A new Dixon resultant algorithm for solving parametric polynomial systems using sparse multivariate rational function interpolation

Ayoola Jinadu[a,b], Michael Monagan[a]

[a] *Department of Mathematics, Simon Fraser University, Burnaby, V5A 1S6, B.C., Canada*
[b] *Department of Mathematics, Alexander College, Burnaby, V5H 4T6, B.C., Canada*

Abstract

Elimination methods 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 when used in Computer Algebra Systems such as Maple and Magma they often struggle with systems that have many parameters. To address this problem, we present a new interpolation algorithm for solving parametric polynomial systems over $\mathbb{Q}$ using the Dixon resultant. The Dixon resultant is a multiple of the unique generator of an elimination ideal of a polynomial system. It can be expressed as the determinant of a matrix of polynomial entries called the Dixon matrix.

In practice, the Dixon resultant R often has many repeated factors and a large polynomial content. To avoid computing these unwanted factors of R, our new algorithm interpolates the rational function coefficients in the parameters of the monic square-free factors of R from monic univariate images of R. It does this using our newly developed sparse multivariate rational function interpolation method. By not computing R in expanded form, our approach often dramatically reduces the number of images needed to interpolate the monic square-free factors of R over $\mathbb{Q}$.

We have implemented our new Dixon resultant algorithm in Maple with many parts of the algorithm coded in C for increased efficiency. 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. Our new algorithm is probabilistic; it may fail to produce an answer, and even when successful, it may return an incorrect answer, but with provably low probability. In this work we identify and classify all the causes of failure in our new algorithm, and we give a detailed failure probability analysis.

Keywords: Black box, Parametric Polynomial system, Dixon Resultants, Sparse Multivariate Rational Function Interpolation, Kronecker Substitution

1. Introduction

Solving parametric polynomial systems has become increasingly important in many fields such as computer vision, robotics, physics, and control theory. Various elimination techniques including Gröbner bases [6] and triangular sets [2] have been used to solve these systems. However, experiments by Lewis in [26, 27, 28] showed that when these methods are used to solve parametric polynomial systems arising from practical applications in Computer Algebra Systems such as Maple and Magma, they often fail on systems with many parameters. They can take a very long time to execute or they run out of memory due to the intermediate expression swell caused by the parameters.

To address this problem, we study the Dixon resultant method [8, 9], a determinant approach for solving polynomial systems which eliminates $n-1$ variables from n polynomial equations in n variables. The Dixon resultant method is recognized as the most efficient method of all known resultant methods. Notably, comparison results from [14] show that the dimension of the Dixon matrix (see Section 2) and the size of the polynomial entries in it are smaller compared to the resultant matrices produced by the Macaulay and sparse resultant methods.

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 $|\mathcal{F}| = |X| = n$. 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 $I = \langle \hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n \rangle$ be the ideal generated by $\mathcal{F}$ and let $J = I \cap \mathbb{Q}(Y)[x_1]$ be the elimination ideal in $\mathbb{Q}(Y)[x_1]$. The Dixon resultant R of $\mathcal{F}$ in x_1 is a multiple of the unique generator of J [8, 9, 13], and it can be expressed as a determinant of a matrix of polynomial entries in $x_1, y_1, \cdots, y_m$ called the Dixon matrix (see Section 2). Suppose

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

where $\hat{d} = \deg(R, x_1)$. If $\hat{d} = 0$, then $\mathcal{F}$ 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 R.

In practice, when R factors over $\mathbb{Q}$, it often has many repeated factors with large degrees and a large polynomial content C. To avoid computing these unwanted polynomial factors, we do not try to compute R. Instead, we compute the monic square-free factors R_j of R. The monic square-free factorization of R is a factorization of the form $\hat{r} \prod_{j=1}^{l} R_j^j$ where each R_j can be written as

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

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, that is, $\gcd(R_j, \partial R_j/\partial x_1) = 1$,

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. the leading coefficient of g_{jk} denoted by $\mathrm{LC}(g_{jk}) = 1$.

This monic square-free factorization exists and it is unique [15, Section 14.6]. We remark that the monic square-free factors R_j of R are not necessarily irreducible in $\mathbb{Q}(Y)[x_1]$. To give the reader an idea of what we are computing, we give the following real example from [26, Section 8, page 247].

Example 1. *The robot arms system* $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4\} \subset \mathbb{Q}[Y][X]$ *where* $X = \{x_1, x_2, x_3, x_4\}$ *and* $Y = \{y_1, y_2, \ldots, y_7\}$. *The four polynomials* $\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4$ *are listed in Appendix A. Let*

$$
\begin{aligned}
C &= -65536\left(y_2^2+1\right)^8\left(y_2^2y_4^2+2y_2^2y_4y_5+y_2^2y_5^2+y_4^2-2y_4y_5+y_5^2\right)^4 y_4^8\\
A_1 &= x_1^2+1\\
A_2 &= (y_2^2y_3^2+2y_2^2y_3y_6-y_2^2y_4^2-2y_2^2y_4y_5-y_2^2y_5^2+y_2^2y_6^2+y_2^2y_7^2+y_3^2+2y_3y_6\\
&\quad -y_4^2+2y_4y_5-y_5^2+y_6^2+y_7^2)x_1^2+(-4y_2^2y_3y_7-4y_3y_7)x_1+y_2^2y_3^2-y_4^2\\
&\quad -2y_2^2y_3y_6-y_2^2y_4^2-2y_2^2y_4y_5-y_2^2y_5^2+y_2^2y_6^2+y_2^2y_7^2+y_3^2-2y_3y_6\\
&\quad +2y_4y_5-y_5^2+y_6^2+y_7^2\\
A_3 &= \left(y_1^2+2y_1y_4\right)x_1^2+y_1^2-4y_1y_3+2y_1y_4+4y_3^2-4y_3y_4\\
A_4 &= \left(y_1^2-2y_1y_4\right)x_1^2+y_1^2-4y_1y_3-2y_1y_4+4y_3^2+4y_3y_4.
\end{aligned}
$$

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

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

Our new Dixon resultant algorithm interpolates R_1, R_2 and R_3 where $R_1 = A_1$, $R_2 = \mathrm{monic}(A_2, x_1)$ and $R_3 = \mathrm{monic}(A_3A_4, x_1)$ where $\mathrm{monic}(A, x_1) = A/\mathrm{LC}(A, x_1)$ means that A is monic in x_1. The largest polynomial coefficients of R_1, R_2 and R_3 to be interpolated by our algorithm are the coefficients of A_2 of degree 0 and 2 in x_1 (shown in blue) which both have only 14 terms. For comparison, Kapur, Saxena and Yang [14] use Zippel's algorithm [37] to interpolate R which has **6,924,715** terms. Note that R_1 and R_2 are irreducible over $\mathbb{Q}(y_1, y_2, \ldots, y_7)$ but R_3 is not.

1.1. Modular Black box model for R

A black box is a device or a computer program in which its inputs and output are known, but the internal functionality is unknown. The black box model was first studied in Computer Algebra by Kaltofen and Trager in [23] who gave algorithms for computing the greatest common divisor of two polynomials

represented by two black boxes and factoring a polynomial given by a black box. A black box can be constructed for a polynomial, a rational function, and a parametric linear system. A function call to the black box is referred to as a **black box** probe.

For our purposes, we assume we can modify a given black box that works over $\mathbb{Z}$ or $\mathbb{Q}$ to work modulo a prime p. We do this to improve the efficiency of evaluating the black box at a point to avoid large integer or rational number arithmetic. Figure 1 depicts a modular black box representation of $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, where p is prime and $\alpha \in \mathbb{Z}_p^n$ is an evaluation point.

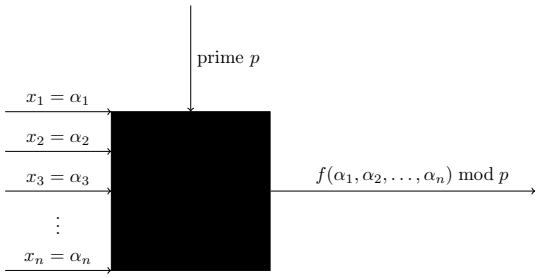

Figure 1: Modular Black box model for $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$

In this work, the modular black box for our proposed Dixon resultant algorithm denoted by **BB** will represent a determinant computation. We construct the modular black box **BB** from a Dixon matrix D of polynomials in $x_1, y_1, y_2, \ldots, y_m$ over $\mathbb{Z}$ 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 then input $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ to our proposed Dixon resultant algorithm which treats it as a black box.

1.2. Overview of our Dixon resultant algorithm

Given the modular black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for the Dixon resultant R, our new Dixon resultant algorithm probes **BB** at many points to interpolate the monic square-free factors R_j one at a time. It does not interpolate R. Our new algorithm interpolates the R_j from monic univariate images of R in x_1 using our new sparse multivariate rational function interpolation method to interpolate the rational function coefficients of the monic square-free factors R_j in $\mathbb{Q}(Y)$ modulo primes. It then uses Chinese remaindering and rational number reconstruction [15, 30] to recover the rational coefficients of R_j.

We interpolate the monic square-free factors because it is (1) cheap to compute a square-free factorization of a monic image of R in $\mathbb{Z}_p[x_1]$, (2) the square-free factorizations will have the same degree pattern with high probability, and (3) the square-free factors of R will often be smaller than the square-free part of R. Interpolating the R_j's instead of R often results in a huge gain because all unwanted repeated factors and the polynomial content are avoided. This is

the main advantage of our algorithm over using polynomial interpolation algorithms such as the Ben-Or/Tiwari algorithm [4] and Zippel's algorithm [37] for interpolating R. Furthermore, the number of primes used by our algorithm to recover the rational coefficients of R_j using Chinese remaindering and rational number reconstruction is also reduced. The number of polynomial terms in the R_j to be interpolated is much less than in R so the number of black box probes required to interpolate the R_j is often much fewer than the number required to interpolate R.

To interpolate the rational function coefficients of the R_j, we have developed a new sparse multivariate rational function interpolation method for our Dixon resultant algorithm. We modify the sparse multivariate rational function interpolation algorithm of Cuyt and Lee [7] and the Ben-Or/Tiwari algorithm [4] to use a Kronecker substitution on the parameters $y_1, y_2, \ldots, y_m$ and a new set of randomized evaluation points.

Previous methods for computing Dixon resultants

To the best of our knowledge, the work of Kapur, Saxena and Yang in 1995 [14] is the only previous attempt to interpolate the Dixon resultant R. The authors used Zippel's sparse interpolation [37]. Zippel's algorithm makes $O(\hat{D}t)$ probes to the black box **BB** for the first image modulo a prime where $\hat{D} = \deg(R, x_1) + \sum_{i=1}^{m} \deg(R, y_i)$, m is the number of parameters and t is the number of terms of R. To recover the integer coefficients of R using Chinese remaindering, one can use the support of the result obtained for the first prime for the subsequent primes. Zippel's algorithm uses $O(t)$ probes for each subsequent prime.

