pr[\text{the returned answer } S \text{ is correct}] > \left(1-\frac{1}{2^k}\right)^2$.

Proof. We define the events involved as follows.

- Let A be the event that Algorithm 19 returns an answer S with only one smooth prime.
- Let B be the event that the probabilistic test (Algorithm 24) correctly verifies that the answer S is correct.

By Theorems 6.34 and 6.39, we have that

$$\Pr[A^{\complement}] \leq \frac{1}{2^k} \text{ and } \Pr[B^{\complement}|A] \leq \frac{1}{2^k}$$

since prime $q \geq p_{\min} \geq p_{s_{\min}}$. Thus,

$$\Pr[\text{the returned answer } S \text{ is correct}] = \Pr[A \cap B] = \Pr[A] \times \Pr[B|A] > \left(1-\frac{1}{2^k}\right)^2.$$

□

Proposition 6.41. Suppose additional primes are picked at random from the list of primes P and they are needed to interpolate the monic square-free factor S by Algorithm 19. Let $k \in \mathbb{Z}^+$. If the smallest prime $p_{s_{\min}} = \min(P_s)$ satisfies

$$p_{s_{\min}} > 2^{k+1}\left(13N_a^2D_{R_y}^2 nsd_{\max,1}\prod_{j=1}^{m}(1+nsD_{\max,j})\right)+1,$$

and the number of primes in the list of smooth primes P_s satisfies

$$N_s > 2^{k+1}\left(6N_aD_{R_y}\log_{p_{s_{\min}}}(\|S\|_\infty\|R\|_\infty nsd_{\max,1})\right),$$

then $\Pr[\text{the returned answer } S \text{ is correct}] > \left(1-\frac{1}{2^k}\right)^3$.

Proof. We define the events involved as follows.

- Let A be the event that Algorithm 19 returns a support of S in Line 39.
- Let B be the event that Algorithm 20 uses H additional primes from the list of primes P to produce more images of S using the support of S from Algorithm 19.

- Let C be the event that the probabilistic test (Algorithm 24) correctly verifies that the answer produced by Algorithm 20 is correct. So,

$$\Pr[S \text{ is correct}] = \Pr[C \cap B \cap A] = \Pr[C|B \cap A] \times \Pr[B \cap A] = \Pr[C|B \cap A] \times \Pr[B|A] \times \Pr[A].$$

Using Theorems 6.34 and 6.39, and Lemma 6.37, we get

$$\begin{aligned}
&\Pr[B^{\complement}|A] \leq \left(\frac{2}{2^{k+1}}\right)^H \leq \left(\frac{1}{2^k}\right)^H \leq \frac{1}{2^k} \\
&\Pr[A^{\complement}] \leq \frac{1}{2^k}, \text{ and} \\
&\Pr[C^{\complement} \,|B \cap A] \leq \frac{1}{2^k}.
\end{aligned}$$

Thus, we have that $\Pr[S \text{ is correct}] > \left(1 - \frac{1}{2^k}\right)^3$. □

Example 6.42 (Heron 2d system)**.** The following quantities, namely,

- $N_{\max} = 5, H = 2, T_{\max} = 3, m = 3, n = 3, s = 3, d_{max,1} = 1$, and $D_{\max} = 2$.

  are real data for the Heron2d parametric polynomial system (Example 1.42).

Using the bounds from Theorem 6.5 (since we do not know what R is), we determined that

- $\|R\|_\infty = 7.29 \times 10^{11}$ and $\|S\|_\infty = 4.676419456 \times 10^{62}$.

Taking $p_{s_{\min}} > 2^{61}$ and solving for k as in Proposition 6.40, we get $k = 29$ and $N_s = 2.226511046 \times 10^{13}$. Since the number of auxiliary rational functions N_a used in our experiment to solve this system is 16, using Theorems 6.34 and 6.39, it follows that

$$\Pr[\text{Algorithm 19 returns FAIL}] \leq 1.862645149 \times 10^{-9},$$

and

$$\Pr[\text{Our probabilistic test fails }] \leq 1.347399558 \times 10^{-13}.$$

We remark that one 62 bit smooth prime was enough to interpolate the only monic square-free factor of this system (See Table 5.4). Thus, we do not need to determine a failure probability bound for Algorithm 20 which involves using additional primes.

We present another parametric polynomial system which our Dixon resultant algorithm used more than a prime to interpolate its only monic square-free factor.

Example 6.43 (Geddes2 system in Appendix section A.3)**.** The following quantities, namely,

- $N_{\max} = 32, H = 6, T_{\max} = 24, m = 1, n = 4, s = 24, d_{max,1} = 2$, and $D_{\max} = 2$

are real data for the Geddes2 parametric polynomial system. Using the bounds obtained in Theorem 6.5, we calculate the heights bound for R and S to obtain

- $\|R\|_\infty = 8.017623607 \times 10^{291}$ and $\|S\|_\infty = 1.075796791 \times 10^{542}$.

Additionally, the number of auxiliary rational functions N_a used in our experiment to solve this system is 54. Taking $p_{\min} > 2^{61}$ and solving for k as in Proposition 6.40 yields $k = 17$ and $N_s = 5.870683423 \times 10^{11}$. Using Theorems 6.34, 6.35 and 6.39, we get

$$\Pr[\text{Algorithm 19 returns FAIL}] \leq 7.629394531 \times 10^{-6},$$

$$\Pr[\text{Algorithm 20 returns FAIL}] \leq 3.955982350 \times 10^{-6},$$

and

$$\Pr[\text{Our probabilistic test fails}] \leq 4.939799663 \times 10^{-11}.$$

The greatest probability of failure is highlighted in red which is Algorithm 19. We note that the bounds that we have obtained are huge but the actual failure probability is much less. In our implementation, two 62 bit primes were enough to solve this system (See Table 5.4).

6.5 Complexity Analysis

In this section, we estimate the cost of a black box probe for our Dixon resultant R and the total number of black box probes required by our Dixon resultant algorithm. We remark that no input primes caused our experiments to fail because the primes used in our experiments are 62 bit primes and the support of the first image $\hat{S}$ of S obtained using the first prime is always correct, i.e., $\text{supp}(\hat{S}) = \text{supp}(S)$.

6.5.1 The cost of a black box probe

Theorem 6.44. Let M be a $s \times s$ Dixon matrix such that $\#M_{ij} \leq T_{\max}$. Suppose the integer coefficients of M_{ij} are l base B digits long. That is, $\|M_{ij}\|_\infty \leq B^l$. Let prime p chosen at random from the list of primes P satisfy $B < p < 2B$. Let

$$\hat{D}_{\max} = \max(d_{\max,1}, \max_{k=1}^{m} D_{\max,k}).$$

A black box probe costs

$$O(\underbrace{s^2 l T_{\max}}_{\text{Modular reduction}} + \underbrace{s^2 n m T_{\max} \hat{D}_{\max}}_{\text{Matrix Evaluation}} + \underbrace{s^3}_{\text{Gaussian elimination}})$$

arithmetic operations in $\mathbb{Z}_p$.

Proof. Let the entries of the Dixon matrix M denoted by $M_{ij} = \sum_{k=1}^{T_{\max}} a_k M_{ij_k}(x_1, y_1, \ldots, y_m)$ where $M_{ij_k}(x_1, y_1, \ldots, y_m)$ are monomials in variables $x_1, y_1, \ldots, y_m$. The cost of performing the modular reduction $M_{ij} \bmod p$ is $O(lT_{\max})$. Thus the total cost of reducing $M \bmod p$

is $O(s^2 l T_{\max})$. Let $\alpha \in \mathbb{Z}_p^{m+1}$ be an evaluation point. The maximum partial degree of the entries of M is at most $\max(\deg(\Delta_{X_e}, x_1), \max_{k=1}^{m} \deg(\Delta_{X_e}, y_k)) \leq n\hat{D}_{\max}$. The number of multiplications done to compute a monomial evaluation $M_{ij_k}(\alpha)$ is $mn\hat{D}_{\max}$. For $1 \leq k \leq T_{\max}$, all monomial evaluations $M_{ij_k}(\alpha)$ are computed using $O(nmT_{\max}\hat{D}_{\max})$ multiplications and $T_{\max}$ multiplications for the product $a_k M_{ij_k}(\alpha)$. Hence the cost of evaluating Dixon matrix M is $O(s^2 nm\hat{D}_{\max}T_{\max})$. The cost of the determinant computation which is done by Gaussian elimination over $\mathbb{Z}_p$ is $O(s^3)$ arithmetic operations in $\mathbb{Z}_p$. Thus a black box probe costs $O(s^2 T_{\max} l + s^2 T_{\max} mn\hat{D}_{\max} + s^3)$.

□

6.5.2 The number of black box probes required by our algorithm

Theorem 6.45. Let $e_{\max} = 2 + \max_{k=0}^{T-1}\{\deg(f_k) + \deg(g_k)\}$ be the number of points needed to perform univariate rational interpolation and let $d_{x_1} = \deg(R, x_1)$. Let $\hat{N}_{\max} = \max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\})$ where $f_{i,k}, g_{i,k}, f_k, g_k$ from S are as defined in (5.2), (5.3). Let H be the number of primes needed by Algorithm NewPrime to reconstruct the coefficients of S using rational number reconstruction. The number of black box probes required by our algorithm is $O(Hd_{x_1}e_{max}\hat{N}_{\max})$.

Proof. We need $d_{x_1}+1$ probes to the black box **BB** to interpolate a monic univariate image of R in x_1. In order to interpolate an auxiliary rational function A_j in Line 24 of Algorithm 19, we need to use $e_{\max}$ coefficients from the monic polynomial images H in x_1 obtained in Line 20 of Algorithm 19.

The size of the supports $\#f_{i,k}$ and $\#g_{i,k}$ are unknown, and they will be discovered by the BMA (Line 1 of Subroutine BMStep) via some feedback polynomial $\lambda(z)$. In particular, for each $i, \#f_{i,k}$ is obtained when $\deg(\lambda_1, z) < \frac{\#f_{i,k}}{2}$ for some feedback polynomial $\lambda_1 \in \mathbb{Z}_p[z]$ with a probability of at least

$$1 - \frac{\#f_{i,k}(\#f_{i,k}+1)\deg(K_r(f_{i,k}))}{2(p-1)}.$$

However, by design, the size of the supports $\#f_{\deg(f_k),k}$ and $\#g_{\deg(f_k),k}$ are discovered first using $O(2\max\{\#f_{\deg(f_k),k}, g_{\deg(g_k),k}\})$ coefficients from the computed auxiliary rational functions. These coefficients from the A_j's may be enough to discover $\#f_{i,k}$ and $\#g_{i,k}$. But in most cases, they are not enough, so more auxiliary rational functions must be computed.

In the worst case, the maximum total number of auxiliary rational functions that need to be interpolated for the first prime is $O(4\hat{N}_{\max})$. Furthermore, using the support obtained from the first prime, $O(H\hat{N}_{\max})$ new auxiliary functions are needed if additional primes are required to solve for the unknown coefficients of the $f_{i,k}$'s and $g_{i,k}$'s. Therefore, the total number of black box probes required by our Dixon resultant algorithm is $O(Hd_{x_1}e_{max}\hat{N}_{\max})$.