In 2015, in [26], Lewis developed the Dixon-EDF (Early Detection Factor) algorithm for computing the Dixon resultant R. This algorithm is a variant of the Gaussian elimination algorithm. It is a modified row reduction of the Dixon matrix that factors out the gcd of each pivot row at each elimination step. The Dixon-EDF method is able to detect factors of the Dixon resultant R early. However, if there are many parameters, a severe expression swell may occur when computing in $\mathbb{Q}[Y][x_1]$. If this happens Lewis often switches to the Gentleman & Johnson minor expansion algorithm [12] to try to finish the computation. The implementation of the Dixon-EDF algorithm was done in Fermat; a Computer Algebra System designed and implemented by Lewis whose built-in multivariate gcd algorithm uses Zippel's gcd algorithm [25]. In 2017, another variant of the Dixon-EDF algorithm was designed and implemented in Maple by Minimair [32] which requires fewer gcd computations than Lewis' Dixon-EDF method.

New Contributions

This paper is the full version of our preliminary work [19] which was presented and published in the proceedings of CASC 2022. It was also presented at the ISSAC 2022 poster session and was subsequently published as an extended abstract [20]. However, the failure probability analysis and the complexity analysis of the algorithm in terms of the number of black box probe were left for

future work. The referees for the CASC paper asked for the failure probability analysis and one of the ISSAC referees asked us to compare our Dixon resultant algorithm with an implementation of the Dixon-EDF algorithm. In comparison with our previous works [19, 20], the Dixon resultant algorithm presented in this paper is an improved version. We have redesigned our algorithm to pre-compute certain degree bounds for speed up (see Subsection 4.3.2 for benchmarks). The failure probability analysis of our algorithm, the benchmarks and timing results data are new. Algorithm 8 which uses the support of the first image of the R_j's modulo the first prime to get new images when additional primes are required is new. It uses at most 50% of the total number of black box probes used for the first prime to get a new image.

Our Maple and C code is freely available for download at: `https://www.cecm.sfu.ca/personal/monaganm/code/DixonRes/DixonRes/`. There you will also find the polynomial systems we use in our benchmarks.

Paper Outline

We review the Dixon resultant formulation for solving polynomial systems in Section 2, and we present some new results which include Theorems 8, 14 and 16. In Section 3, we give an overview of the rational function interpolation algorithm of Cuyt and Lee [7] and the Ben-Or/Tiwari sparse polynomial algorithm [4]. Then we discuss how we modify these algorithms to use a Kronecker substitution and a new randomized evaluation point sequence to adress both the large prime problem and the issue of unlucky evaluation points that occur when the adopted sparse polynomial algorithm in Cuyt and Lee's method is the Ben-Or/Tiwari sparse polynomial algorithm [4]. Our Dixon resultant algorithm is presented in Section 4. We also compare our new Dixon resultant algorithm with the Gentleman and Johnson minor expansion algorithm, Lewis' Dixon-EDF algorithm, and a hybrid Maple and C implementation of Zippel's algorithm to interpolate R on real parametric polynomial systems that emerged from practical applications. A detailed failure probability analysis and the complexity analysis of our Dixon resultant algorithm in terms of the number of black box probes used is presented in Section 5.

1.3. Some Useful Results

Many of the proofs in this paper require the use of the Schwartz-Zippel Lemma [33, 37]. We state the lemma and some useful results now.

Lemma 2 (Schwartz-Zippel Lemma)**.** *Let $\mathbb{K}$ be a field and let S be a finite subset of $\mathbb{K}$. Let f be a non-zero polynomial in $\mathbb{K}[y_1, y_2, \cdots, y_m]$. If α is chosen at random from S^m then* $\Pr[f(\alpha) = 0] \leq \frac{\deg(f)}{|S|}$.

Definition 3. *Let $f = \sum_{i=1}^{t} a_i N_i \in \mathbb{Z}[y_1, y_2, \cdots, y_m]$ where the coefficients a_i are non-zero in $\mathbb{Z}$, N_i is a monomial in variables $y_1, y_2, \cdots, y_m$. Let $t = \#f$ denote the number of terms in f and let* $\mathrm{supp}(f) = \{N_i : 1 \leq i \leq t\}$. *The height*

of f denoted by $\|f\|_\infty$ is defined as $\|f\|_\infty = \max_{i=1}^{t}|a_i|$. Let

$$H = \sum_{k=0}^{d_T} \frac{f_k(y_1, y_2, \cdots, y_m)}{g_k(y_1, y_2, \cdots, y_m)} x_1^k$$

where the f_k and g_k are polynomials in $\mathbb{Z}[y_1, y_2, \cdots, y_m]$. We also define $\|H\|_\infty = \max_{k=0}^{d_T}(\|f_k\|_\infty, \|g_k\|_\infty)$ and $\#H = \sum_{k=0}^{dT}(\#f_k + \#g_k)$.

Theorem 4. *[17, Proposition 2] Let A be a $t \times t$ matrix with $A_{ij} \in \mathbb{Z}[y_1, \cdots, y_m]$, $\#A_{ij} \leq N$ and $\|A_{ij}\|_\infty \leq h$. Then $\|\det(A)\|_\infty < t^{\frac{t}{2}} N^t h^t$.*

Lemma 5. *[11, Lemma 2, page 135] Let $f, g \in \mathbb{Z}[y_1, y_2, \cdots, y_m]$. If $g|f$ then $\|g\|_\infty \leq \mathrm{e}^{\sum_{i=1}^{m} \deg(f, y_i)} \|f\|_\infty$ where $\mathrm{e} \approx 2.718$ is the Euler number. Note, for almost all polynomials if $g|f$ we will have $\|g\|_\infty \leq \|f\|_\infty$.*

2. Dixon Resultants

Let $\hat{f}$ be a polynomial in $x_1, x_2, \ldots, x_n$. Let $\{\bar{x}_2, \cdots, \bar{x}_n\}$ be the set of new variables corresponding to $x_2, \cdots, x_n$ respectively. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where each $\alpha_i \geq 0$. Let $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ and let $\pi_1(\mathbf{x}^\alpha) = \mathbf{x}^\alpha$. Fix $i \geq 2$ and let x_1 be the main variable. Let

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

The evaluation map π_i can be extended naturally to polynomials as

$$\pi_i(\hat{f}(x_1, x_2, \cdots, x_n)) = \hat{f}(x_1, \underbrace{\bar{x}_2, \ldots, \bar{x}_i}_{i-1 \text{ variables}}, x_{i+1}, x_{i+2} \ldots, x_n).$$

Observe that the evaluation map π_i replaces the $(i-1)$ variables $x_2, \ldots, x_i$ in $\hat{f}$ with the new variables $\bar{x}_2, \ldots \bar{x}_i$, so x_1 is never affected.

There are four major steps involved in computing the Dixon resultant of a given parametric system polynomial $\mathcal{F}$. Our presentation follows Kapur [13].

2.1. Step 1: Constructing the Cancellation Matrix $\mathcal{C}$.

Definition 6. *Given a parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$, let $X_e = \{x_2, \cdots, x_n\}$ be the set of variables to be eliminated from $\mathcal{F}$ and let x_1 be the main variable to appear in the Dixon resultant R. 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*

$$\mathcal{C} = \begin{pmatrix} \pi_1(\hat{f}_1) & \pi_1(\hat{f}_2) \ldots & \pi_1(\hat{f}_n) \\ \pi_2(\hat{f}_1) & \pi_2(\hat{f}_2) \ldots & \pi_2(\hat{f}_n) \\ \vdots & \vdots & \vdots \\ \pi_n(\hat{f}_1) & \pi_n(\hat{f}_2) \ldots & \pi_n(\hat{f}_n) \end{pmatrix}. \tag{1}$$

Definition 7. *Let*

$$\Delta_{X_e} = \frac{\det(\mathcal{C})}{\prod_{i=2}^{n}(x_i - \bar{x}_i)} \in \mathbb{Q}[Y, x_1][X_e, \overline{X}_e]. \tag{2}$$

We refer to Δ_{X_e} as the Dixon polynomial of $\mathcal{F}$ with respect to X_e.

Notice that $\det(\mathcal{C})$ is a multiple of Δ_{X_e}. Thus, if the number of variables n is large, and since there are 2^{n-1} terms in $\prod_{i=2}^{n}(x_i - \bar{x}_i)$ when expanded, then computing Δ_{X_e} using (2) will 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.

To compute Δ_{X_e} we do not use (2). Instead, we use an idea communicated to us by Lewis [29]. We construct a matrix $\hat{\mathcal{C}}$ from $\mathcal{C}$ as follows. Define $\mathrm{Row}_1(\hat{\mathcal{C}}) = \mathrm{Row}_1(\mathcal{C})$ and

$$\mathrm{Row}_j(\hat{\mathcal{C}}) = \frac{\mathrm{Row}_j(\mathcal{C}) - \mathrm{Row}_{j-1}(\mathcal{C})}{x_j - \bar{x}_j} \text{ for } j = 2, 3, \ldots n \tag{3}$$

where $\mathrm{Row}_j(\mathcal{C})$ is the j-th row of $\mathcal{C}$. To obtain a tight height bound for $\|\Delta_{X_e}\|_\infty$ in Theorem 14, Theorem 8 avoids the polynomial divisions by $x_j - \bar{x}_j$ in (3).

Theorem 8. *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 $d_j = \max_{\hat{f} \in \mathcal{F}} \deg(\hat{f}, x_j)$. Then for $j = 2, 3, \ldots, n$, and $k = 1, 2, \ldots, n$,*

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

$$\det(\hat{\mathcal{C}}) = \Delta_{X_e} = \frac{\det(\mathcal{C})}{\prod_{i=2}^{n}(x_i - \bar{x}_i)}.$$

(ii) Furthermore, by expressing

$$\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) = \sum_{u=0}^{d_j} \tilde{f}_{u,j,k} x_j^u, \tag{4}$$

where for $u \neq 0$, $\tilde{f}_{u,j,k} \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_{j+1}, \ldots, x_n]$ and $\tilde{f}_{0,j,k} \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n]$, we obtain

$$\hat{\mathcal{C}}_{j,k} = \frac{\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)}{x_j - \bar{x}_j} = \sum_{i=0}^{d_j - 1} \left(\sum_{u=i}^{d_j - 1} \tilde{f}_{u+1,j,k}\ x_j^{u-i} \right) \bar{x}_j{}^i. \tag{5}$$

Proof. For $2 \leq j \leq n$, observe that $(x_j - \bar{x}_j)$ divides $\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k) = \hat{f}_k(x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, x_{j+2} \ldots, x_n) - \hat{f}_k(x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_j, x_{j+1} \ldots, x_n)$. Thus,

the entries $\hat{\mathcal{C}}_{j,k} = \frac{\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)}{x_j - \bar{x}_j}$ are polynomials. Now let

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

where the i-th row of $\mathcal{E}$ denoted by $\mathrm{Row}_i(\mathcal{E}) = \mathrm{Row}_i(\mathcal{C}) - \mathrm{Row}_{i-1}(\mathcal{C})$. It follows that $\det(\mathcal{E}) = \det(\mathcal{C})$. Next, since $x_j - \bar{x}_j$ divides $\pi_j(\hat{f}_k) - \pi_{j-1}(\hat{f}_k)$, 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 $\hat{\mathcal{C}}_{j,k} \in \mathbb{Z}[y_1, \dots, y_m][x_1, \bar{x}_2, \dots, \bar{x}_j, x_j, x_{j+1}, \dots, x_n]$. So matrix $\mathcal{E}$ becomes

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

Therefore, $\det(\mathcal{E}) = \det(\hat{\mathcal{C}}) \prod_{j=2}^{n}(x_j - \bar{x}_j)$. Using (2), it follows that

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

This completes part (i). For the proof of part (ii), we recall the formal power series representation of $(x_j - \bar{x}_j)^{-1}$ when expanded about $\bar{x}_j$ is given by

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

Using (4), it follows that

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

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

$$\hat{\mathcal{C}}_{j,k} = \sum_{u=0}^{d_j} \tilde{f}_{u,j,k} x_j^u \left(\sum_{i=1}^{u} x_j^{-i} \bar{x}_j{}^{i-1} \right). \tag{6}$$

Observe that

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

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,

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

Thus, (6) becomes

$$\hat{\mathcal{C}}_{j,k} = \sum_{u=1}^{d_j} \tilde{f}_{u,j,k} \left(\sum_{i=0}^{u-1} x_j^i \ \bar{x}_j{}^{u-i-1} \right). \tag{7}$$

By expanding (7) and rearranging the terms in powers of $\bar{x}_j$, we get

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

□

Remark 9. *Simplifying* (6) *to get* (5) *yields a tighter bound for* $\|\Delta_{X_e}\|_\infty$.

2.2. Step 2: Constructing the Dixon Matrix from the Dixon Polynomial

The second step is to build the Dixon matrix D from the Dixon polynomial Δ_{X_e}. To do this, we first need degree bounds for Δ_{X_e} in x_i and $\bar{x}_i$ using (2). Let $d_i = \max_{\hat{f}\in\mathcal{F}} \deg(\hat{f}, x_i)$. Since the evaluation map π_i does not affect x_1, we have that $\deg(\Delta_{X_e}, x_1) \leq nd_1$. Notice that $\deg(\Delta_{X_e}, x_2) \leq d_2 - 1$ and $\deg(\Delta_{X_e}, \bar{x}_2) \leq (n-1)d_2 - 1$ because x_2 is replaced with $\bar{x}_2$ from row 2 to row n of matrix $\mathcal{C}$ and we have to do a division by $x_2 - \bar{x}_2$ from $\prod_{i=2}^{n}(x_i - \bar{x}_i)$ in (2). Following the same reasoning, for $2 \leq i \leq n$, it follows that

$$\deg(\Delta_{X_e}, x_i) \leq (i-1)d_i - 1 \tag{8}$$

and

$$\deg(\Delta_{X_e}, \bar{x}_i) \leq (n-i+1)d_i - 1. \tag{9}$$

Let $\overline{V}$ be a monomial column vector in $\overline{X}_e = \{\bar{x}_2, \bar{x}_3 \cdots, \bar{x}_n\}$ when Δ_{X_e} is viewed as a polynomial in $\overline{X}_e$ and let V be a monomial row vector in $X_e = \{x_2, x_2, \ldots, x_n\}$ when Δ_{X_e} is viewed as a polynomial in X_e. Notice that

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

because $|\overline{X}_e| = n-1$ and the maximum number of possible monomials that appears in $\overline{V}$ in $\bar{x}_i$ including the constant term 1 is at most $(n-i+1)d_i$. Following the same argument as before, one can see that

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

Lemma 10. *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*. In bilinear form,* Δ_{X_e} *can be written as*

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

such that D *is a* $s \times t$ *matrix with* $t, s \leq (n-1)! \prod_{i=2}^{n} d_i$ *and* $D_{ij} \in \mathbb{Q}[Y, x_1]$.

Definition 11. *The* $s \times t$ *matrix* D *in the above lemma is called the Dixon matrix and* $R = \det(D) \in \mathbb{Q}[Y, x_1]$ *is the Dixon resultant if* $s = t$ *and* $R \neq 0$.

Example 12. *Let* $\mathcal{F} = \{x_2^2 + x_3^2 - y_3^2,\ (x_2 - y_1)^2 + x_3^2 - y_2^2,\ -x_3y_1 + 2x_1\}$. *Let* $X_e = \{x_2, x_3\}$ *be the variables to be eliminated from* $\mathcal{F}$ *and let* $\overline{X}_e = \{\bar{x}_2, \bar{x}_3\}$ *be the new variables corresponding to* X_e*. Using our new formula* (5)*, we get*

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

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

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

Finally, the Dixon resultant

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

2.3. Step 3: Extracting a sub-matrix M of maximal rank from the Dixon matrix

In practice, the Dixon matrix D is often rectangular. This is evident from our bounds for the dimensions of D in Lemma 10. Also, when D is square, $R = \det(D)$ is often 0, thus providing no information about the solutions of $\mathcal{F}$. These problems were addressed by Kapur, Saxena and Yang in [13]. They proved that the determinant of any square sub-matrix M of D with $\text{rank}(M) = \text{rank}(D)$ is an element of the elimination ideal $I \cap \mathbb{Q}(Y)[x_1]$. Thus, once the Dixon matrix D is constructed, the third step is to identify M.

Our Probabilisitic Approach

In this paper, we select a square sub-matrix M of maximal rank from D as follows. We pick a 62 bit prime p and choose an evaluation point $\beta \in \mathbb{Z}_p^{m+1}$ at random. Then we compute $B = D(\beta)$ and identify a square sub-matrix of maximal rank from B in the Dixon matrix D. This requires doing Gaussian elimination over $\mathbb{Z}_p$ only and in contrast to [13] crucially avoids doing polynomial arithmetic in $\mathbb{Q}[Y, x_1]$. However, the evaluation point β or the input prime may result in the selection of a sub-matrix M with $\text{rank}(M) < \text{rank}(D)$. Example 13 illustates this failure and Theorem 16 bounds the failure probability.

Example 13. *Consider the matrices D and M below.*

$$D = \begin{pmatrix} x_1^2y_1 & 1 & 0 & 0 & 0 \\ 2x_1y_1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 5 & x_1^2 & 0 \\ 7 & 8 & 10 & 4x_1 & 0 \end{pmatrix} \quad M = \begin{pmatrix} x_1^2y_1 & 1 & 0 & 0 \\ 2x_1y_1 & 2 & 0 & 0 \\ 1 & 0 & 5 & x_1^2 \\ 7 & 8 & 10 & 4x_1 \end{pmatrix}.$$

It is not hard to see M is a submatrix of D of maximal rank as $\det(M) = -20x_1^4y_1 + 60x_1^3y_1 - 40x_1^2y_1$. *Since* $\det(M)$ *has a root $x_1 = 2, y_1 = 3$, for any prime p used, if $\beta = (2, 3)$ is selected, then an incorrect sub-matrix*

$$\begin{pmatrix} x_1^2y_1 & 1 & 0 \\ 2x_1y_1 & 2 & 0 \\ 1 & 0 & 5 \end{pmatrix}$$

would be chosen whose rank is less than $\text{rank}(D) = 4$.

2.4. Step 4: Computing $\det(M)$ *the Dixon resultant*

The final step is to compute $\det(M)$ where M is an s by s matrix of polynomials in the ring $\mathbb{Z}[x_1, y_1, \ldots, y_m]$. The Bareiss-Edmonds fraction free algorithm [3, 10] is implemented in Maple and many Computer Algebra Systems. It is a fraction-free variation of Gaussian elimination which does $O(s^3)$ multiplications and exact divisions in the ring $\mathbb{Z}[x_1, y_1, \ldots, y_m]$. We do not use it because a severe expression swell occurs when m is large. In [12] Gentleman and Johnson compared minor expansion with Bareiss-Edmonds on polynomial matrices. Even though minor expansion does $O(s2^s)$ ring operations, it is much faster than Bareiss-Edmonds when m is not small. We have implemented a sparse variation of Gentleman-Johnson in Maple. We find it is effective for small matrices and modestly sized sparse matrices on our benchmarks in Section 4. Another division free algorithm is the Berkowitz algorithm [5] which does $O(s^4)$ ring operations. We implemented a sparse version of it in Maple. It performed poorly on our benchmarks. We also implemented Lewis' Dixon-EDF algorithm [26] in Maple. It performs well on many of our benchmarks.

2.5. Height and Degree Bounds

We derive some degree and height bounds for the Dixon resultant R and its monic square-free factors which we will interpolate. Let the parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$. Let $H = \max_{\hat{f}\in\mathcal{F}} \|\hat{f}\|_\infty$, $N = \max_{\hat{f}\in\mathcal{F}} \#\hat{f}$, $d_x = \max_{i=1}^{n}(\max_{\hat{f}\in\mathcal{F}} \deg(\hat{f}, x_i))$ and $D_y = \max_{i=1}^{m}(\max_{\hat{f}\in\mathcal{F}} \deg(\hat{f}, y_i))$. Let D be the rectangular Dixon matrix obtained by Step 2 from $\mathcal{F}$ with $D_{ij} \in \mathbb{Z}[x_1, y_1, y_2, \ldots, y_m]$. Let $s = \text{rank}(D)$, $t = \max_{i,j}(\#D_{ij})$, and $D_{max} = \max_{i,j}(\deg(D_{ij}))$. These parameters appear in the next two theorems and in Section 5.

Theorem 14. *Let* M *be any* $s \times s$ *sub-matrix of* D *with* $\text{rank}(M) = s$. *Let*

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

be the Dixon resultant and suppose its monic square-free factors are

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

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* $\hat{d} > 0$. *Then*

(i) $\deg(R, x_1) \leq nsd_x$.

(ii) $\deg(R, y_k) \leq nsD_y$ *for* $1 \leq k \leq m$.

(iii) $\|\Delta_{X_e}\|_\infty \leq n^{\frac{n}{2}} H^n N^n$ *where* Δ_{X_e} *is the Dixon polynomial.*

(iv) $\|R\|_\infty \leq t^s n^{\frac{n}{2}} (HN)^{ns} s^{\frac{s}{2}}$.