□

6.5.3 Theoretical Comparison

Let R be the Dixon resultant. Let d be the maximum partial degree of R and let $t = \#R$. The following table (Table 6.1) compares our Dixon resultant algorithm with Zippel's sparse interpolation algorithm and the Ben-Or/Tiwari algorithm for interpolating R in terms of the number of black box probes required and the sizes of the primes needed for these algorithms.

Table 6.1: Comparing sparse algorithms in terms of # of probes and the size of their primes

	Number of black box probes	size of prime
Our new Dixon resultant algorithm	$O(Hd^2\hat{N}_{\max})$	$p > (d+1)^m$
Zippel's algorithm	$O(mdt + Ht)$	$p > 2md^2t^2$
Ben-Or/Tiwari algorithm	$O(Ht)$	$p > p_m^{(m+1)d}$

Chapter 7

Solving $Ax = b$

7.1 Summary of Contributions

Our main contribution in this chapter is a new black box algorithm for solving parametric linear systems which uses the new sparse rational function interpolation method developed in Chapter 4. We have implemented our algorithm in Maple + C and we compare our new algorithm with 4 other algorithms for solving parametric linear systems. The failure probability and complexity analysis for our new algorithm is presented. The results in this chapter have been published in the Proceedings of CASC '23 [Jinadu and Monagan, 2023].

7.2 Introduction

Let $Ax = b$ be a parametric linear system where the coefficient matrix $A \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^{n \times n}$ is of full rank n and the right hand side column vector $b \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^n$. Suppose that the number of terms in the entries of A and b denoted by $\#A_{ij}, \#b_i \leq t$ and the total degrees $\deg(A_{ij}), \deg(b_i) \leq d$. We present a new black box algorithm that uses our new sparse rational function interpolation method from Chapter 4 to interpolate the solution vector of rational functions

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T = \begin{bmatrix} \frac{f_1}{g_1} & \frac{f_2}{g_2} & \cdots & \frac{f_n}{g_n} \end{bmatrix}^T$$

such that for $f_k, g_k \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$, $g_k \neq 0$, $g_k | \det(A)$ and $\gcd(f_k, g_k) = 1$ for $1 \leq k \leq n$.

7.3 The Algorithm

Let p be a prime and let $\mathbf{BB} : \left(\mathbb{Z}_p^m, p\right) \to \mathbb{Z}_p^n$ with $m \geq 1$ and $n \geq 1$ be a black box for the augmented matrix $B = [A|b]$ denoting a parametric linear system $Ax = b$. That is, it takes a list of integers $\alpha \in \mathbb{Z}_p^m$ and a prime p as inputs and solves for

$$x(\alpha) = A^{-1}(\alpha)b(\alpha) \in \mathbb{Z}_p^n$$

using Gaussian elimination over $\mathbb{Z}_p$.

Let the polynomials f_k and g_k of the entries $x_k = \frac{f_k}{g_k}$ of the solution vector x be viewed as

$$f_k = \sum_{i=0}^{\deg(f)} f_{i,k}(y_1, y_2, \ldots, y_m) \text{ and } g_k = \sum_{j=0}^{\deg(g)} g_{j,k}(y_1, y_2, \ldots, y_m) \tag{7.1}$$

such that polynomials $f_{i,k}$ and $g_{j,k}$ are homogeneous polynomials and $\deg(f_{i,k}) = i$ and $\deg(g_{j,k}) = j$.

Given a black box **BB** for a parametric linear system $Ax = b$, we divide the steps to interpolate the unique vector x by our new black box algorithm (Algorithm 26) into seven main steps. The first step in our algorithm is to obtain the degrees needed to interpolate x. These include the total degrees $\deg(f_k), \deg(g_k)$ for $1 \leq k \leq n$, which are needed to densely interpolate the univariate auxiliary rational functions, the maximum partial degrees $\max_{k=1}^{n} (\max(\deg(f_k, y_i), \deg(g_k, y_i)))$ for $1 \leq i \leq m$, which are needed to apply Kronecker substitution and the total degrees of the polynomials $f_{i,k}$ and $g_{i,k}$ which helps avoid doing unnecessary work when the effect of the basis shift is removed in Subroutine 18 (See Lines 1-5 of Algorithm 26).

With high probability, we describe how to discover these degrees as follows. Let p be a large prime. First, pick $\alpha, \beta \in (\mathbb{Z}_p \setminus \{0\})^m$ at random, and use enough distinct points for z selected at random from $\mathbb{Z}_p$ to interpolate the univariate rational function

$$h_k(z) = \frac{N_k(z)}{D_k(z)} = \frac{f_k(\alpha_1 z + \beta_1, \ldots, \alpha_m z + \beta_m)}{g_k(\alpha_1 z + \beta_1, \ldots, \alpha_m z + \beta_m)} \in \mathbb{Z}_p(z),$$

via probes to the black box so that $\deg(f_k) = \deg(N_k)$ and $\deg(g_k) = \deg(D_k)$ with high probability. Next, pick $\gamma \in (\mathbb{Z}_p \setminus \{0\})^{m-1}$, $\theta \in \mathbb{Z}_p \setminus \{0\}$ at random and probe the black box to interpolate the univariate rational function

$$H_i(z) := \frac{H_{f_i}}{H_{g_i}} = \frac{f_k(\gamma_1, \ldots, \gamma_{i-1}, \theta z, \gamma_{i+1}, \cdots, \gamma_m)}{g_k(\gamma_1, \ldots, \gamma_{i-1}, \theta z, \gamma_{i+1}, \cdots, \gamma_m)} \in \mathbb{Z}_p(z),$$

using enough distinct random points for z from $\mathbb{Z}_p$. With high probability $\deg(H_{f_i}, z) = \deg(f_k, y_i)$ and $\deg(H_{g_i}, z) = \deg(g_k, y_i)$ for $1 \leq i \leq m$.

Finally, suppose we have obtained $\deg(f_k), \deg(g_k)$ correctly for $1 \leq k \leq n$. Then pick $\alpha \in (\mathbb{Z}_p \setminus \{0\})^m$ at random and use enough random distinct points for z selected from $\mathbb{Z}_p \setminus \{0\}$ to interpolate the univariate rational function

$$W_k(z) = \frac{\overline{N}_k}{\overline{D}_k} = \frac{\sum_{j=0}^{d_{f_k}} \bar{N}_{i,k}(z)}{\sum_{i=0}^{d_{g_k}} \bar{D}_{i,k}(z)} = \frac{f_k(\alpha_1 z, \ldots, \alpha_m z)}{g_k(\alpha_1 z, \ldots, \alpha_m z)} \in \mathbb{Z}_p(z)$$

where $d_{f_k} = \deg(\overline{N}_k)$ and $d_{g_k} = \deg(\overline{D}_k)$. Now, if $\deg(f_k) = d_{f_k}$ and $\deg(g_k) = d_{g_k}$ then $\deg(f_{i,k}) = \deg(\bar{N}_{i,k})$ and $\deg(g_{i,k}) = \deg(\bar{D}_{i,k})$ with high probability. But, if there

is no constant term in f_k or g_k, then $\deg(f_k) \neq d_{f_k}$ and $\deg(g_k) \neq d_{g_k}$ because $e_k = \deg(\gcd(\overline{N}_k, \overline{D}_k)) > 0$. Since we do not know what e_k is, it follows that if $e_k = \deg(f_k) - d_{f_k} = \deg(g_k) - d_{g_k}$ with high probability then $\deg(f_{i,k}) = \deg(\bar{N}_{j,k}) + e_k$ and $\deg(g_{i,k}) = \deg(\bar{D}_{i,k}) + e_k$ with high probability.

After obtaining all the degree bounds, the second step in our algorithm is to probe the black box **BB** with input evaluation points $\alpha \in \mathbb{Z}_p^m$ to obtain images $x(\alpha) = A^{-1}(\alpha)b(\alpha) \in \mathbb{Z}_p^n$ (See Lines 17-19).

The third step is to perform dense interpolation of auxiliary univariate rational functions labelled as $A_j(z)$ using the images $x(\alpha) = A^{-1}(\alpha)b(\alpha) \in \mathbb{Z}_p^n$ (See Lines 23-27). By design, the fourth step is to determine the number of terms in the leading term polynomials $f_{\deg(f_k),k}$ and $g_{\deg(f_k),k}$ and interpolate them via calls to Subroutine BMStep in Lines 29-30.

Next, $\#f_{i,k}$ and $\#g_{i,k}$ as defined in (7.1) are determined by calls to Subroutine RemoveShift in Lines 33-34 where the effect of the basis shift $\beta \neq 0$ is removed and the coefficients of the auxiliary rational functions $A_j(z)$ are adjusted in order to interpolate $f_{i,k}$ and $g_{i,k}$. Note that for each $i, \#f_{i,k}$ (or $\#g_{i,k}$) is obtained when $\deg(\lambda, z) < \frac{\#f_{i,k}}{2}$ for some feedback polynomial $\lambda \in \mathbb{Z}_p[z]$ generated by the Berlekamp-Massey algorithm in Line 1 of Subroutine BMStep.

Once $f_{i,k}, g_{i,k}$ modulo a prime have been interpolated, the sixth step in our algorithm is to apply rational number reconstruction (RNR) on the assembled vector $\overline{X} = [\frac{f_k}{g_k} \bmod p, 1 \leq k \leq n]$ to get x in Line 41. If RNR process fails, then more primes and images of x are needed to interpolate the vector x using Chinese remaindering and rational number reconstruction.

The final step is to call Algorithm 27, an algorithm similar to Algorithm 26, except that $\#f_{i,k}$ and $g_{i,k}$ are now known, and Algorithm 27 uses more primes to get the solution x using Chinese remaindering and rational number reconstruction.

7.4 Analysis

7.4.1 Failure Probability Analysis

For the rest of this section, let

1. $d = \max_{i,j=1}^{n}(\deg(b_j), \deg(A_{ij}), \deg(f_i), \deg(g_i))$,
2. $t = \max_{i,j=1}^{n}(\#A_{ij}, \#b_j, \#f_i, \#g_i)$
3. $\|A_{ij}\|_\infty, \|b_j\|_\infty \leq h$.
4. $P_s = \{p_{s_1}, p_{s_2}, \ldots, p_{N_s}\}$ be the list of smooth primes to be used in Algorithm 26 such that $p_{s_{\min}} = \min_{i=1}^{N_s}\{p_i\}$, $|P_s| = N_s$ and N_s is a large positive integer.
5. $P = \{p_1, p_2, \ldots, p_N\}$ be the list of primes (not necessarily smooth) to be used in Algorithm 27 such that $p_{\min} = \min_{i=1}^{N}\{p_i\}$, $|P| = N$ and N is a large positive integer with $N \geq N_s$ and $p_{\min} \geq p_{s,\min}$.

Algorithm 26: ParamLinSolve

Input: The black box $\mathbf{BB} : (\mathbb{Z}_p^m, p) \to \mathbb{Z}_p^n$ for the augmented system $[A|b]$ with $m \geq 1$.
Output: The vector $x \in \mathbb{Z}(y_1, \ldots, y_m)^n$ such that $Ax = b$ or **FAIL**.