(v) $\|R_j\|_\infty \leq \mathrm{e}^{nsd_x + 2nmsD_y} \|R\|_\infty$ *where* $\mathrm{e} \approx 2.718$ *is the Euler number.*

Proof. For claim (i) and(ii), we have

$$\deg(R, x_1) \leq s \times \max_{1 \leq i,j \leq s} \{\deg(M_{ij}, x_1)\} \leq s \times \deg(\Delta_{X_e}, x_1) \leq nsd_x,$$

and

$$\deg(R, y_k) \leq \ s \times \max_{1 \leq i,j \leq s} \{\deg(M_{ij}, y_k)\} \leq s \times \deg(\Delta_{X_e}, y_k) \leq nsD_y.$$

We prove claim (iii) using the formula (5) derived in Theorem 8 for creating the new cancellation matrix $\hat{\mathcal{C}}$. Recall that

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

where $\tilde{f}_{0,j,k} \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, \bar{x}_2, \ldots, \bar{x}_j, x_{j+1}, \ldots, x_n]$ and for $u \neq 0$, $\tilde{f}_{u,j,k} \in \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, \bar{x}_2, \ldots, \bar{x}_{j-1}, x_{j+1}, \ldots, x_n]$. Since $\tilde{f}_{u,j,k}$ does not contain variables x_j and $\bar{x}_j$ for $u \neq 0$, we get

$$\|\hat{\mathcal{C}}_{j,k}\|_\infty \leq \|\tilde{f}_{u,j,k}\|_\infty \leq \|\hat{f}_k\|_\infty \leq H. \tag{10}$$

Now, using Theorem 4, we have

$$\|\Delta_{X_e}\|_\infty \leq \|\det(\hat{\mathcal{C}})\|_\infty \leq n^{\frac{n}{2}} \|\hat{\mathcal{C}}_{ij}\|_\infty^n N^n \leq n^{\frac{n}{2}} H^n N^n. \tag{11}$$

Since $R = \det(M)$ and $\|M_{ij}\|_\infty \leq \|\Delta_{X_e}\|_\infty$, it follows that

$$\|R\|_\infty \leq \left(t \, \|M_{ij}\|_\infty \sqrt{s}\right)^s \leq t^s n^{\frac{n}{2}} (HN)^{ns} s^{\frac{s}{2}}$$

by Theorem 4. This proves claim (iv). Finally, we prove claim (v).

Suppose we clear the fractions of $R_j = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \dfrac{f_{jk}(y_1, y_2, \ldots, y_m)}{g_{jk}(y_1, y_2, \ldots, y_m)} x_1^{d_{j_k}}$ in $y_1, y_2, \ldots, y_m$. Let $L_j = \mathrm{LCM}\{g_{jk} \in \mathbb{Z}[x_1, y_1, \ldots, y_m] : 0 \leq k \leq T_j - 1\}$ be the least common multiple of the g_{jk} and let $H_j = L_j R_j \in \mathbb{Z}[x_1, y_1, \ldots, y_m]$ such that $H_j = \sum_{k=0}^{T_j} a_{jk}(y_1, \ldots, y_m) x_1^{d_{j_k}}$. Since $H_j | R$, by Lemma 5, we get

$$\|H_j\|_\infty \leq \mathbf{e}^{\deg(R, x_1) + \sum_{k=1}^m \deg(R, y_k)} \|R\|_\infty \leq \mathbf{e}^{nsd_x + nmsD_y} \|R\|_\infty.$$

Observe that $\dfrac{H_j}{L_j} = x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \dfrac{a_{jk}(y_1, \ldots, y_m)}{L_j} x_1^{d_{j_k}}$. Let $h_{jk} = \gcd(a_{jk}, L_j)$.

Observe that

$$\frac{a_{jk}/h_{jk}}{L_j/h_{jk}} = \frac{f_{jk}}{g_{jk}}.$$

So, $f_{jk}|a_{jk}$ and $g_{jk}|L_j \Longrightarrow g_{jk}|\mathrm{LC}(R, x_1)$. Using Lemma 5, we get

$$\|f_{jk}\|_\infty \leq \mathbf{e}^{nmsD_y}\|a_{jk}\|_\infty \leq \mathbf{e}^{nmsD_y}\|H_j\|_\infty \leq \mathbf{e}^{nsd_x+2nmsD_y}\|R\|_\infty$$

and

$$\|g_{j,k}\|_\infty \leq \mathbf{e}^{nmsD_y}\|\mathrm{LC}(R, x_1)\|_\infty \leq \mathbf{e}^{nmsD_y}\|R\|_\infty.$$

Therefore, $\|R_j\|_\infty \leq \max_{k=0}^{T_j-1}(\max(\|f_{jk}\|_\infty, \|g_{jk}\|_\infty)) \leq \mathbf{e}^{nsd_x+2nmsD_y}\|R\|_\infty$.

□

Remark 15. *The height bound for $\|R_j\|_\infty$ obtained in Theorem 14(v) is a worst case bound because $\|R_j\|_\infty$ is always smaller than $\|R\|_\infty$ in our experiments. It is rare for the factors of R to have larger coefficients than R.*

2.6. Failure Probability

We give a failure probability bound for our probabilistic approach in Step 3 to extract a submatrix M of maximal rank from the Dixon matrix D.

Theorem 16. *Let D be the rectangular Dixon matrix obtained from $\mathcal{F}$ in Step 2 with $s = \mathrm{rank}(D)$. Let p be a random prime selected from a list of pre-computed primes $\mathbb{P}$ and let $p_{\min} = \min(\mathbb{P})$. Let β be an evaluation point chosen at random from $\mathbb{Z}_p^{m+1}$ and let $B = D(\beta)$. Then*

$$\Pr[\mathrm{rank}(B) < s] \leq \frac{\log_{p_{\min}}|t^s n^{\frac{n}{2}}(HN)^{ns}s^{\frac{s}{2}}|}{|\mathbb{P}|} + \frac{sD_{\max}}{p}.$$

Proof. Let M be a $s \times s$ sub-matrix of D with $\mathrm{rank}(M) = s$. Since M has full rank, $\det(M) \neq 0$, thus $\deg(\det(M)) \leq sD_{\max}$. To extract M from B in D, we compute B and then perform row operations on B over $\mathbb{Z}_p$ using Gaussian elimination. Thus, an incorrect sub-matrix of D is obtained if $\mathrm{rank}(B) < s \Longrightarrow \det(M(\beta)) = 0$ or $\mathrm{rank}(D \bmod p) < s$. Using Lemma 2, we have

$$\begin{aligned}
\Pr[\mathrm{rank}(B) < \mathrm{rank}(D)] &\leq \Pr[\mathrm{rank}(D \bmod p) < s] + \Pr[\det(M(\beta)) = 0] \\
&\leq \Pr[p|\det(M)] + \frac{\deg(\det M)}{p} \\
&\leq \Pr[p \text{ divides one term in } \det(M)] + \frac{\deg(\det M)}{p} \\
&\leq \frac{\log_{p_{\min}}\|\det(M)\|_\infty}{|\mathbb{P}|} + \frac{sD_{\max}}{p} \\
&\leq \frac{\log_{p_{\min}}|t^s n^{\frac{n}{2}}(HN)^{ns}s^{\frac{s}{2}}|}{|\mathbb{P}|} + \frac{sD_{\max}}{p}
\end{aligned}$$

by Theorem 14. □

Example 17. *For the robot arms system listed in Appendix A, we determined that* $s = 16, D_{\max} = 16, N = 34, t = 305, n = 4$ *and* $H = 4$. *So using* $\mathbb{P} = \{$*the set of* 62 *bit primes*$\}$, $|\mathbb{P}| \approx 5.28 \times 10^{16}$, *and* $p = 2^{62} - 57$, *it follows that* $\Pr[\text{rank}(B) < \text{rank}(D)] < 2.13 \times 10^{-16} < 2^{-52}$.

3. Modified Interpolation using Kronecker Substitution

In order to minimize the number of black box probes needed by our proposed Dixon resultant algorithm to interpolate the rational function coefficients of the monic square-free factors of R, we adapt the sparse multivariate rational function interpolation algorithm of Cuyt and Lee [7], and the Ben-Or/Tiwari algorithm [4] for interpolating sparse polynomials for our purposes.

3.1. The algorithm of Cuyt and Lee

Let $\mathbb{K}$ be a field and let $f/g \in \mathbb{K}(y_1, \cdots, y_m)$ be a rational function such that $\gcd(f, g) = 1$. Suppose the polynomials f and g can be written as

$$f = \sum_{i=0}^{\deg(f)} f_i(y_1, y_2, \ldots, y_m) \text{ and } g = \sum_{j=0}^{\deg(g)} g_j(y_1, y_2, \ldots, y_m)$$

such that f_i and g_j are homogeneous with $\deg(f_i) = i$ and $\deg(g_j) = j$.

Cuyt and Lee's algorithm to interpolate f/g must be combined with a sparse polynomial interpolation algorithm to interpolate f and g. The main advantage of their algorithm is that it exploits the sparsity structure of f and g by interpolating f_i and g_j instead of f and g directly which have more terms.

The first step of their algorithm is to introduce a homogenizing variable z to form an auxiliary rational function

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

and then normalize it using either constant terms $f_0 \neq 0$ or $g_0 \neq 0$. However, if both f_0 and g_0 are zero, one has to pick a basis shift $\beta \in (\mathbb{K} \setminus \{0\})^m$ such that $g(\beta) \neq 0$ and then form a new auxiliary rational function as

$$\frac{f(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}{g(y_1 z + \beta_1, \ldots, y_m z + \beta_m)} := \frac{F(z)}{G(z)} = \frac{\sum_{j=0}^{\deg(f)} \bar{f}_j(y_1, \ldots, y_m)z^j}{\sum_{j=0}^{\deg(g)} \bar{g}_j(y_1, \ldots, y_m)z^j} \in \mathbb{K}(y_1, \ldots, y_m)(z)$$

where $F(0) = \bar{f}_0(y_1, \ldots, y_m) = \tilde{c} \times f(\beta_1, \beta_2, \ldots, \beta_m)$ and $G(0) = \bar{g}_0(y_1, \ldots, y_m) = \tilde{c} \times g(\beta_1, \beta_2, \ldots, \beta_m) \neq 0$ for some $\tilde{c} \in \mathbb{K}$. Notice that the introduction of β forces the production of a constant coefficient in the auxiliary rational function so that it can be normalized using either $\hat{f}_0$ or $\hat{g}_0$. Therefore, we can write

$$\frac{f(y_1 z + \beta_1, \ldots, y_m z + \beta_m)}{g(y_1 z + \beta_1, \ldots, y_m z + \beta_m)} = \frac{\sum_{j=0}^{\deg(f)} \frac{\bar{f}_j(y_1, \ldots, y_m)z^j}{\bar{g}_0}}{1 + \sum_{j=1}^{\deg(g)} \frac{\bar{g}_j(y_1, \ldots, y_m)z^j}{\bar{g}_0}}.$$

The drawback of introducing a basis shift β when needed in the formation of the auxiliary rational function is that it destroys the sparsity of f/g. In particular, $\bar{f}_{\deg(f)}$ and $\bar{g}_{\deg(f)}$ coincide with $f_{\deg(f)}$ and $g_{\deg(f)}$ respectively, but the non-leading lower degree polynomials $\bar{f}_i \neq f_i$ and $\bar{g}_j \neq g_j$, requiring the effect of the basis shift β to be removed before f/g can be recovered. Thus, if a rational function $f/g \in \mathbb{Q}(y_1, \ldots, y_m)$ is represented by a modular black box $\mathbf{B}$, we can recover it by densely interpolating univariate rational functions

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

first using image points obtained from probes to $\mathbf{B}$ for some evaluation point $\alpha \in \mathbb{Z}_p^m$, then we adjust the non-leading coefficients in the numerator and denominator of $\hat{A}(\alpha^j, z)$ by the contributions from the higher degree coefficients before applying sparse polynomial interpolation to recover f/g. Thus, using an appropriate sparse polynomial interpolation algorithm, the adjusted coefficients of the auxiliary rational functions produce the desired rational function f/g that was represented by a black box. We demonstrate how to do this with an example in Subsection 3.5. In order to densely interpolate $\hat{A}(\alpha^j, z)$, we use the Maximal Quotient Rational Function Reconstruction algorithm (MQRFR) [30] which requires $\deg(f) + \deg(g) + 2$ black box probes on z.

3.2. The Ben-Or/Tiwari Algorithm

Let $f = \sum_{k=1}^{t} a_k N_k(x_1, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_n]$ with $a_k \neq 0$ and $t \geq 1$. The maximum number of possible terms in f is $A = \binom{n+\deg(f)}{\deg(f)}$. We say that f is sparse if $t < \sqrt{A}$. A sparse polynomial f can be written as

$$f = \sum_{k=1}^{t} a_k N_k(x_1, \cdots, x_n) = \sum_{k=1}^{t} a_k x_1^{e_{k,1}} x_2^{e_{k,2}} \cdots x_n^{e_{k,n}}.$$

The Ben-Or/Tiwari algorithm [4] interpolates f using $2T$ prime power evaluation points $\{(2^j, 3^j, \cdots, p_n^j) : 0 \leq j \leq 2T-1\}$ where p_n is the n-th prime assuming a term bound $T \geq t$ is known. Let $\hat{m}_i = N_i(2, 3, \cdots, p_n)$ be the monomial evaluations and let $\lambda(z) = \prod_{k=1}^{t}(z - \hat{m}_k) \in \mathbb{Z}[z]$. The Ben-Or/Tiwari algorithm can be easily implemented using the following five main steps:

1. Compute $v \in \mathbb{Z}^{2T}$ where $v_j = f(2^j, 3^j, \cdots, p_n^j)$ for $0 \leq j \leq 2T-1$.
2. Compute t and $\lambda(z)$ from v using the Berlekamp-Massey algorithm [1].
3. Compute the integer roots $\hat{m}_1, \hat{m}_2, \ldots, \hat{m}_t$ of $\lambda(z)$.
4. Obtain the exponents $e_{i,j}$ for N_i for $1 \leq j \leq n$ by factoring $\hat{m}_i$ via repeated trial divisions by the successive primes $2, 3, \ldots, p_n$. For example, $88200 = 2^{\mathbf{3}}3^{\mathbf{2}}5^{\mathbf{2}}7^{\mathbf{2}}$ corresponds to the monomial $x_1^3 x_2^2 x_3^2 x_4^2$.

5. Determine the unknown coefficients a_k by solving the transposed Vandermonde system

$$Va = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \hat{m}_1 & \hat{m}_2 & \cdots & \hat{m}_t \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{t-1} & \hat{m}_2^{t-1} & \cdots & \hat{m}_t^{t-1} \end{pmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v. \qquad (12)$$

The above linear system can be solved in $O(t^2)$ arithmetic operations [37]. We also note that the solution $a \in \mathbb{Z}^t$ is unique since the monomial evaluations are distinct as they are evaluated at powers of primes.

Due to the size of the constant term $\prod_{k=1}^{t} \hat{m}_k$ in $\lambda(z)$ which is a large integer, the Ben-Or/Tiwari algorithm must be performed modulo a prime p satisfying $p > \max_{i=1}^{t} m_i \leq p_n^d$ where $d = \deg(f)$. However, such a prime p may be too large to use machine arithmetic. For example, suppose $n = 8$ and $\deg(f, x_i) = 11$. Then the prime p required by the Ben-Or/Tiwari sparse polynomial algorithm must be larger than $2^{11}3^{11}\cdots 19^{11} = 7.2 \times 10^{77}$. This is the primary disadvantage of using the Ben-Or/Tiwari algorithm. Also, one has to deal with unlucky evaluation points problem posed by using points $(2^j, 3^j, \cdots, p_n^j)$ in modular GCD algorithms [17]. To avoid these problems, we modify the Cuyt and Lee's algorithm and the Ben-Or/Tiwari sparse polynomial interpolation algorithm to use a Kronecker substitution with randomized evaluation points.

We also note that good term bounds $T \geq t$ are not known. In particular, if f is given by a black box then t is not known. For our purposes, we follow the solution of Kaltofen, Lee and Lobo in [24]. We compute $\lambda(z)$ using $j = 2, 4, 6, \ldots$ points for a sufficiently large prime p and we stop when the degree of λ does not change. That is, $\deg(\lambda, z) = 1, 2, 3, \ldots, t-2, t-1, t, t, t, \ldots$ with high probability.

3.3. A new sparse multivariate rational function interpolation method

We develop a new sparse multivariate rational function interpolation algorithm that modifies the Cuyt and Lee's method and the Ben-Or/Tiwari algorithm. Our approach involves the use of a new set of randomized evaluation points and employs a Kronecker substitution to effectively reduce the size of our working primes. Thus, we transform the problem of interpolating a multivariate rational function into a univariate rational function interpolation problem modulo a prime.

3.3.1. 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 18. *Let $\mathbb{K}$ be an integral domain and let $A = f/g \in \mathbb{K}(y_1, \ldots, y_m)$ such that $\gcd(f, g) = 1$. 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

$$F(y, z) = \frac{f(zy, zy^{r_1}, \ldots, zy^{r_1 r_2 \cdots r_{m-1}})}{g(zy, zy^{r_1}, \ldots, zy^{r_1 r_2 \cdots r_{m-1}})} \in \mathbb{K}[y](z). \tag{13}$$

To guarantee the existence of a constant term in the denominator of $F(y, z)$, we use a basis shift $\beta \in (\mathbb{K} \setminus \{0\})^m$ such that $g(\beta) \neq 0$, and instead formally define an auxiliary rational function with a Kronecker substitution as follows.

Definition 19. *Let* $\mathbb{K}$ *be a field and let* $f/g \in \mathbb{K}(y_1, \ldots, y_m)$ *with* $\gcd(f, g) = 1$*. 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 and* $\beta \neq (0, 0, \ldots, 0) \in \mathbb{K}^m$*. 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 $F(y, z, \beta)$ 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 [22] and pick a smooth prime $p > \prod_{j=1}^m r_j$ such that $p - 1 = 2^k \hat{s}$ where $\hat{s}$ is small. An example illustrating how our method works is provided in Subsection 3.5.

Remark 20. *Suppose we want to interpolate a polynomial f such that $m = 6, \deg(f, y_i) = 10$ and $\deg(f) = 60$. If we use the Ben-Or/Tiwari algorithm, we require $p > 13^{60} = 6.8 \times 10^{66}$. Using a Kronecker substitution, we only need $p > 11^6 = 1.7 \times 10^6$.*

To invert a Kronecker substitution K_r, we must know the partial degrees of f and g for all variables. We also need to know the total degrees of f and g in order to 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 pre-compute these degrees with high probability now.

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 represented by a black box. 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

$$A = f/g = \frac{\sum_{k=0}^{d_{f_i}} a_k(y_1, \ldots, y_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m) y_i^k}{\sum_{k=0}^{d_{g_i}} b_k(y_1, \ldots, y_{i-1}, y_{i+1}, y_{i+2}, \ldots, y_m) y_i^k} \tag{14}$$

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 sufficiently large 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 probe the modular black box for $A = f/g$ 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.

Pre-computing the total degrees of f and g in $A = f/g$

Similar to our approach for pre-computing the partial degrees of f and g in $A = f/g$, we describe how to pre-compute $\deg(f)$ and $\deg(g)$. Let p be a sufficiently large prime and let z be a new variable. Let $\alpha, \beta \in (\mathbb{Z}_p \setminus \{0\})^m$ be random evaluation points. Using enough random distinct points for z from $\mathbb{Z}_p$, we discover the total degrees of f and g by probing the modular black box for $A = f/g$ to interpolate the univariate rational function $h(z)$ where

$$h(z) := \frac{\bar{f}(z)}{\bar{g}(z)} = \frac{f(\beta_1 z + \alpha_1, \ldots, \beta_m z + \alpha_m)}{g(\beta_1 z + \alpha_1, \ldots, \beta_m z + \alpha_m)} \in \mathbb{Z}_p(z) \tag{15}$$

such that $\deg(\bar{f}) = \deg(f)$ and $\deg(\bar{g}) = \deg(g)$. The evaluation points $\alpha, \beta \in (\mathbb{Z}_p \setminus \{0\})^m$ are selected at random to ensure that $\deg(\bar{f}) = \deg(f)$ and $\deg(\bar{g}) = \deg(g)$ with high probability.

3.4. 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 now need to interpolate many univariate numerator and denominator polynomials in $\mathbb{Z}_p[y]$.

Let $H = \sum_{j=1}^{t} a_j N_j(y)$ be one of the univariate polynomials to be interpolated in either the numerator or denominator of $K_r(A)$. We avoid unlucky evaluation point (which causes a degree loss of the total degree of the numerator and the denominator of the univariate auxiliary rational functions in z) with high probability by randomizing the evaluation points α^j for $j \geq 0$. 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 N_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 [17]

$$Va = \begin{bmatrix} \hat{m}_1^{\hat{s}} & \hat{m}_2^{\hat{s}} & \cdots & \hat{m}_t^{\hat{s}} \\ \hat{m}_1^{\hat{s}+1} & \hat{m}_2^{\hat{s}+1} & \cdots & \hat{m}_t^{\hat{s}+1} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{\hat{s}+t-1} & \hat{m}_2^{\hat{s}+t-1} & \cdots & \hat{m}_t^{\hat{s}+t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v,$$

where $\hat{m}_i^j = N_i(\alpha^j)$. To compute the coefficients a_i we first solve the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \hat{m}_1 & \hat{m}_2 & \cdots & \hat{m}_t \\ \vdots & \vdots & \vdots & \vdots \\ \hat{m}_1^{t-1} & \hat{m}_2^{t-1} & \cdots & \hat{m}_t^{t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{t-1} \end{bmatrix} = v,$$

using Zippel's $O(t^2)$ algorithm [38] which yields $c = W^{-1}v$. Notice that $V = WD$ 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

$$Va = v \Longrightarrow (WD)a = v \Longrightarrow (Da) = W^{-1}v = c \Longrightarrow a = D^{-1}c.$$

Therefore, for our new method which uses a Kronecker substitution, we use the randomized evaluation point sequence $\{\alpha^{\hat{s}+i} : j \geq 0\}$ where $\alpha \in \mathbb{Z}_p^*$, and a random shift $\hat{s} \in [0, p-2]$ where p is prime, we interpolate the rational functions

$$\frac{f^\beta(\alpha^{\hat{s}+i}, z)}{g^\beta(\alpha^{\hat{s}+i}, z)} = \frac{f(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_{m-1}} + \beta_m)} \quad (16)$$

for $i = 0, 1, 2, \cdots$. Randomizing the point sequence $\{\alpha^{\hat{s}+i} : i \geq 0\}$ ensures $\deg(f^\beta(\alpha^{\hat{s}+i}, z)) = \deg(f)$ and $\deg(g^\beta(\alpha^{\hat{s}+i}, z)) = \deg(g)$ with high probability.