Compute total degrees $(\deg(f_k), \deg(g_k))$ for $1 \leq k \leq n$.
$e_k \leftarrow \deg(f_k) + \deg(g_k) + 2$ for $1 \leq k \leq n$.
$e_{\max} \leftarrow \max_{k=1}^{n} \{e_k\}$.
Compute (E_{f_k}, E_{g_k}) where E_{f_k} and E_{g_k} denote the lists of the total degrees of the polynomials f_{ik} and g_{ik} in f_k and g_k respectively as defined in (7.1).
$D_{y_i} \leftarrow \max\left(\max_{k=1}^{n}(\deg(f_k, y_i), \deg(g_k, y_i))\right)$ for $1 \leq i \leq m$.
Initialize $r_i = D_{y_i} + 1$ for $1 \leq i \leq m$ and let $r = (r_1, r_2, \ldots, r_{m-1})$.
Pick a random smooth prime $p > \prod_{j=1}^{m} r_i$ and a random basis shift $\beta \neq 0 \in \mathbb{Z}_p^m$.
// p is the prime to be used by the black box.
Let $K_r : \mathbb{Z}_p(y_1, y_2, \ldots, y_m) \to \mathbb{Z}_p(y)$ be the Kronecker substitution $K_r(f_k/g_k)$.
Pick a random shift $\hat{s} \in [0, p-2]$ and any generator α for $\mathbb{Z}_p^*$.
Let z be the homogenizing variable.
Pick $\theta \in \mathbb{Z}_p^{e_{\max}}$ at random with $\theta_i \neq \theta_j$ for $i \neq j$.
$M \leftarrow \prod_{i=1}^{e_{\max}}(z - \theta_i) \in \mathbb{Z}_p[z]$; ..... $O(e_{\max}^2)$
$k \leftarrow 1$.
for $i = 1, 2, \cdots$ **while** $k \leq n$ **do**
    $\hat{Y}_i \leftarrow (\alpha^{\hat{s}+i-1}, \alpha^{(\hat{s}+i-1)r_1}, \ldots, \alpha^{(\hat{s}+i-1)(r_1 r_2 \cdots r_{m-1})})$. // Apply the Kronecker substitution K_r here
    for $j = 1, 2, \ldots, e_{max}$ **do**
        $Z_j \leftarrow \theta_j \cdot \hat{Y}_i + \beta \in \mathbb{Z}_p^m$;
        $v_j \leftarrow \mathbf{BB}(Z_j, p)$ // Here $v_j = A^{-1}(Z_j)b(Z_j) \in \mathbb{Z}_p^n$
        if $v_j =$ **FAIL then** return **FAIL end** // $\operatorname{rank}(A(Z_j)) < n$.
    end
    if $i \notin \{2, 4, 8, 16, 32, \cdots\}$ **then** next **end**
    for $j = 1, 2, \ldots, i$ **do**
        Interpolate $U \in \mathbb{Z}_p[z]$ using points $(\theta_i, v_{kj} : 1 \leq j \leq e_k)$; ..... $O(e_k^2)$
        $A_j(z) \leftarrow \text{MQRFR}(M, U, p)$; ..... $O(e_k^2)$
        Let $A_j(z) = \frac{N_j(z)}{\hat{N}_j(z)} \in \mathbb{Z}_p(z)$ // This is the auxiliary function in z.
        if $\deg(N_j) \neq \deg(f_k)$ or $\deg(\hat{N}_j) \neq \deg(g_k)$ return **FAIL end**
        Normalize $A_j(z)$ such that $\hat{N}_j(z) = 1 + \sum_{i=1}^{\deg(\hat{N})} a_i z^i$.
    end
    $F_k \leftarrow \text{BMStep}([\text{coeff}(N_j, z^{\deg(f_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r)$; ..... $O(i^2 + \#F_k^2 \log p)$
    $G_k \leftarrow \text{BMStep}([\text{coeff}(\hat{N}_j, z^{\deg(g_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r)$; ..... $O(i^2 + \#G_k^2 \log p)$
    // Here $F_k = f_{\deg(f_k),k} \bmod p$ and $G_k = g_{\deg(g_k),k} \bmod p$
    if $F_k \neq$ **FAIL** *and* $G_k \neq$ **FAIL then**
        $f_k \leftarrow \text{RemoveShift}(F_k, \beta, E_{f_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [N_1, \ldots, N_i], r)$
        $g_k \leftarrow \text{RemoveShift}(G_k, \beta, E_{g_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [\hat{N}_1, \ldots, \hat{N}_i], r)$
        if $f_k \neq$ **FAIL** *and* $g_k \neq$ **FAIL then**
            $k \leftarrow k + 1$ // we have interpolated $x_k \bmod p$
        end
    end
end
$\overline{X} \leftarrow [\frac{f_k}{g_k}, 1 \leq k \leq n]$ // Here $\overline{X} = x \bmod p$
Apply rational number reconstruction on the coefficients of $\overline{X} \bmod p$ to get x
if $x \neq$ FAIL **then** clear the fractions of x and return x **end**
return MorePrimes($\mathbf{BB}, \overline{X}, ((\deg(f_k), \deg(g_k)) : 1 \leq k \leq n), p$)

Algorithm 27: MorePrimes

Input: The Black box $\mathbf{BB} : \left(\mathbb{Z}_q^m, q\right) \to \mathbb{Z}_q^n$ for $[A|b]$ with $m \geq 1$, a vector $\bar{X} = x \bmod p$, where p is the smooth prime used by Algorithm 26 and the list of degrees $[(\deg(f_k), \deg(g_k)) : 1 \leq k \leq n]$.

Output: The vector $x \in \mathbb{Z}(y_1, \ldots, y_m)^n$ such that $Ax = b$ or **FAIL**.

Let $e_k = \deg(f_k) + \deg(g_k) + 2$ for $1 \leq k \leq n$ and let $e_{\max} = \max_{k=1}^{n}(e_k)$.

Let $B_1 = [f_{\deg(f_k)-1,k}, \ldots, f_{0,k}]$ and $B_2 = [g_{\deg(g_k)-1,k}, \ldots, g_{0,k}]$ where $f_{i,k}, g_{i,k}$ are as defined in (7.1)

Let $N_{\max} = \max_{k=1}^{n} \left\{ \max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\} \right\}$ and set $P = p$.

do

  Get a new prime $q \nmid P$. // The black box **BB** uses a new prime $q \neq p$.

  Pick $\alpha, \beta \in (\mathbb{Z}_q \setminus \{0\})^m, \theta \in \mathbb{Z}_q^{e_{\max}}$ and shift $\hat{s} \in [0, q-2]$ at random.

  for $i = 1, 2, \ldots, N_{\max}$ **do**

    $\hat{Y}_i \leftarrow (\alpha_1^{\hat{s}+i-1}, \alpha_2^{\hat{s}+i-1} \cdots, \alpha_m^{\hat{s}+i-1})$.

    for $j = 1, 2, \ldots, e_{max}$ **do**

      $Z_j \leftarrow \hat{Y}_i \cdot \theta_j + \beta \in \mathbb{Z}_p^m$

      $v_j \leftarrow \mathbf{BB}\,(Z_j, q)$ // Here $v_j = A^{-1}(Z_j)b(Z_j) \in \mathbb{Z}_p^n$

      if $v_j =$ **FAIL then** return **FAIL end** // $\operatorname{rank}(A(Z_j)) < n$).

    end

  end

  for $k = 1, 2, \ldots, n$ **do**

    $(\hat{n}, \hat{M}) \leftarrow (\#f_{\deg(f_k),k}, \operatorname{supp}(f_{\deg(f_k),k}))$ // supp means support.

    $(\bar{n}, \bar{M}) \leftarrow (\#g_{\deg(g_k),k}, \operatorname{supp}(g_{\deg(g_k),k}))$

    $(\hat{m}, \bar{m}) \leftarrow ([\hat{M}_i(\alpha) : 1 \leq i \leq \hat{n}], [\bar{M}_i(\alpha) : 1 \leq i \leq \bar{n}])$; ..... $O(m(\hat{n} + \bar{n}))$

    if the monomial evaluations $\hat{m}_i = \hat{m}_j$ or $\bar{m}_i = \bar{m}_j$ then return **FAIL end.**

    $M \leftarrow \prod_{i=1}^{e_k}(z - \theta_i) \in \mathbb{Z}_q[z]$; ..... $O(e_k^2)$

    for $j = 1, 2, \ldots, N_{\max}$ **do**

      Interpolate $u \in \mathbb{Z}_p[z]$ using points $(\theta_i, v_{kj} : 1 \leq j \leq e_k)$; ..... $O(e_k^2)$

      $B_j \leftarrow \text{MQRFR}(M, u, p)$//$B_j = N_j(z)/\hat{N}_j(z) \in \mathbb{Z}_q(z)$..... $O(e_k^2)$

      Normalize $B_j(z)$ s.t. $\hat{N}_j(z) = 1 + \sum_{i=1}^{\deg(\hat{N})} b_i z^i$.

      if $\deg(N_j) \neq \deg(f_k)$ or $\deg(\hat{N}_j) \neq \deg(g_k)$ then return **FAIL end.**

    end

    Let $a_i = \text{LC}(N_j, z)$ and let $b_i = \text{LC}(\hat{N}_j, z)$ for $1 \leq i \leq N_{\max}$.

    $F_k \leftarrow$VandermondeSolver$(\hat{m}, [a_1, \ldots, a_{\hat{n}}], \hat{s}, \hat{M})$; ..... $O(\hat{n}^2)$

    $G_k \leftarrow$VandermondeSolver$(\bar{m}, [b_1, \ldots, b_{\bar{n}}], \hat{s}, \bar{M})$; ..... $O(\bar{n}^2)$

    $F_k \leftarrow \text{GetTerms}(F_k, \alpha, \beta, \hat{s}, B_1, \hat{Y}_1, \ldots, \hat{Y}_{N_{\max}}], [N_1, \ldots, N_{N_{\max}}], q)$

    $G_k \leftarrow \text{GetTerms}(G_k, \alpha, \beta, \hat{s}, B_2, [\hat{Y}_1, \ldots, \hat{Y}_{N_{\max}}], [\hat{N}_1, \ldots, \hat{N}_{N_{\max}}], q)$

    if $F_k =$ **FAIL** or $G_k =$ **FAIL** then return **FAIL end**

  end

  $\hat{X} \leftarrow [\frac{F_k}{G_k}, 1 \leq k \leq n]$ // Here $\hat{X} = x \bmod q$

  Solve $\{\hat{F} \equiv \overline{X} \bmod P$ and $\hat{F} \equiv \hat{X} \bmod q\}$ using Chinese remaindering and set $P = P \times q$.

  Apply rational number reconstruction on coefficients of $\hat{F}$ mod P to get x

  if $x \neq$ **FAIL then** return $\overline{F}$ **else** $(\overline{X}, p) \leftarrow (\hat{F}, q)$ **end**

end

We now estimate the height of the entries x_k of the solution vector x.

Theorem 7.1. We have

$$\|x_k\|_\infty \ \leq \mathrm{e}^{nmd} n^{\frac{n}{2}} t^n h^n$$

where e is the Euler number and $e \approx 2.718$.