3.5. An illustrative example of our new method

We demonstrate how our new sparse rational function interpolation method works with the following example before the algorithm is presented. Let

$$A = f/g = \frac{y_1^4 + y_2^4 + y_3^4 + y_4^2 + y_5^2 + y_8}{y_6^4 + y_7^4 + y_8^4 + y_6} \in \mathbb{Z}(y_1, y_2, \ldots, y_8)$$

be represented by a black box and suppose we want to interpolate A. Let $f_4 = y_1^4 + y_2^4 + y_3^4, f_2 = y_4^2 + y_5^2, f_1 = y_8$ and $g_4 = y_6^4 + y_7^4 + y_8^4$ and $g_1 = y_6$. So, $f = f_4 + f_2 + f_1$ and $g = g_4 + g_1$. Suppose we have discovered $\deg(f) = \deg(g) = 4$ and the maximum partial degrees of f and g in each variable denoted by d_i for $1 \leq i \leq 8$ using the description presented in Subsection 3.3.1.

We use a Kronecker substitution $K_r : \mathbb{Z}_p(y_1, \ldots, y_m) \to \mathbb{Z}_p(y)$ such that

$$K_r(A) = \frac{f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_7})}{g(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_7})} = \frac{y^{28125} + y^{750} + y^{250} + y^{100} + y^{20} + y^4}{y^{112500} + y^{22500} + y^{4500} + y^{1125}}$$

where our smooth prime $p = 7 \cdot 2^{20} + 1 > \prod_{i=1}^{8} r_i$ and $r = (d_1 + 1, d_2 + 1, \ldots, d_8 + 1) = (4, 4, 4, 2, 2, 4, 4, 4)$. So, $K_r(A)$ is what we want to interpolate. For the sake of brevity, we will only show how to interpolate f.

Now, let $\alpha = 3$, which is a generator for $\mathbb{Z}_p^*$, and let shift $\hat{s} = 2 \in [0, p-2]$ be picked at random. Let $\beta = (2, 0, 5, 11, 19, 14, 73, 0)$. We use β as our basis shift since $g(\beta) = 28436671 \neq 0$. Note that β must be selected at random in $\mathbb{Z}_p^8$ and $\beta \neq 0$.

Step 1: Let $I = \{1, 2, 4\}$ and $J = \{1, 4\}$. Let $N_a = 2 \times \max_{i \in I, j \in J}(\#f_i, \#g_j)$ denote the (minimum) number of auxiliary rational functions needed to interpolate $K_r(A)$. For the purpose of description, suppose we know that $N_a = 6$. We compute the auxiliary rational functions in (16) for $0 \leq i < N_a$. They are

$$\begin{aligned}
\frac{f^\beta(\alpha^{\hat{s}+0}, z)}{g^\beta(\alpha^{\hat{s}+0}, z)} &= \frac{1533140z^4 + 372219z^3 + 2414380z^2 + 5792080z + 3074789}{541036z^4 + 3498541z^3 + 3660193z^2 + 2300570z + 1} \\
\frac{f^\beta(\alpha^{\hat{s}+1}, z)}{g^\beta(\alpha^{\hat{s}+1}, z)} &= \frac{2677008z^4 + 1189072z^3 + 6815537z^2 + 4155022z + 3074789}{3557971z^4 + 1747545z^3 + 398839z^2 + 2439065z + 1} \\
\frac{f^\beta(\alpha^{\hat{s}+2}, z)}{g^\beta(\alpha^{\hat{s}+2}, z)} &= \frac{1087572z^4 + 5756913z^3 + 7222964z^2 + 1730591z + 3074789}{3167873z^4 + 2963937z^3 + 3921591z^2 + 4402146z + 1} \\
\frac{f^\beta(\alpha^{\hat{s}+3}, z)}{g^\beta(\alpha^{\hat{s}+3}, z)} &= \frac{3241826z^4 + 1542919z^3 + 4207334z^2 + 3394522z + 3074789}{6152965z^4 + 6720854z^3 + 4202034z^2 + 6224289z + 1} \\
\frac{f^\beta(\alpha^{\hat{s}+4}, z)}{g^\beta(\alpha^{\hat{s}+4}, z)} &= \frac{1275646z^4 + 5200608z^3 + 3365526z^2 + 2444422z + 3074789}{5437940z^4 + 5888099z^3 + 6974374z^2 + 6413638z + 1} \\
\frac{f^\beta(\alpha^{\hat{s}+5}, z)}{g^\beta(\alpha^{\hat{s}+5}, z)} &= \frac{3290219z^4 + 2061131z^3 + 4627299z^2 + 1433977z + 3074789}{1372709z^4 + 1670491z^3 + 6302257z^2 + 6233953z + 1}.
\end{aligned}$$

In practice, these 6 univariate rational functions must be interpolated by probing the black box for the rational function A. Now since $\deg(f) = 4$, we

attempt to interpolate all possible homogeneous polynomials f_k in f of degrees $k = 4, 3, 2, 1, 0$, in that order using the coefficients of $f^\beta(\alpha^{\hat{s}+i}, z)$ for $0 \leq i \leq 5$.

Step 2: Next, for all i, we check that $\deg(f^\beta(\alpha^{\hat{s}+i}, z)) = \deg(f) = 4$ and $\deg(g^\beta(\alpha^{\hat{s}+i}, z)) = \deg(g) = 4$. In this case, the degrees are equal so we continue. The degree 4 homogeneous polynomial f_4 in f is the first polynomial that must be interpolated. To do this, we collect the leading coefficient sequence

$$v = [1533140, 2677008, 1087572, 3241826, 1275646, 3290219]$$

where $v_i = \mathrm{LC}(f^\beta(\alpha^{\hat{s}+i}, z), z)$ for $0 \leq i \leq 5$. Next, we run the Berlekamp-Massey algorithm (BMA) on $v \in \mathbb{Z}_p^6$ which generates the feedback polynomial

$$\lambda_4(z) = z^3 + 6573867z^2 + 1966358z + 566808 \in \mathbb{Z}_p[z].$$

Step 3: Computing the roots of $\lambda_4(z)$ over $\mathbb{Z}_p$ yields the monomial evaluations $\hat{m} = \{268726, 81, 497359\}$. Using Shanks [34] and the Pohlig-Helman algorithm [35], we solve the discrete logarithms $\{3^{e_1} = 268726, 3^{e_2} = 81, 3^{e_3} = 497359\}$ in $\mathbb{Z}_p^*$ to obtain the exponents $\{e_1 = 20, e_2 = 4, e_3 = 100\}$. Thus, the monomials of $K_r(f_4)$ are $\{y^{20}, y^4, y^{100}\}$.

Step 4: Let $K_r(f_4) = a_1y^{20} + a_2y^4 + a_3y^{100}$. Since $\#K_r(f_4) = 3$, we set $v = [v_1, v_2, v_3]$. We now need to solve for for the coefficients a_i in the following 3×3 shifted transposed Vandermonde system

$$\underbrace{\begin{bmatrix} \hat{m}_1^{\hat{s}+0} = 2418422 & \hat{m}_2^{\hat{s}+0} = 6561 & \hat{m}_3^{\hat{s}+0} = 6862781 \\ \hat{m}_1^{\hat{s}+1} = 6348552 & \hat{m}_2^{\hat{s}+1} = 531441 & \hat{m}_3^{\hat{s}+1} = 3749719 \\ \hat{m}_1^{\hat{s}+2} = 6474694 & \hat{m}_2^{\hat{s}+2} = 6346556 & \hat{m}_3^{\hat{s}+2} = 907481 \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} 1533140 \\ 2677008 \\ 1087572 \end{bmatrix}}_{v}.$$

To solve the above linear system $Va = v$, we follow our explanation in Subsection 3.4 by first solving the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 & 1 \\ \hat{m}_1 = 268726 & \hat{m}_2 = 81 & \hat{m}_3 = 497359 \\ \hat{m}_1^2 = 2418422 & \hat{m}_2^2 = 6561 & \hat{m}_3^2 = 6862781 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = v$$

to get $\{c_1 = 5641816, c_2 = 303581, c_3 = 2927776\}$. Then we compute $a_i = c_i\hat{m}_i^{-\hat{s}} = 5244685$ for $1 \leq i \leq 3$ to get

$$K_r(f_4) = 5244685y^{20} + 5244685y^4 + 5244685y^{1000}.$$

Step 5: Next, we invert the Kronecker map K_r to obtain $y^4 \mapsto y_1^4, y^{20} \mapsto y_2^4$, and $y^{100} \mapsto y_3^4$. So, $f_4 = 5244685y_1^4 + 5244685y_2^4 + 5244685y_3^4$.

Step 6: Since $\deg(f) = 4$, we now attempt to interpolate all the homogeneous polynomials in f of degree less than 4. in f. First, we attempt to interpolate f_3 (if there is such polynomial). To do this, we have to compute

$$v_i = \mathrm{Coeff}(f^\beta(\alpha^{s+i}, z), z^3) - \mathrm{Coeff}(H_i(z), z^3)$$

for $0 \le i \le 5$ where

$$H_i(z) = f_4(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1 r_2 \cdots r_7} + \beta_8) \in \mathbb{Z}_p[z].$$

The above computation of v is the coefficient adjustment that must be done to remove the coefficient contributions of f_4 to f_3 due to the basis shift β. If one uses the coefficients of z^3 in $f^{\beta}(\alpha^{\hat{s}+i}, z)$ then the wrong polynomial will be obtained. To compute the univariate polynomials $H_i(z)$ we interpolate them from values in z. We get

$$\begin{aligned}
H_0(z) &= 1533140z^4 + 372219z^3 + 254554z^2 + 5583z + 107971 \\
H_1(z) &= 2677008z^4 + 1189072z^3 + 5247335z^2 + 2751103z + 107971 \\
H_2(z) &= 1087572z^4 + 5756913z^3 + 6050364z^2 + 2571585z + 107971 \\
H_3(z) &= 3241826z^4 + 1542919z^3 + 7104743z^2 + 1035452z + 107971 \\
H_z(4) &= 1275646z^4 + 5200608z^3 + 5708463z^2 + 1191952z + 107971 \\
H_5(z) &= 3290219z^4 + 2061131z^3 + 7026380z^2 + 5172735z + 107971
\end{aligned}$$

Observe that the coefficients of z^3 in $f^4(\alpha^{\hat{s}+i}, z)$ and $H_i(z)$ which are highlighted in blue are equal hence $v = [0, 0, 0, 0, 0, 0]$ which indicates $f_3 = 0$.

Step 7: Next, we attempt to interpolate f_2 the homogeneous polynomial of total degree 2 in f. Similar to the previous steps, we compute

$$v = [2159826, 1568202, 1172600, 4442624, 4997096, 4940952]$$

where $v_i = \text{Coeff}(f^{\beta}(\alpha^{\hat{s}+i}, z), z^2) - \text{Coeff}(H_i(z), z^2)$ for $0 \le i \le 5$. Then, 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 $\lambda_2(z)$ over $\mathbb{Z}_p$ yields the monomial evaluations $\hat{m} = \{4600185, 3153711\}$.

Step 8: Next, we solve $\{3^{e_1} = 4600185, 3^{e_2} = 3153711\}$ in $\mathbb{Z}_p^*$ to obtain the exponents $\{e_1 = 51840, e_2 = 8640\}$. Thus, the monomials of $K_r(f_2)$ are $\{y^{250}, y^{750}\}$.

Step 9: Let $K_r(f_2) = a_1 y^{250} + a_2 y^{750}$. To solve for a_1 and a_2 we set up the 2×2 shifted transposed Vandermonde system

$$Va = \begin{bmatrix} m_1^{\hat{s}+0} = 5213509 & m_2^{\hat{s}+0} = 1555861 \\ m_1^{\hat{s}+1} = 3153711 & m_2^{\hat{s}+1} = 4630034 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 2159826 \\ 1568202 \end{bmatrix} = v,$$

and first, solve the transposed Vandermonde system

$$Wc = \begin{bmatrix} 1 & 1 \\ \hat{m}_1 = 4600185 & \hat{m}_2 = 3153711 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 2159826 \\ 1568202 \end{bmatrix} = v$$

to obtain $\{c_1 = 1019250191, c_2 = 230831155\}$. Thus $a_i = c_i m_i^{-\hat{s}} = 5244685$ for

$1 \leq i \leq 2$. Inverting the Kronecker map K_r yields $y^{250} \mapsto y_4^2, y^{750} \mapsto y_5^2 \Longrightarrow f_2 = 5244685y_4^2 + 5244685y_5^2$.

Step 10: Before we attempt to interpolate any more homogeneous polynomials degree less than 2 in f, we update the H polynomials because of the coefficient contributions by f_4 and f_2 due to the basis shift. We compute

$$H_i(z) := H_i(z) + f_2(z\alpha^{\hat{s}+i} + \beta_1, z\alpha^{(\hat{s}+i)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+i)r_1r_2\cdots r_7} + \beta_8)$$

for $0 \leq i \leq 5$ and we obtain

$$\begin{aligned}
H_0(z) &= 1533140z^4 + 372219z^3 + 2414380z^2 + 4661291z + 3074789 \\
H_1(z) &= 2677008z^4 + 1189072z^3 + 6815537z^2 + 4830020z + 3074789 \\
H_2(z) &= 1087572z^4 + 5756913z^3 + 7222964z^2 + 4724315z + 3074789 \\
H_3(z) &= 3241826z^4 + 1542919z^3 + 4207334z^2 + 2942806z + 3074789 \\
H_z(4) &= 1275646z^4 + 5200608z^3 + 3365526z^2 + 1555699z + 3074789 \\
H_z(5) &= 3290219z^4 + 2061131z^3 + 4627299z^2 + 6009748z + 3074789
\end{aligned}$$

Step 11: To determine a possible polynomial of degree 1 in f, we compute

$$v = [1130789, 6665035, 4346309, 451716, 888723, 2764262]$$

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

$$\lambda_1(z) = z + 5062589 \in \mathbb{Z}_p[z].$$

Step 12: Computing the roots of $\lambda_1(z)$ yields the monomial evaluation $\hat{m} = \{2277444\}$. Next, we solve the discrete logarithms $\{3^{e_1} = 2277444\}$ to obtain the exponent $\{e_1 = 28125\}$. Thus, the corresponding monomial in y is y^{28125}. Next, we set up and solve the shifted transposed Vandermonde system

$$Va = \begin{bmatrix} \ m_1^{\hat{s}} = 4934082 \ \end{bmatrix} \begin{bmatrix} a_1 \end{bmatrix} = \begin{bmatrix} 1130789 \end{bmatrix} = v.$$

to get $a = [5244685]$. Inverting the Kronecker map K_r yields

$$y^{28125} \mapsto y_8 \Longrightarrow f_1 = 5244685y_8.$$

Step 13: Next we update the H polynomials for $0 \leq i \leq 5$ by computing $H_i(z) = H_i(z) + f_1(z\alpha^{s+i} + \beta_1, z\alpha^{(s+i)r_1} + \beta_2, \ldots, z\alpha^{(s+i)r_1r_2\cdots r_7} + \beta_8)$ in order to interpolate f_0 of f. We get

$$\begin{aligned}
H_0(z) &= 1533140z^4 + 372219z^3 + 2414380z^2 + 5792080z + 3074789 \\
H_1(z) &= 2677008z^4 + 1189072z^3 + 6815537z^2 + 4155022z + 3074789 \\
H_2(z) &= 1087572z^4 + 5756913z^3 + 7222964z^2 + 1730591z + 3074789 \\
H_3(z) &= 3241826z^4 + 1542919z^3 + 4207334z^2 + 3394522z + 3074789
\end{aligned}$$

$$H_z(4) = 1275646z^4 + 5200608z^3 + 3365526z^2 + 2444422z + 3074789$$
$$H_5(z) = 3290219z^4 + 2061131z^3 + 4627299z^2 + 1433977z + 3074789$$

Step 14: Next, we attempt to interpolate f_0. Observe that

$$v_i = \text{Coeff}(f^\beta(\alpha^{\hat{s}+i}, z), z^0) - \text{Coeff}(H_i(z), z^0) = 0 \quad \text{for } 0 \le i \le 5.$$

So, $f_0 = 0$. Hence,

$$\frac{f_4 + f_2 + f_1}{g_4 + g_1} = \frac{5244685\left(y_1^4 + y_2^4 + y_3^4 + y_4^2 + y_5^2 + y_8\right)}{5244685\left(y_6^4 + y_7^4 + y_8^4 + y_6\right)} \in \mathbb{Z}_p(y_1, y_2, \ldots, y_8).$$

Notice that multiplying the numerator and denominator by 5244685^{-1} yields

$$A = f/g = \frac{f_4 + f_2 + f_1}{g_4 + g_1}.$$

For each rational function interpolation in z in Step 1, we use the Maximal Quotient Rational Function Reconstruction algorithm of Monagan [30] which needs $\deg f + \deg g + 2 = 4 + 4 + 2$ points. Therefore, our algorithm needed only $60 = 6 \times (4 + 4 + 2)$ black box probes to reconstruct A!

Finding $\#f_i, \#g_j$ using the Berlekamp Massey Algorithm (BMA)

For convenience, we assumed that the (minimum) number of auxiliary rational functions N_a needed to interpolate $A = f/g$ in step 1 of the above illustrative example is known. This information cannot be determined beforehand because $\#f_i$ and $\#g_i$ are not known. We discuss how to use the BMA to discover N_a.

By design, the homogeneous polynomials $f_{\deg(f)}$ and $g_{\deg(g)}$ must be interpolated first. Thus, we discover $\#f_{\deg(f)}$ and $\#g_{\deg(g)}$ 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 the feedback polynomials $\lambda_1(z)$ and $\lambda_2(z)$. Then we check if $\deg(\lambda_1, z) < \frac{i}{2}$ and $\deg(\lambda_2, z) < \frac{i}{2}$. If these degree conditions are satisfied then $\#f_{\deg(f)} = \deg(\lambda_1, z)$ and $\#g_{\deg(g} = \deg(\lambda_2, z)$ with high probability. If the condition is not satisfied, then more auxiliary rational functions are needed. We note that for $0 \le j < \deg(f)$ and $0 \le k < \deg(g)$, the number of terms in the polynomials f_j or g_k might be greater than $\#f_{\deg(f)}$ or $\#g_{\deg(g)}$. Therefore, we must also check that we have enough auxiliary rational functions to interpolate the lower degree homogeneous polynomials after removing the effect of the basis shift. As before, we feed the adjusted coefficients to the BMA and wait until the degree of the corresponding feedback polynomial $< \frac{i}{2}$. Otherwise, more auxiliary rational function coefficients are needed to complete the interpolation process.

Pre-computing the total degrees of f_i of f and g_i of g in $A = f/g$

Let $A = f/g$ be a sparse rational function in $\mathbb{Q}(y_1, y_2, \ldots, y_m)$ such that $f = \sum_{i=0}^{\deg(f)} f_i$ and $g = \sum_{j=0}^{\deg(g)} g_j$ where f_i and g_j are homogeneous poly-

nomials with $\deg(f_i) = i$ and $\deg(g_j) = j$. To avoid performing unnecessary coefficient adjustment computation in our new sparse rational function interpolation method especially when f and g are very sparse, we must discover $\deg(f_i)$ and $\deg(g_j)$ for all i and j. For example, if $f = f_{100000} + f_0$, then after interpolating f_{100000}, we should not try to interpolate $f_{99999}, f_{99998}, \ldots, f_1$, since f_0 is what we should interpolate next. We describe how to pre-compute $\deg(f_i) = i$ and $\deg(g_j) = j$.

Suppose A is represented by a modular black box $\mathbf{B}$ where p is a sufficiently large prime and suppose we have obtained the total degrees $\deg(f)$ and $\deg(g)$ correctly. Then pick $\alpha \in (\mathbb{Z}_p \backslash \{0\})^m$ at random, and use enough random distinct points for z selected from $\mathbb{Z}_p \setminus \{0\}$ to interpolate the rational function

$$W(z) = \frac{\overline{N}}{\overline{D}} = \frac{\sum_{j=0}^{d_f} \bar{N}_i(z)}{\sum_{i=0}^{d_g} \bar{D}_i(z)} = \frac{f(\alpha_1 z, \ldots, \alpha_m z)}{g(\alpha_1 z, \ldots, \alpha_m z)} \in \mathbb{Z}_p(z),$$

via probes to $\mathbf{B}$, 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.