Proof. Let $R_k = \det(A^k)$ where A^k denotes the matrix obtained by replacing the k-th column of the coefficient matrix A by the right hand side column vector b and let $R = \det(A)$. By Cramer's rule, the solutions of $Ax = b$ are given by

$$x_k = \frac{R_k}{R}.$$

Let $h_k = \gcd(R_k, R)$. Observe that

$$\frac{R_k/h_k}{R/h_k} = \frac{f_k}{g_k} = x_k$$

where $\gcd(f_k, g_k) = 1$. Therefore $f_k | R_k$ and $g_k | R$. By Lemma 6.3, it follows that

$$\|g_k\|_\infty \ \leq \mathrm{e}^{\sum_{i=1}^{m} \deg(R, y_i)} \|R\|_\infty \leq \mathrm{e}^{\sum_{i=1}^{m} nd} \|R\|_\infty \leq e^{nmd} \|R\|_\infty \tag{7.2}$$

and similarly,

$$\|f_k\|_\infty \ \leq \mathrm{e}^{nmd} \|R_k\|_\infty \tag{7.3}$$

because $\deg(R, y_i) \leq \deg(R) \leq n \times \max_{i=1}^{n} \{\deg(A_{ij})\} \leq nd$. Therefore

$$\|x_k\|_\infty \ \leq \max\left(\|f_k\|_\infty, \|g_k\|_\infty\right) \leq \mathrm{e}^{nmd} \max\left(\|R_k\|_\infty, \|R\|_\infty\right) \leq \mathrm{e}^{nmd} n^{\frac{n}{2}} t^n h^n$$

by Theorem 6.2. □

We remark that the above bound for the height of x_k is a worst case bound. Usually, $\|f_k\|_\infty \leq \|R_k\|_\infty$ and $\|g_k\|_\infty \leq \|R\|_\infty$.

7.4.2 Unlucky Primes and Evaluation Points

Let A be our input $n \times n$ coefficient matrix where $A_{i,j} \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$.

Definition 7.2. A prime p is said to be unlucky if $p | \det(A)$.

Definition 7.3. Suppose p is not an unlucky prime. Let $\alpha \in \mathbb{Z}_p^m$ be an evaluation point. We say that α is unlucky if $\det(A)(\alpha) = 0$.

Lemma 7.4. Let p be a prime chosen at random from the list of primes P such that $p_{\min} = \min(P)$ and $|P| = N$. Then

$$\Pr[p \text{ is unlucky}] \leq \frac{\log_{p_{\min}} \left(n^{\frac{n}{2}} t^n h^n\right)}{N}.$$

Proof. Let $R = \det(A)$ and let c be an integer coefficient of R. The number of primes p from P that can divide c is at most $\lfloor \log_{p_{\min}} c \rfloor$. So

$$\Pr[\, p \text{ divides } c \,] \leq \frac{\log_{p_{\min}} c}{N}.$$

By definition, prime p is unlucky $\iff p|R \implies p$ divides one term in R. So

$$\Pr[p \text{ is unlucky}] = \Pr[p \text{ divides } R] \leq \Pr[p \text{ divides one term in } R] \leq \frac{\log_{p_{\min}} \|R\|_\infty}{N}.$$

Using Theorem 6.2, it follows that $\Pr[p \text{ is unlucky}] \leq \dfrac{\log_{p_{\min}} \left(n^{\frac{n}{2}} t^n h^n \right)}{N}$. □

Lemma 7.5. Let p be a prime chosen at random from the list of primes P. Let $\alpha \in \mathbb{Z}_p^m$ be an evaluation point. If p is not an unlucky prime then

$$\Pr[\ \alpha \text{ is unlucky }] \leq \frac{nd}{p}.$$

Proof. Using Lemma 2.17, we have

$$\Pr[\ \alpha \text{ is unlucky }] = \Pr[\ \det(A)(\alpha) = 0\ ] \leq \frac{\deg(\det(A))}{p} \leq \frac{nd}{p}.$$

□

7.4.3 Bad Evaluation Points, Primes and Basis Shifts

Definition 7.6. We say that $\alpha \in \mathbb{Z}_p \setminus \{0\}$ is a **bad evaluation point** if $\deg(f_k^\beta(\alpha, z)) < \deg(f_k, z)$ or $\deg(g_k^\beta(\alpha, z)) < \deg(g_k, z)$ for any k.

Definition 7.7. We say that $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ is a **bad basis shift** if $gcd(f_k, g_k) = 1$ but $\deg(\gcd(f_k^\beta(\alpha, z), g_k^\beta(\alpha, z))) > 0$ for any k.

Definition 7.8. We say a prime p is **bad** if $p|\mathrm{LC}(f_k^\beta(y, z))$ in z or $p|\mathrm{LC}(g_k^\beta(y, z))$ in z for any k.

To avoid the occurrence of bad evaluation points with high probability in Algorithm 26, we had to interpolate $F_k(\alpha^{\hat{s}+i}, z, \beta)$ for some random point $\hat{s} \in [0, p-2]$ instead of $F_k(\alpha^i, z, \beta)$ for $i = 0, 1, 2, \ldots$. This is labelled as A_j in Line 25. Line 26 detects the occurrence of bad evaluation points, a bad basis shift or a bad prime.

7.4.4 Main Results

Theorem 7.9. Let N_a be greater than the required number of auxiliary rational functions needed to interpolate the vector x and suppose all the degree bounds obtained in Lines

1-5 of Algorithm 26 are correct. Suppose Algorithm 26 only needs one smooth prime to interpolate x and let e be the Euler number where $\mathrm{e} \approx 2.718$. If a smooth prime p is chosen at random from the list of smooth primes P_s such that $p_{s_{\min}} = \min(P_s)$ then the probability that Algorithm 26 returns FAIL is at most

$$\frac{6N_a n^2 d\left(\log_{p_{s_{\min}}}(th\sqrt{n})\right) + 2N_a n^2 md \log_{p_{s_{\min}}}(\mathrm{e})}{N_s} + \frac{2n(1+d)^m\left(N_a + t^2 + t^2 d\right) + 5n^2 N_a d^2}{p-1}.$$

Proof. Recall that $e_{\max} = \max_{k=1}^{n}\{\deg(f_k) + \deg(g_k) + 2\} \leq 4d$. Notice that

$$\Pr[v_j = \text{FAIL in Line 19}] = \Pr[\text{prime } p \text{ or evaluation point } Z_j \text{ in Line 17 is unlucky}].$$

By Lemma 7.4 and 7.5, we have that $\Pr[\text{Algorithm 26 returns FAIL in Line 19}] \leq$

$$e_{\max} n N_a \left(\frac{nd}{p} + \frac{\log_{p_{s_{\min}}}\left(n^{\frac{n}{2}} t^n h^n\right)}{N_s}\right) \leq 4n^2 d N_a \left(\frac{d}{p} + \frac{\log_{p_{s_{\min}}}(th\sqrt{n})}{N_s}\right) \tag{7.4}$$

There are three causes of FAIL in Line 26 of Algorithm 26. Again, all three failure causes (bad evaluation point, bad basis shift and bad prime) are a direct consequence of our attempt to interpolate auxiliary rational functions A_j in Line 25. We will handle the bad evaluation point case first. Let

$$\Delta(y) = \prod_{k=1}^{n} \mathrm{LC}(f_k^{\beta}(y,z))\mathrm{LC}(g_k^{\beta}(y,z)) \in \mathbb{Z}_p[y].$$

Notice that the evaluation point $\alpha^{\hat{s}+j-1}$ in Line 15 is random on $[0,p)$ since $\hat{s} \in [0, p-2]$ is random. Since a basis shift β does not affect the degree and the leading coefficients of auxiliary rational functions, we have that if $\alpha^{\hat{s}+j-1}$ is a bad then $\Delta(\alpha^{\hat{s}+j-1}) = 0$. Thus

$$\Pr[\alpha^{\hat{s}+j-1} \text{ is a bad for } 0 \leq j \leq N_a - 1] \leq \frac{N_a \deg(\Delta)}{p-1} \leq \frac{2N_a n(1+d)^m}{p-1}.$$

Now suppose $\theta_j := \alpha^{\hat{s}+j-1}$ is not bad for $1 \leq j \leq N_a$. Let $w_1, w_2, \cdots w_m$ be new variables and let

$$G_{kj} = \frac{\hat{f}_{k_j}}{\hat{g}_{k_j}} = \frac{f_k(\theta_j z + w_1, \ldots, z\theta_j^{(r_1 r_2 \cdots r_{m-1})} + w_m)}{g_k(\theta_j z + w_1, \ldots, z\theta_j^{(r_1 r_2 \cdots r_{m-1})} + w_m)} \in \mathbb{Z}_p(w_1, w_2, \ldots, w_m)(z).$$

Recall that $\mathrm{LC}(\hat{f}_{k_j})(\beta) \neq 0$ and $\mathrm{LC}(\hat{g}_{k_j})(\beta) \neq 0$. Let $\overline{R}_{kj} = \mathrm{res}(\hat{f}_{k_j}, \hat{g}_{k_j}, z) \in \mathbb{Z}_p[w_1, w_2, \ldots, w_m]$ be the Sylvester resultant and let $\Delta(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{N_a}\prod_{k=1}^{n} \overline{R}_{kj}$. Clearly, β picked at random in Line 7 is a bad basis shift $\iff \deg(\gcd(\hat{f}_{k_j}(z,\beta), \hat{g}_{k_j}(z,\beta)) > 0 \iff \Delta(\beta) = 0$ for

any k and j. Using Bezout's bound [Hu and Monagan, 2016, Lemma 4], we have

$$\deg(\overline{R}_{kj}) \leq \deg(f_k)\deg(g_k) \leq d^2.$$

Thus

$$\Pr[\beta \text{ is a bad basis shift}] = \Pr[\Delta(\beta) = 0] \leq \frac{\deg(\Delta)}{p-1} \leq \frac{nd^2N_a}{p-1}.$$

Finally, we deal with the bad prime case. The probability that p is bad is at most

$$\Pr[p \text{ divides one term of LC}(f_k) \text{ or LC}(g_k) \text{ for } 1 \leq k \leq n] \leq \frac{n\log_{p_{s_{\min}}}(\|f_k\|_\infty\|g_k\|_\infty)}{N_s}.$$

Using (7.2) and (7.3), we have $\Pr[\text{prime } p \text{ is bad for any } \ 1 \leq j \leq N_a]$

$$\leq \frac{N_a n\log_{p_{s_{\min}}}(\mathrm{e}^{nmd}n^{\frac{n}{2}}t^nh^n)^2}{N_s} \leq \frac{2N_an^2\left(\log_{p_{s_{\min}}}(th\sqrt{n}) + md\log_{p_{s_{\min}}}\mathrm{e}\right)}{N_s}.$$

Thus $\Pr[\text{Algorithm 26 returns FAIL in Line 26}]$ is at most

$$\frac{2N_an^2\left(\log_{p_{s_{\min}}}(th\sqrt{n}) + md\log_{p_{s_{\min}}}\mathrm{e}\right)}{N_s} + \frac{2N_an(1+d)^m}{p-1} + \frac{nd^2N_a}{p-1}. \tag{7.5}$$

Since N_a is greater than the required number of auxiliary rational function needed by Algorithm 26 to interpolate x, then Line 2 of Subroutine 17 will never return FAIL. However the feedback polynomial $\lambda \in \mathbb{Z}_p[z]$ generated to find the number of terms in $f_{i,k}$ or $g_{i,k}$ in Line 4 of Subroutine 17 might be wrong so it will return FAIL which causes Algorithm 26 to return FAIL in either Lines 29 or 30 or 33 or 34. By Theorem 6.28, we have that $\Pr[\text{ getting the wrong } \#f_{i,k} \ \text{ or } \ \#g_{i,k}]$ is at most

$$\frac{\sum_{k=1}^{n}\left(\sum_{i=0}^{\deg(f_k)}\#f_{i,k}(\#f_{i,k}+1)\deg(K_r(f_{i,k})) + \sum_{i=0}^{\deg(g_k)}\#g_{i,k}(\#g_{i,k}+1)\deg(K_r(g_{i,k})\right)}{2(p-1)}.$$

We have

$$\Pr[\text{Algorithm 19 returns FAIL in Lines 29 or 30 or 33 or 34}] \leq \frac{2nt^2(1+d)^{m+1}}{p-1}. \tag{7.6}$$

since $\#f_{i,k}, \#g_{i,k} \leq t, \deg(f_k), \deg(g_k) \leq d$ and $\deg(K_r(f_{i,k})), \deg(K_r(g_{i,k})) \leq (1+d)^m$. Our result follows by adding (7.4), (7.5) and (7.6). □

Theorem 7.10. Let N_a be greater than the required number of auxiliary rational functions needed to interpolate x. Let q be a new prime selected at random from the list of primes P to reconstruct the coefficients of x using rational number reconstruction such that $|P| = N$ and $p_{\min} = \min(P)$. Let $e \approx 2.718$ be the Euler number. Then $\Pr[\text{Algorithm 27 returns FAIL}]$

is at most

$$\frac{6N_an^2d\left(\log_{p_{\min}}(th\sqrt{n})\right)+2N_an^2md\log_{p_{\min}}(\mathrm{e})}{N}+\frac{7n^2d^2N_a+4nd^2t^2}{q-1}.$$

Proof. Using (7.4), the probability that Algorithm 27 returns FAIL in Line 12 is at most

$$4n^2dN_a\left(\frac{d}{q}+\frac{\log_{p_{\min}}(th\sqrt{n})}{N}\right). \tag{7.7}$$

If the monomial evaluations obtained in Line 19 of Algorithm 27 or the monomial evaluations obtained in Line 24 of Subroutine 21 are not distinct then

$$\begin{aligned}&\Pr[\text{Algorithm 27 returns FAIL in Line 19 or 30 or 31}]\\ &\leq\sum_{k=1}^{n}\frac{(\sum_{i=0}^{\deg(f_k)}\binom{\#f_{i,k}}{2}\deg(f_{i,k})+\sum_{i=0}^{\deg(g_k)}\binom{\#g_{i,k}}{2}\deg(g_{i,k}))}{q-1}\leq\frac{4nd^2t^2}{q-1}.\end{aligned} \tag{7.8}$$

Notice that the functions B_j obtained in Line 23 are of the form

$$\frac{f_k^{\beta}(y_1,y_2,\ldots,y_m,z)}{g_k^{\beta}(y_1,y_2,\ldots,y_m,z)}=\frac{f_k(y_1z+\beta_1,\ldots,y_mz+\beta_m)}{g_k(y_1z+\beta_1,\ldots,y_mz+\beta_m)},$$

and are different from the A_j obtained in Algorithm 26 because a Kronecker map is not used. Let

$$\Delta=\prod_{k=1}^{n}\mathrm{LC}(f_k^{\beta})\mathrm{LC}(g_k^{\beta})\in\mathbb{Z}_p[y_1,y_2,\ldots,y_m].$$

Since $\deg(\Delta)\leq 2nd$ and $N_a\geq\hat{N}_{\max}$, we have that

$$\Pr[\hat{Y}_j\text{ picked in Line 8 of Algorithm 27 is bad}:0\leq j\leq\hat{N}_{\max}-1]\leq\frac{2ndN_a}{q-1}.$$

Hence Pr[Algorithm 27 returns FAIL in Line 25] $\leq$

$$\frac{2N_an^2\left(\log_{p_{\min}}(th\sqrt{n})+md\log_{p_{\min}}\mathrm{e}\right)}{N}+\frac{2ndN_a}{q-1}+\frac{nd^2N_a}{q-1}. \tag{7.9}$$

Our result follows by adding (7.7), (7.8) and (7.9). □

7.4.5 Complexity Analysis

Theorem 7.11. Let $B=[A|b]$ be a $n\times(n+1)$ augmented matrix such that $\#B_{ij}\leq t$ and $\deg(B_{ij})\leq d$. Suppose the integer coefficients of B_{ij} are l base C digits long. That is, $\|B_{ij}\|_{\infty}\leq C^l$. Let prime p chosen at random from P and $C<p<2C$. A black box probe costs $O(n^2tl+n^2mdt+n^3)$ arithmetic operations in $\mathbb{Z}_p$.

Proof. This follows from Theorem 6.44. □

Theorem 7.12. Let $\hat{N}_{\max} = \max_{k=1}^{n}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\})$ where $f_{i,k}, g_{i,k}, f_k, g_k$ are as defined in (7.1) and let $e_{\max} = 2+\max_{k=1}^{n}\{\deg(f_k)+\deg(g_k)\}$. Let $H = \max_k(\|f_k\|_\infty, \|g_k\|_\infty))$. Then the number of black box probes required by our algorithm to interpolate the solution vector x is $O(e_{\max}\hat{N}_{\max}\log H)$.

Proof. This follows from Theorem 6.45. □

7.5 Implementation and Benchmarks

7.5.1 Implementation

We have implemented our new algorithm in Maple with some parts coded in C to improve its overall efficiency. The parts coded in C include evaluating an augmented matrix at integer points modulo prime p, solving the evaluated augmented matrix with integer entries over $\mathbb{Z}_p$ using Gaussian elimination, finding and factoring the feedback polynomial produced by the Berlekamp-Massey algorithm, solving a $t \times t$ shifted Vandermonde system and performing dense rational function interpolation using the MQRFR algorithm modulo a prime. Each probe to the black box is computed using C code, and it supports primes up to 63 bits in length. We have benchmarked our code on a 24 core Intel Gold 6342 processor (`maple` server) with 256 gigabytes of RAM using only 1 core running at 2.8GHz (base) and 3.5GHz (turbo).

7.5.2 Benchmarks

To test the performance of our algorithm, we create the following artificial problem where $\#\det(A) \gg \#g_k$ and $\#\det(A^k) \gg \#f_k$. Let $D \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^{n\times n}$ with $\text{rank}(D) = n$. Let the coefficient matrix A be a diagonal matrix such that its diagonal entries are non-zero polynomials $g_1, \ldots, g_n$ and let the vector

$$b = \begin{bmatrix} f_1 & f_2 & \cdots & f_n \end{bmatrix}^T.$$

Clearly, the vector

$$x = \begin{bmatrix} \frac{f_1}{g_1} & \frac{f_2}{g_2} & \cdots & \frac{f_n}{g_n} \end{bmatrix}^T$$

solves $Ax = b$. But, suppose we create a new parametric linear system

$$Wx^* = c$$

by premultiplying $Ax = b$ by D so that

$$Wx^* = (DA)x^* = Db = c.$$

Then both parametric systems $Ax = b$ and $Wx^* = c$ are equivalent. That is,

$$x^* = W^{-1}c = \frac{\text{adj}(DA)c}{\det(DA)} = \frac{\text{adj}(A)\text{adj}(D)Db}{\det(D)\det(A)} = \frac{\text{adj}(A)b}{\det(A)} = A^{-1}b = x$$

where adj denotes the adjoint matrix. Our main idea for creating this test problem is to ensure that $\gcd(\det(W^i), \det(W))$ is large, so that our new method avoids the gcd computations and computing large determinants.

In Table 7.1, we compare our new algorithm (row ParamLinSolve) with a Maple implementation of the Bareiss/Edmonds fraction free one step Gaussian elimination method with Lipson's fraction formula for back substitution (row Bareiss), a Maple implementation of the Gentleman & Johnson minor expansion method (row Gentleman) and using Maple's commands `ReducedRowEchelonForm` (row ReducedRow) and `LinearSolve` (row LinearSolve) for solving the systems $Wx^* = c$ that were created artificially. The artificial systems $Wx^* = c$ for Table 7.1 were created using the following Maple code:

```
CreateSystem := proc(n,m,T,dT,t,d) local A, D,W,c,b,Y,i;
   Y := [ seq(y||i,i=1..m) ];
   D := Matrix(n,n, () -> randpoly( Y,terms=T, degree=dT));
   b := Vector[column](n, () -> randpoly(Y, terms = t, degree= d));
   i := [ seq( randpoly( Y, terms = t, degree= d),i=1..n) ];
   A := DiagonalMatrix(i);
   W,c := D.A, D.b; return W,c,A,D;
end:
```

The three input systems solved in Table 7.3 are real systems (Example 1.12 and two other real systems) which were the motivation for this work. Note that the timings reported for the real systems in Table 7.3 are in the columns and not in rows as in Table 7.1. The breakdown of the timings for all individual algorithms involved for computing the system named bigsys are reported in 7.2. Column max in Table 7.3 contains the number of terms in the largest polynomial to be interpolated in the rational functions of the unique solution of a parametric linear system.

The artificial input systems $Wx^* = c$ were created by generating matrices D, A and column vector b randomly, with all of their entries in $\mathbb{Z}[y_1, \ldots, y_m]$ where $m = 10, \deg(D_{ij}) \leq d_T = 5, \#D_{i,j} = T \leq 2$ and $\deg(A_{ij}), \deg(b_j) \leq d = 10, \#A_{i,j}, \#b_j = t \leq 5$ and $\text{rank}(A) = \text{rank}(D) = n$ for $3 \leq n \leq 10$. Using the Gentleman & Johnson algorithm, we obtain $\#\det(A), \#\det(D), \#\det(W)$ (rows 2-4) and the total CPU time used to compute each of them are reported in rows 10-13. We remark that we did not compute the $\gcd(\det(A^k), \det(A))$ when the Gentleman & Johnson algorithm was used. As the reader can see from Table 7.1, our algorithm performed better than other algorithms for $n \geq 5$.

Table 7.1: CPU Timings for solving $Wx^* = c$ with $\#f_i, \#g_i \leq 5$ for $3 \leq n \leq 10$.

n	3	4	5	6	7	8	9	10
$\#\det(A)$	125	625	3,125	15,500	59,851	310,796	1,923,985	9,381,213
$\#\det(D)$	40	336	3,120	38,784	518,009	8,477,343	156,424,985	NA
$\#\det(W)$	5,000	209,960	9,741,747	NA	NA	NA	NA	NA
ParamLinSolve	0.079s	0.176s	0.154s	0.211s	0.220s	0.239s	0.259s	0.317s
LinearSolve	0.129s	1.26s	304.20s	124200s	!	!	!	!
ReducedRow	0.01s	0.083	11.05s	3403.2s	!	!	!	!
Bareiss	2.02s	!	!	!	!	!	!	!
Gentleman	0.040s	3.19s	239.40s	!	!	!	!	!
time-det(A)	0s	0s	0.003s	0.08s	0.898s	0.703s	17.03s	25.32s
time -det(D)	0s	0s	0.007s	1.21s	1.39s	601.8s	2893.8s	!
time-det(W)	0s	0.310s	20.44s	!	!	!	!	!

! = out of memory and NA means Not Attempted

Table 7.2: Breakdown of CPU timings for all individual algorithms for computing bigsys

	Time(ms)	Percentage
Matrix Evaluation	151.48s	1.9 %
Gaussian Elimination	110.71s	1.4 %
Univariate Rational Function Interpolation	706.07s	9 %
Finding $\lambda \in \mathbb{Z}_p[z]$ using the Berlekamp-Massey Algorithm	208.25s	2.6 %
Roots of λ over $\mathbb{Z}_p$	4856.96s	62 %
Solving Vandermonde systems	434.46s	5.6 %
Multiplication and Addition of Evaluation points	257.40s	3.3 %
Computing Discrete logarithms	586.64s	7.6 %
Miscellaneous	464.67s	9.4 %
Overall Time	7776s	100 %

Table 7.3: CPU Timings for solving three real parametric linear systems

system names	n	m	max	ParamLinSolve	Gentleman	LinearSolve	ReducedRow	Bareiss	$\#\det(A)$
Bspline	21	5	26	0.220s	2623.8s	0.021s	0.026s	0.500s	1033
Bigsys	44	48	58240	7776s	!	17.85s	1.66s	!	6037416
Caglar	12	56	23072	1685.57s	NA	1232.40s	15480.35s	NA	15744
Sys66a	66	34	145744	665507.32s (184.86) hours	!	!	!	!	NA
Sys66b	66	31	107468	255819.27s (71.06) hours	!	!	!	!	NA

! = out of memory and NA=Not Attempted

In our experiments, the cost of evaluating augmented matrices over $\mathbb{Z}_p$ is often the most expensive part. But as the reader can see in Table 7.2, computing the roots of the feedback polynomial for the bigsys system is the dominating cost. This is because the number of terms in many of the polynomials f_i, g_i to be interpolated is large. In particular, it has four polynomials where $\max(\#f_i, \#g_i) > 50,000$ and our root finding algorithm for computing the roots of $\lambda(z)$ costs $O(t^2 \log p)$ where $t = \deg(\lambda)$ is the number of terms of the polynomials f_i and g_i being interpolated. Faster root finding algorithms such as the Tangent Graeffe algorithm by Grenet, vander Hoeven and Lecerf [Grenet et al., 2015] and vander Hoeven and Monagan [van der Hoeven and Monagan, 2020] which uses $O(t \log p)$ arithmetic operations in $\mathbb{Z}_p$ could be used to speed up our new algorithm.

Chapter 8

Conclusion

In this thesis, we have developed a new sparse multivariate rational function interpolation algorithm which uses a new set of evaluation points and employs a Kronecker substitution to effectively reduce the size of our working primes. This new approach transforms the problem of interpolating a multivariate rational function using Cuyt and Lee's method and the Ben-Or/Tiwari algorithm into a univariate rational function interpolation problem.

We have designed a new Dixon resultant algorithm that computes the monic square-free factors of the Dixon resultant R of a parametric polynomial system using our new sparse rational function interpolation method. We have shown that there is a huge reduction in the number of terms, the number of black box probes required, and the number of primes needed when the monic square-free factors R_j of the Dixon resultant R are interpolated instead of interpolating the Dixon resultant R on problems in practice.

We have implemented our Dixon resultant algorithm in Maple and implemented several subroutines in C including the evaluation of the Dixon matrix modulo a prime, computing the determinant of integer matrices modulo a prime, solving a $t \times t$ shifted Vandermonde system and performing dense rational function interpolation using the MQRFR algorithm modulo a prime. We have tested our new Dixon resultant code on many real parametric polynomial systems that emerged from practical problems.

Our benchmarks showed that our new Dixon resultant algorithm is much faster than other algorithms (Zippel's sparse interpolation, Gentleman & Johnson and Dixon-EDF method) for computing R in expanded form whenever the Dixon resultant is large while it has relatively small square-free factors or a large polynomial content. However, our new code is not always the fastest.

We have comprehensively identified all the causes of failure in our new Dixon resultant algorithm. Additionally, we have presented a thorough analysis of failure probabilities, and we have also presented the complexity of our new algorithm in terms of the number of black box probes involved.

Furthermore, we have applied our new sparse rational function interpolation method to solve the problem of interpolating the rational function entries of the unique solution

vector of a parametric linear system. A new black box algorithm has been developed and implemented in Maple and C. We have compared the performance of our new black box algorithm for solving parametric linear systems to other known algorithms/implementations. A detailed failure probability and complexity analysis was also presented.

Bibliography

Albert Dixon. On a form of the eliminant of two quantics. *Proceedings of the London Mathematical Society*, **2**(1):468–478, 1908.

Albert Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, **6**(4969):209–236, 1909.

Aleksandr Gelfond. *Transcendental and Algebraic numbers*. Courier Dover Publications, 2015.

Annie Cuyt and Wen-shin Lee. Sparse interpolation of multivariate rational functions. *Theoretical Computer Science*, **412**(16):1445–1456, 2011.

Arthur Cayley. Note sur la méthode d'élimination de Bezout. *J. Reine Angew. Math.*, **53**: 366–367, 1857.

Arthur Cayley. On the theory of elimination. *Cambridge and Dublin Mathematical Journal*, **III**:210–270, 1865.

Arthur D Chtcherba and Deepak Kapur. On the efficiency and optimality of Dixon-based resultant methods. In *Proceedings of ISSAC '2002*, pages 29–36. ACM, 2002.

Ayoola Jinadu and Michael Monagan. An Interpolation Algorithm for computing Dixon Resultants. In *Proceedings of CASC '2022, LNCS **13366**:185–205*. Springer, 2022a.

Ayoola Jinadu and Michael Monagan. A new interpolation algorithm for computing Dixon resultants. In *Comms. in Computer Algebra **3**:88-91*, pages 88–91. ACM, 2022b.

Ayoola Jinadu and Michael Monagan. Solving Parametric Linear Systems using Sparse Rational Function Interpolation. In *Proceedings of CASC '2023, LNCS **14139**:233–254*. Springer Nature Switzerland, 2023.

Bruno Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, **41**:475–511, 2006.

Bruno Grenet, Joris van Der Hoeven, and Gregoire Lecerf. Randomized root finding over finite FFT-fields using tangent Graeffe transforms. In *Proceedings of ISSAC 2015*, pages 197–204. ACM, 2015.

Daniel Roche. What can (and can't) we do with sparse polynomials? In *Proceedings of ISSAC*, pages 25–30. ACM, 2018.

Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc.*, volume **20**, pages 415–440, 1971.

David Cox, John Little, and Donal O'Shea. *Using Algebraic Geometry.* Graduate Texts in Mathematics. Springer New York, 2005.

David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms.* Fourth Edition, Springer Cham, 2015.

Deepak Kapur and Lakshman Yagati. Elimination methods: An introduction. In *Proceedings of Symbolic and Numerical Computation for Artificial Intelligence*, pages 45–89. Academic Press, 1992.

Deepak Kapur and Tushar Saxena. Comparison of various multivariate resultant formulations. In *Proceedings of ISSAC 1995*, pages 187–194. ACM, 1995.

Deepak Kapur and Tushar Saxena. Extraneous factors in the Dixon resultant formulation. In *Proceedings of ISSAC 1997*, pages 141–148. ACM, 1997.

Deepak Kapur, Tushar Saxena, and Lu Yang. Algebraic and geometric reasoning using Dixon resultants. In *Proceedings of ISSAC 1994*, pages 99–107. ACM, 1994.

Duane Storti. Algebraic Skeleton Transform: A symbolic computation challenge. *Submitted to Faculty Papers and Data, Mechanical Engineering, ResearchWorks Archive. http://hdl.handle.net/1773/48587*, 2022.

Erich Kaltofen. Fifteen years after DSC and WLSS2 what parallel computations i do today: invited lecture at PASCO 2010. In *PASCO*, volume **10**, pages 10–17. ACM, 2010.

Erich Kaltofen and Barry M Trager. Computing with polynomials given by black boxes for their evaluations: greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, **9**(3):301–320, 1990.

Erich Kaltofen and Zhengfeng Yang. On exact and approximate interpolation of sparse rational functions. In *Proceedings of ISSAC 2007*, pages 203–210. ACM, 2007.

Erich Kaltofen, Wen-shin Lee, and Austin A Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of zippel's algorithm. In *Proceedings of ISSAC 2000*, pages 192–201. ACM, 2000.

Erwin Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. of Computation*, **22**(103):565–578, 1968.

Francis Macaulay. Some formulae in elimination. *Proceedings of the London Mathematical Society*, **35**(1):3–27, 1902.

Francis Macaulay. *The Algebraic Theory of Modular Systems*, volume **19**. Cambridge Tracts in Mathematics and Mathematical Physics, 1916.

Hirokazu Murao and Tesuro Fujise. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *Journal of Symbolic Computation*, **21**(4): 377–396, 1996.

Jack Edmonds. Systems of distinct representatives and Linear algebra. *J. Res. Nat. Bur. Standards Sect. B*, **71**(4):241–245, 1967.

Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, **27**(4):701–717, 1980.

Jennifer de Kleine, Michael Monagan, and Allan Wittkopf. Algorithms for the non-monic case of the sparse modular GCD algorithm. In *Proceedings of the ISSAC 2005*, pages 124–131. ACM, 2005.

Jiaxiong Hu. *Computing polynomial greatest common divisors using sparse interpolation.* PhD Thesis, Simon Fraser University, 2018.

Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. In *Proceedings of ISSAC 2016*, pages 271–278. ACM, 2016.

Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. *Journal of Symbolic Computation*, **105**:28–63, 2021.

Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra.* Cambridge university press, 2013.

John D Lipson. Symbolic methods for the computer solution of linear equations with applications to flowgraphs. *Proceedings of SISMC*, pages 233–303, 1969.

Joris van der Hoeven and Michael Monagan. Implementing the tangent Graeffe root finding method. In *Proceedings of ICMS 2020*, pages 482–492. Springer, 2020.

Keith O Geddes, Stephen R Czapor, and George Labahn. *Algorithms for computer algebra.* Kluwer, 1992.

M Moreno Maza. On triangular decompositions of algebraic varieties. Technical report, TR 4/99, NAG Ltd, Oxford, UK, 1999. Presented at the MEGA-2000 Conference, Bath, England, 2000.

Manfred Minimair. Randomized detection of extraneous factors. In *Proceedings of ISSAC 2014*, pages 335–342. ACM, 2014.

Manfred Minimair. Computing the Dixon Resultant with the Maple Package DR. In *Applications of Computer Algebra: Kalamata, Greece, July 20–23 2015*, pages 273–287. Springer, 2017.

Mark Giesbrecht and Daniel S Roche. Interpolation of shifted-lacunary polynomials. *Computational Complexity **19**(3): 333–354*, 2010.

Mark Giesbrecht, George Labahn, and Wen-shin Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. In *Proceedings of ISSAC 2006*, pages 116–123, 2006.

Mark Giesbrecht, George Labahn, and Wen-shin Lee. Symbolic–numeric sparse interpolation of multivariate polynomials. *Journal of Symbolic Computation*, **44**(8):943–959, 2009.

Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on theory of computing*, pages 301–309, 1988.

Michael Kalkbrener. *Three contributions to elimination theory.* PhD Thesis, Research Institute for Symbolic Computation, 1991.

Michael Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In *Proceedings of ISSAC 2004*, pages 243–249. ACM, 2004.

Michael Monagan. Private Communication. 2023a.

Michael Monagan. Implementation of the Gentleman & Johnson minor expansion algorithm and the Dixon-EDF algorithm in Maple, 2023b.

Michael Monagan and Roman Pearce. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *ACM Communications in Computer Algebra*, 48(3/4):166–186, 2015.

Nadia Ben Atti, Gema M Diaz-Toca, and Henri Lombardi. The Berlekamp-Massey algorithm revisited. *AAECC*, **17**:75–82, 2006.

Niels Möller and Torbjorn Granlund. Improved division by invariant integers. *IEEE Transactions on Computers*, **60**(2):165–175, 2010.

Paul S Wang. A p-adic algorithm for univariate partial fractions. In *Proceedings of the fourth ACM symposium on Symbolic and algebraic computation*, pages 212–217, 1981.

Paul Vrbik and Michael Monagan. Lazy and Forgetful Polynomial Arithmetic and Applications. 2009.

Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM 79*, pages 216–226. Springer, 1979.

Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, **9**(3):375–403, 1990.

Robert Lewis. The Kapur-Saxena-Yang Variant of the Dixon Resultant, 1996. URL https://www.math.unm.edu/~vageli/SEMINAR/KSYResult.pdf.

Robert Lewis. Comparison of the greatest common divisor (GCD) in several systems, 2004. URL https://home.bway.net/lewis/fermat/gcdcomp.

Robert Lewis. Dixon-EDF: the premier method for solution of parametric polynomial systems. In *ACA 2015*, pages 237–256. Springer, 2017.

Robert Lewis. Private Communication. 2018a.

Robert Lewis. Resultants, Implicit parameterizations, and Intersections of surfaces. In *Proceedings of ICMS 2018*, pages 310–318. Springer, 2018b.

Robert Lewis. New heuristics and extensions of the Dixon resultant for solving polynomial systems. *Talk at: Applications of Computer Algebra, Montreal, Canada*, pages 16–20, 2019.

Robert Lewis. Image Analysis: Identification of Objects via Polynomial Systems. *Mathematics in Computer Science*, **14**(3):551–558, 2020.

Sara Khodadad and Michael Monagan. Fast rational function reconstruction. In *Proceedings of ISSAC 2006*, pages 184–190. ACM, 2006.

Shafi Goldwasser and Guy Rothblum. On best-possible obfuscation. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 194–213. Springer, 2007.

Snehal Bhayani, Zuzana Kukelova, and Janne Heikkila. A sparse resultant based method for efficient minimal solvers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1770–1779, 2020.

Stephen Pohlig and Hellman Martin. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. **24**:106–110, 1978.

W. Gentleman and S. Johnson. The evaluation of determinants by expansion by minors and the general problem of substitution. *Mathematics of Computation **28**(126):543–548*, 1974.

Wen-tsun Wu. A zero structure theorem for polynomial equations solving and its applications. **1**:2–12, MM Research Preprints, 1997.

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

Wolfgang Schmidt. *Equations over finite fields: an elementary approach*, volume 536. Springer, 2006.

Appendix A

Input Parametric Systems for the data reported in Table 1.3

A.1 Robot arms system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_7][x_1, x_2, x_3, x_4]$ using lexicographical order with $x_2 > x_3 > x_4 > x_1$ where

$$
\begin{aligned}
\hat{f}_1 = {} & x_1^2x_2^2x_3^2x_4^2y_1 + x_1^2x_2^2x_3^2y_1 + 2x_1^2x_2^2x_3^2y_4 + x_1^2x_2^2x_4^2y_1 - 2x_1^2x_2^2x_4^2y_4 + x_1^2x_3^2x_4^2y_1 \\
& + x_2^2x_3^2x_4^2y_1 - 2x_2^2x_3^2x_4^2y_3 + x_1^2x_2^2y_1 + x_1^2x_3^2y_1 + 2x_1^2x_3^2y_3 + 2x_1^2x_3^2y_4 + x_1^2x_4^2y_1 + 2x_1^2x_4^2y_3 \\
& - 2x_1^2x_4^2y_4 + x_2^2x_3^2y_1 - 2x_2^2x_3^2y_3 + 2x_2^2x_3^2y_4 + x_2^2x_4^2y_1 - 2x_2^2x_4^2y_3 - 2x_2^2x_4^2y_4 + x_3^2x_4^2y_1 \\
& + x_1^2y_1 + 2x_1^2y_3 + x_2^2y_1 - 2x_2^2y_3 + x_3^2y_1 + 2x_3^2y_4 + x_4^2y_1 - 2x_4^2y_4 + y_1 + 2x_1^2x_3^2x_4^2y_3 \\
\hat{f}_2 = {} & 2x_1^2x_2^2x_3^2x_4y_4 - 2x_1^2x_2^2x_3x_4^2y_4 + 2x_1^2x_2x_3^2x_4^2y_3 - 2x_1x_2^2x_3^2x_4^2y_3 - 2x_1^2x_2^2x_3y_4 + 2x_1^2x_2^2x_4y_4 \\
& + 2x_1^2x_2x_3^2y_3 + 2x_1^2x_2x_4^2y_3 + 2x_1^2x_3^2x_4y_4 - 2x_1^2x_3x_4^2y_4 - 2x_1x_2^2x_3^2y_3 - 2x_1x_2^2x_4^2y_3 \\
& + 2x_2^2x_3^2x_4y_4 - 2x_2^2x_3x_4^2y_4 + 2x_2x_3^2x_4^2y_3 + 2x_1^2x_2y_3 \\
& - 2x_1x_3^2y_3 - 2x_1x_4^2y_3 - 2x_2^2x_3y_4 + 2x_2^2x_4y_4 + 2x_2x_3^2y_3 + 2x_2x_4^2y_3 + 2x_3^2x_4y_4 - 2x_3x_4^2y_4 \\
& - 2x_1y_3 + 2x_2y_3 - 2x_3y_4 + 2x_4y_4 - 2x_1^2x_3y_4 + 2x_1^2x_4y_4 - 2x_1x_2^2y_3 - 2x_1x_3^2x_4^2y_3 \\
\hat{f}_3 = {} & -x_1^2x_3^2y_2^2y_3 - x_1^2x_3^2y_2^2y_4 - x_1^2x_3^2y_2^2y_5 - x_1^2x_3^2y_2^2y_6 - x_1^2x_3^2y_3 - x_1^2x_3^2y_4 + x_1^2x_3^2y_5 \\
& - 4x_1^2x_3y_2y_5 - x_1^2y_2^2y_3 + x_1^2y_2^2y_4 + x_1^2y_2^2y_5 - x_1^2y_2^2y_6 + x_3^2y_2^2y_3 - x_3^2y_2^2y_4 - x_3^2y_2^2y_5 \\
& - x_1^2y_3 + x_1^2y_4 - x_1^2y_5 - x_1^2y_6 + x_3^2y_3 - x_3^2y_4 + x_3^2y_5 - x_3^2y_6 - 4x_3y_2y_5 + y_2^2y_3 + y_2^2y_4 \\
& - y_2^2y_6 + y_3 + y_4 - y_5 - y_6 - x_1^2x_3^2y_6 - x_3^2y_2^2y_6 + y_2^2y_5 + y_2^2y_5 \\
\hat{f}_4 = {} & -x_1^2x_3^2y_2^2y_7 - 2x_1^2x_3^2y_2y_5 + 2x_1^2x_3y_2^2y_4 + 2x_1^2x_3y_2^2y_5 + 2x_1x_3^2y_2^2y_3 - x_1^2x_3^2y_7 - x_1^2y_2^2y_7 \\
& - x_3^2y_2^2y_7 + 2x_1^2x_3y_4 - 2x_1^2x_3y_5 + 2x_1^2y_2y_5 + 2x_1x_3^2y_3 + 2x_1y_2^2y_3 - 2x_3^2y_2y_5 \\
& + 2x_3y_2^2y_4 + 2x_3y_2^2y_5 - x_1^2y_7 - x_3^2y_7 - y_2^2y_7 + 2x_1y_3 + 2x_3y_4 - 2x_3y_5 + 2y_2y_5 - y_7.
\end{aligned}
$$

A.2 Circle system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4, \cdots \hat{f}_9\} \subset \mathbb{Q}[y_1, y_2, y_3, y_4][x_1, x_2, x_3, x_4, \ldots, x_9]$ using lexicographical order with $x_2 > x_3 > x_4 > \cdots > x_9 > x_1$ where

$$\hat{f}_1 = x_7^2 - 2x_7y_1 + x_8^2 - 2x_8x_9 + x_9^2 + y_1^2 - y_4^2$$
$$\hat{f}_2 = x_7^2y_2^4 + x_8^2y_2^4 - y_2^4y_4^2 - 4x_1x_7y_2^3 - 4x_8y_2^4 + 4x_1^2y_2^2 + 2x_7^2y_2^2 + 2x_8^2y_2^2 + 4y_2^4 - 2y_2^2y_4^2 - 4x_1x_7y_2 - 4x_8y_2^2 + x_7^2 + x_8^2 - y_4^2$$
$$\hat{f}_3 = x_1x_7y_2^4 - 2x_1^2y_2^3 - 2x_8y_2^3 + 2x_1^2y_2 + 4y_2^3 - x_1x_7 - 2x_8y_2$$
$$\hat{f}_4 = -x_3y_2^2y_4^2 + x_7^2y_2^2 - x_7y_1y_2^2 + x_8^2y_2^2 - x_8x_9y_2^2 - 2x_1x_7y_2 + 2x_1y_1y_2 - x_3y_4^2 - 2x_8y_2^2 + 2x_9y_2^2 + x_7^2 - x_7y_1 + x_8^2 - x_8x_9$$
$$\hat{f}_5 = -x_1^2x_3^2y_2^4 - x_2^2y_2^4y_4^2 + 2x_1^2x_3^2y_2^2 + x_1^2y_2^4 - 2x_2^2y_2^2y_4^2 - x_1^2x_3^2 - 2x_1^2y_2^2 - x_2^2y_4^2 - 4x_3^2y_2^2 + x_1^2 + 4y_2^2$$
$$\hat{f}_6 = x_1x_7y_2^4 + x_4x_7y_2^4 + x_5x_8y_2^4 - 2x_1^2y_2^3 - 2x_1x_4y_2^3 - 2x_5y_2^4 + 2x_4x_7y_2^2 + 2x_5x_8y_2^2 - 2x_8y_2^3 + 2x_1^2y_2 - 2x_1x_4y_2 - 2x_5y_2^2 + 4y_2^3 - x_1x_7 + x_4x_7 + x_5x_8 - 2x_8y_2$$
$$\hat{f}_7 = x_4x_7 - x_4y_1 + x_5x_8 - x_5x_9 - x_6x_8 + x_6x_9 - x_7y_3 + y_1y_3$$
$$\hat{f}_8 = x_1^2y_2^4 + x_1x_4y_2^4 + 2x_1x_7y_2^3 + x_8y_2^4 - 6x_1^2y_2^2 - 2x_5y_2^3 - 2y_2^4 + 2x_1x_7y_2 + x_1^2 - x_1x_4 - 2x_5y_2 + 6y_2^2 - x_8$$
$$\hat{f}_9 = -x_2y_2^2y_4^2 + x_1x_7y_2^2 - x_1y_1y_2^2 + 2x_4x_7y_2^2 - x_4y_1y_2^2 + 2x_5x_8y_2^2 - x_5x_9y_2^2 - x_6x_8y_2^2 - x_7y_2^2y_3 - 2x_1x_4y_2 + 2x_1y_2y_3 - x_2y_4^2 - 2x_5y_2^2 + 2x_6y_2^2 - x_1x_7 + x_1y_1 + 2x_4x_7 - x_4y_1 + 2x_5x_8 - x_5x_9 - x_6x_8 - x_7y_3 - 2x_8y_2 + 2x_9y_2$$

A.3 Geddes2 system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4\} \subset \mathbb{Q}[y_1][x_1, \ldots, x_4]$ using lexicographical order with $x_2 > x_3 > x_4 > x_1$ where

$$\hat{f}_1 = 2(x_1-1)^2 + 2x_3^2 - 2x_3x_2 + 2x_2 + y_1^2(x_2-1)^2 - 2x_1x_2 + 2y_1x_4(1-x_2)(x_2-x_3) + 2x_1x_3x_2x_4(x_4-y_1) + x_1^2x_4^2(1-2x_3) + 2x_1x_4^2(x_3-x_2) + 2x_1x_4y_1(x_3-1) + 2x_1x_3x_2(y_1+1) + \left(x_1^2 - 2x_1\right)x_3^2x_4^2 + 2x_1^2x_3^2 + 4x_1(1-x_1)x_3 + x_4^2(x_3-x_2)^2$$
$$\hat{f}_2 = x_4(2x_3+1)(x_2-x_3) + y_1(x_3+2)(1-x_2) + x_1(x_1-2)x_4 + x_1(1-2x_1)x_3x_4 + x_1y_1(-x_3x_2 + x_2 + x_3 - 1) + x_1(x_1+1)x_3^2x_4$$
$$\hat{f}_3 = -x_1^2(x_3-1)^2 + 2x_3(x_3-x_2) - 2x_2 + 2$$
$$\hat{f}_4 = x_1^2 - 4x_2^2 + 4x_3 + 3y_1^2(x_2-1)^2 - 3x_4^2(x_3-x_2)^2 + 3x_1^2x_4^2(x_3-1)^2 + x_1^2x_3(x_3-2) + 6x_1x_4y_1(x_3x_2 + x_2 + x_3 - 1)$$

A.4 Heron3d system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_6\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_6][x_1, x_2, \ldots, x_6]$ using lexicographical order with $x_2 > x_3 > x_4 > x_5 > x_6 > x_1$ where

$$\begin{aligned}
\hat{f}_1 &= x_2^2 + x_3^2 - y_3^2 \\
\hat{f}_2 &= (x_2 - y_1)^2 + x_3^2 - y_2^2 \\
\hat{f}_3 &= x_4^2 + x_5^2 + x_6^2 - y_6^2 \\
\hat{f}_4 &= (x_4 - y_1)^2 + x_5^2 + x_6^2 - y_4^2 \\
\hat{f}_5 &= (x_4 - x_2)^2 + (x_5 - x_3)^2 + x_6^2 - y_5^2 \\
\hat{f}_6 &= -x_3 x_6 y_1 + 6x_1,
\end{aligned}$$

A.5 Heron4d system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_9\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_{10}][x_1, x_2, \ldots, x_{10}]$ using lexicographical order with $x_2 > x_3 > \cdots > x_9 > x_{10} > x_1$ where

$$\begin{aligned}
\hat{f}_1 &= x_2^2 + x_3^2 - y_3^2 \\
\hat{f}_2 &= (x_2 - y_1)^2 + x_3^2 - y_2^2 \\
\hat{f}_3 &= x_4^2 + x_5^2 + x_6^2 - y_6^2 \\
\hat{f}_4 &= (x_4 - y_1)^2 + x_5^2 + x_6^2 - y_4^2 \\
\hat{f}_5 &= (x_4 - x_2)^2 + (x_5 - x_3)^2 + x_6^2 - y_5^2 \\
\hat{f}_6 &= x_7^2 + x_8^2 + x_9^2 + x_{10}^2 - y_7^2 \\
\hat{f}_7 &= (x_7 - y_1)^2 + x_8^2 + x_9^2 + x_{10}^2 - y_8^2 \\
\hat{f}_8 &= (x_7 - x_2)^2 + (x_8 - x_3)^2 + x_9^2 + x_{10}^2 - y_9^2 \\
\hat{f}_9 &= (x_7 - x_4)^2 + (x_8 - x_5)^2 + (x_9 - x_6)^2 + x_{10}^2 - y_{10}^2 \\
\hat{f}_{10} &= -x_3 x_6 x_{10} y_1 + 24x_1
\end{aligned}$$

A.6 Heron5d system

$\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_9\} \subset \mathbb{Q}[y_1, y_2, \ldots, y_{15}][x_1, x_2, \ldots, x_{15}]$ using lexicographical order with $x_2 > x_3 > x_4 > x_5 > x_6 > \cdots > x_{15} > x_1$ where

$$
\begin{aligned}
\hat{f}_1 &= x_5^2 + x_9^2 - y_3^2 \\
\hat{f}_2 &= -2x_5y_1 + y_1^2 - y_2^2 + y_3^2 \\
\hat{f}_3 &= x_6^2 + x_{10}^2 + x_{13}^2 - y_6^2 \\
\hat{f}_4 &= -2x_6y_1 + y_1^2 - y_4^2 + y_6^2 \\
\hat{f}_5 &= -2x_5x_6 - 2x_9x_{10} + y_3^2 - y_5^2 + y_6^2 \\
\hat{f}_6 &= x_3^2 + x_7^2 + x_{11}^2 + x_{14}^2 - y_7^2 \\
\hat{f}_7 &= -2x_7y_1 + y_1^2 + y_7^2 - y_8^2 \\
\hat{f}_8 &= -2x_5x_7 - 2x_9x_{11} + y_3^2 + y_7^2 - y_9^2 \\
\hat{f}_9 &= -2x_6x_7 - 2x_{10}x_{11} - 2x_{13}x_{14} + y_6^2 + y_7^2 - y_{10}^2 \\
\hat{f}_{10} &= x_2^2 + x_4^2 + x_8^2 + x_{12}^2 + x_{15}^2 - y_{11}^2 \\
\hat{f}_{11} &= -2x_8y_1 + y_1^2 + y_{11}^2 - y_{12}^2 \\
\hat{f}_{12} &= -2x_5x_8 - 2x_9x_{12} + y_3^2 + y_{11}^2 - y_{13}^2 \\
\hat{f}_{13} &= -2x_6x_8 - 2x_{10}x_{12} - 2x_{13}x_{15} + y_6^2 + y_{11}^2 - y_{14}^2 \\
\hat{f}_{14} &= -x_2x_3x_9x_{13}y_1 + 120x_1 \\
\hat{f}_{15} &= -2x_3x_4 - 2x_7x_8 - 2x_{11}x_{12} - 2x_{14}x_{15} + y_7^2 + y_{11}^2 - y_{15}^2
\end{aligned}
$$

Appendix B

Subresultants

Definition B.1. Let F be a field and let

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_{d_f} x^{d_f}$$

$$g(x) = b_0 + b_1 x + b_2 x^2 + \cdots b_{d_g} x^{d_g}$$

where the coefficients $a_i, b_j \in F[y_1, y_2, \ldots, y_m]$, for $0 \leq i \leq d_f = \deg(f)$ and $0 \leq j \leq d_g = \deg(g)$. For $0 \leq k \leq \min\{d_f, d_g\}$, the k-th subresultant of f and g as denoted by σ_k is defined as the determinant $\sigma_k \in F$ of the $(d_g + d_f - 2k) \times (d_g + d_f - 2k)$ matrix

$$S_k = \begin{pmatrix}
a_{d_f} & & & & b_{d_g} & & & & & \\
a_{d_f-1} & a_{d_f} & & & b_{d_g-1} & b_{d_g} & & & & \\
\vdots & & \ddots & & \vdots & & \ddots & & & \\
a_{d_f-d_g+k+1} & \cdots & \cdots & a_{d_f} & b_{k+1} & \cdots & \cdots & b_{d_g} & & \\
\vdots & & & \vdots & \vdots & & & & \ddots & \\
a_{k+1} & \cdots & \cdots & a_{d_g} & b_{d_g-d_f+k+1} & \cdots & \cdots & \cdots & \cdots & b_{d_g} \\
\vdots & & & \vdots & & & & & & \vdots \\
\vdots & & & \vdots & & & & & & \vdots \\
a_{2k-d_g+1} & \cdots & \cdots & a_k & b_{2k-d_f+1} & \cdots & \cdots & \cdots & \cdots & b_k
\end{pmatrix}$$

where S_0 is the Sylvester matrix of f and g.

The matrix S_k is the matrix obtained from the Sylvester matrix by deleting the last $2k$ rows and the last k columns with coefficients of f and the last k columns with coefficients of g. We also note that the matrix S_k is a submatrix of S_i if $k \geq i$.

Theorem B.2. [von zur Gathen and Gerhard, 2013, Theorem 6.53, Theorem 6.54 and Exercise 6.49]

Let r_i^*, q_i^*, s_i^* denote the results of the Extended Euclidean Algorithm and let r_i, q_i, s_i denote the results of the monic Extended Euclidean Algorithm. Let $\alpha_i = \mathrm{LC}(r_i)$ and $n_i = \deg(r_i^*)$ for $0 \leq i \leq l$, σ_k is the k-th sub-resultant and S_k the sub-matrix of the Sylvester matrix $S(f, g)$ whose determinant is σ_k for $0 \leq k \leq n_1$.

(1) Let $\mathcal{K}_i$ and λ_i be the constant coefficients of $s_i = \alpha_i^{-1} s_i^*$ and $t_i = \alpha_i^{-1} t_i^*$ respectively for $2 \leq i \leq l$. We have:

(i) $\mathcal{K}_i = \dfrac{\det(Y_i)}{\sigma_{n_i}}$, $\lambda_i = \dfrac{\det(Z_i)}{\sigma_{n_i}}$ where Y_i, Z_i are matrices that result from S_{n_i} by replacing a certain column by a unit vector.

(ii) $\alpha_2 = \dfrac{\sigma_{n_2}}{\gamma_2}$ and $\alpha_i = \dfrac{(-1)^{i-1}\sigma_{n_i}\sigma_{n_{i-1}}}{\gamma_i \alpha_{i-1}}$ where $\gamma_2 = \det(Y_2)$ and

$$\gamma_i = \det(Y_{i-1})\det(Z_i) - \det(Z_{i-1})\det(Y_i)$$

for $3 \leq i \leq l$.

(iii) $\alpha_i = (-1)^{\frac{(i+1)(i+2)}{2}} \sigma_{n_i} \displaystyle\prod_{j=2}^{i} \gamma_j^{(-1)^{i+j-1}}$ for $2 \leq i \leq l$.

(2) Let F be a field and let $f, g \in F[x, y]$ have degrees at most d in y and $\deg(f, x) = n \geq \deg(g, x) = m$ and $2 \leq i \leq l$. The results r_i^*, q_i^*, s_i^* of the Extended Euclidean Algorithm for f and g in $F(y)[x]$ have numerators and denominators (in the lowest terms) of degree in y at most $(n+m)d$. The numerator and denominator of the leading coefficient α_i has degree in y of at most $i(n+m)d$.