3.6. Our new sparse multivariate rational function interpolation algorithm

We give a pseudocode which outlines the steps involved to interpolate $A = f/g$ using our new sparse rational function interpolation method. The steps are more detailed in our proposed Dixon resultant algorithm where it is applied.

Algorithm NewRationalFunctionInterpolationMethod

Input: The modular black box $\mathbf{B} : (\mathbb{Z}_p^m, p) \to \mathbb{Z}_p$ for $A = f/g$ over $\mathbb{Q}$ which returns *"division by zero"* if $g(\gamma) = 0$ for some evaluation point $\gamma \in \mathbb{Z}_p^m$.
Remark: Polynomials f and g are viewed as $f = \sum_{i=0}^{\deg(f)} f_i$ and $g = \sum_{j=0}^{\deg(g)} g_j$, where f_i and g_j are homogeneous polynomials, $\deg(f_i) = i$ and $\deg(g_j) = j$. The input prime p for $\mathbf{B}$ will be determined while the algorithm is running.
Output: $A = f/g \bmod p$ with high probability.

1. Probe $\mathbf{B}$ with a sufficiently large prime q to obtain $\deg(f)$ and $\deg(g)$, $d_i = \max(\deg(f, y_i), \deg(g, y_i))$ for $1 \leq i \leq m$, and $\deg(f_i)$ for $0 \leq i \leq \deg(f)$ and $\deg(g_j)$ for $0 \leq j \leq \deg(g)$.
2. Pick a smooth prime $p = 2^k s + 1 > \prod_{i=1}^{m}(d_i + 1)$ to be used by $\mathbf{B}$, a random shift $\hat{s} \in [0, p-2]$, and any generator α for $\mathbb{Z}_p^*$. Let $K_r : \mathbb{Z}_p(y_1, \ldots, y_m) \to \mathbb{Z}_p(y)$ be the Kronecker substitution with $r_i > d_i$ for $1 \leq i \leq m$.
3. 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

4. **For** $i = 0, 1, 2, \ldots$ **do**

Probe the black box $\mathbf{B}$ to interpolate the auxiliary rational functions

$$F(\alpha^{\hat{s}+i}, z, \beta) = \frac{f^\beta(\alpha^{\hat{s}+i}, z)}{g^\beta(\alpha^{\hat{s}+i}, z)} \in \mathbb{Z}_p(z)$$

such that $g^\beta(\alpha^{\hat{s}+i}, z)$ is of the form $1 + \sum_{k=1}^{\deg(g)} a_k z^k$.

5. **if** $i \notin \{2, 4, 6, \ldots\}$ **then** go to 4 **end if**

6. Set $(v, w) := \left([\mathrm{LC}(f^\beta(\alpha^{\hat{s}+j}, z), z) : 0 \le j \le i], [\mathrm{LC}(g^\beta(\alpha^{\hat{s}+j}, z), z) : 0 \le j \le i]\right)$.

7. Apply the Berlekamp Massey algorithm (BMA) on v and w to generate feedback polynomials $\lambda_v(z)$ and $\lambda_w(z)$ over $\mathbb{Z}_p$ respectively.

8. **if** $\deg(\lambda_v) < \frac{i}{2}$ and $\deg(\lambda_w) < \frac{i}{2}$ **then**

Interpolate $f_{\deg(f)}, g_{\deg(g)} \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m]$.

else go to 4 **end if**

9. // Interpolate the lower degree homogenous polynomials $f_{\deg(f)-1}, f_{\deg(f)-2}, \ldots, f_0$

For $k = \deg(f) - 1, \deg(f) - 2, \ldots, 0$ **do**

$v \leftarrow [\mathrm{Coeff}(f^\beta(\alpha^{\hat{s}+j}, z), z^k) : 0 \le j \le i]$.

if $\beta \neq (0, 0, \ldots, 0)$ **then**

Let $H = [0, 0, \ldots, 0] \in \mathbb{Z}^i$.

For $j = 0, 1, 2, \ldots, i$

Interpolate the unique polynomial $W_j \in \mathbb{Z}_p[z]$ where

$$W_j := f_{k+1}(z\alpha^{\hat{s}+j} + \beta_1, z\alpha^{(\hat{s}+j)r_1} + \beta_2, \ldots, z\alpha^{(\hat{s}+j)\prod_{i=1}^{m-1} r_i} + \beta_m).$$

Compute $H_j \leftarrow H_j + W_j \in \mathbb{Z}_p[z]$ // Update H_j.

end for

$v \leftarrow [v_j - \mathrm{Coeff}(H_j, z^k) : 0 \le j \le i]$.

end if

Apply the BMA on v to get the feedback polynomial $\lambda \in \mathbb{Z}_p[z]$.

If $\deg(\lambda) < \frac{i}{2}$ **then** interpolate $f_k \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m]$ **end if**

end for

10. // Interpolate the lower degree homogenous polynomials $g_{\deg(g)-1}, g_{\deg(g)-2}, \ldots, g_0$

Execute Step 9 with all instances of f replaced by g.

end for

11. Construct $f = \sum_{i=0}^{\deg(f)} f_i$ and $g = \sum_{i=0}^{\deg(g)} g_i$.

12. **Output** $A = f/g \bmod p$.

4. The Dixon Resultant Algorithm

For the purpose of description, we assume that there is one monic square-free factor to be interpolated. That is, our algorithms are presented to interpolate only one square-free factor. However, we note that the implementation of our Dixon resultant algorithm handles more than one monic square-free factor. Let

$$S := R_1 = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)} x_1^{d_k} \tag{17}$$

be the one monic square-free factor to be interpolated, and suppose

$$f_k = \sum_{i=0}^{\deg(f_k)} f_{i,k}(y_1, \ldots, y_m) \text{ and } g_k = \sum_{j=0}^{\deg(g_k)} g_{j,k}(y_1, \ldots, y_m) \tag{18}$$

such that $f_{i,k}$ and $g_{j,k}$ are homogeneous with $\deg(f_{i,k}) = i$ and $\deg(g_{j,k}) = j$.

4.1. Algorithm DixonRes

Our Dixon resultant algorithm consists of two main parts, the main algorithm, Algorithm DixonRes (Algorithm 5), and the supplementary algorithm, Algorithm NewPrime (Algorithm 8). Algorithm DixonRes calls Subroutines PolyInterp, RatFun, Remove-Shift, VandermondeSolver and BMStep. It also calls Algorithm NewPrime if additional primes are required. Algorithm 5 to interpolate the monic square-free factor S involves eight major steps, namely:

1. The computation of the degrees $[d_0, \ldots, d_T]$ as defined in (17), the total degrees $\deg(f_k,)\deg(g_k)$, and $\deg(f_{i,k}), \deg(g_{i,k})$ as defined in (18), and 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 for $1 \leq i \leq m$ in Lines 1-5.

2. The use of a Kronecker substitution in Lines 6-7 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 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.

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. 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 then a non-zero basis shift is not needed. 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 **BB** for R. If both degrees are the same then our algorithm does not need a basis shift.

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 19. We remark that H_i in Line 19 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 with high probability.

5. The dense interpolation of auxiliary univariate rational functions A_j in Line 23 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 23 via calls to Subroutine Ratfun. 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 5 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 3.4), an algorithm that solves shifted transposed Vandermonde systems using Zippel's quadratic algorithm.

7. The interpolation of the lower degree homogeneous 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 $f_{j,k}$ and $g_{j,k}$ for $j > i$, whenever $\beta \neq 0$.

8. Performing sparse interpolation using additional primes in Line 38 whenever the rational number reconstruction process fails on the integer coefficients of the first image of S for the first prime. Algorithm 8 (is similar to Algorithm 5 but does not use a Kronecker substitution K_r since the first image of S has been found) uses the support obtained from Algorithm 5 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 1: PolyInterp

Inputs: A prime p and the black box $\mathbf{BB} : \left(\mathbb{Z}_p^{m+1}, p\right) \to \mathbb{Z}_p$ 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 $e_{\max} \geq 2$, the support $d = \{x_1^{d_0}, \ldots, x_1^{d_T}\}$ as defined in (17) 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 $G_j = (\mathbf{BB}\left((\delta_i, Z_j), p\right) : 1 \leq i \leq \hat{D}+1)$. // $\hat{D}+1$ probes to **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 2: BMStep

Inputs: 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 $r \in \mathbb{Z}^{m-1}$ which defines the Kronecker substitution K_r and the list of degrees D_y.

Output: A multivariate polynomial $\bar{F} \in \mathbb{Z}_p[y_1, y_2, \ldots, y_m]$ or **FAIL**.

Run the Berlekamp-Massey algorithm [1] 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 $M = [K_r^{-1}(y^{e_i}) : 1 \leq i \leq t]$// Invert the Kronecker map K_r to get the monomials $M_i(y_1, y_2, \ldots, y_m)$.
if $\deg(M_j, y_i) > D_{y_i}$ for any $1 \leq i \leq m, 1 \leq j \leq t$ **then** return **FAIL end**
$F \leftarrow \text{VandermondeSolver}(\hat{m}, [P_1, \cdots P_t], \hat{s}, M)$// $F \in \mathbb{Z}_p[y_1, \ldots, y_m]$ ... $O(t^2)$
return F

Subroutine 3: VandermondeSolver

Inputs: Vectors $\hat{m}, v \in \mathbb{Z}_p^t$, shift $\hat{s} \in [0, p-2]$, and M a list of t monomials.

Output: A polynomial in $\mathbb{Z}_p[y_1, y_2, \ldots, y_m]$.

Let $W_{ij} = \hat{m}_i^{j-1}$ for $1 \leq i, j \leq t$. // The transposed Vandermonde matrix
Solve $Wc = v$ for c using Zippel's [37] $O(t^2)$ algorithm. See Section 3.4.
Compute $a_i = c_i \hat{m}_i^{-\hat{s}}$ for $1 \leq i \leq t$.
return $\sum_{i=1}^{t} a_i M_i$

Subroutine 4: RemoveShift

Inputs: A non-zero 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 $r \in \mathbb{Z}^m$ which defines the Kronecker substitution K_r and the list of degrees D_y.

Output: A polynomial $f_k \in \mathbb{Z}_p[y_1, \ldots, y_m]$ where f_k is as defined in (18) 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**
        $P_j \leftarrow P_j - \text{coeff}(H_j, z^{\bar{d}})$
        // Adjust P_j 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 polynomial of total degree $\bar{d}$.
    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
  $(f_k, d) \leftarrow (f_k + \overline{A}, d-1)$.
end
return f_k

Algorithm 5: DixonRes

Inputs: The modular black box $\mathbf{BB} : (\mathbb{Z}_p^{m+1}, p) \to \mathbb{Z}_p$ for R with $m \geq 1$ and the list $\mathbb{P}_s$ containing smooth primes.

Output: $S \in \mathbb{Q}(y_1, \ldots, y_m)[x_1]$ of R where S is as defined in (17) or **FAIL**.

Compute $d = \{x_1^{d_0}, \ldots, x_1^{d_T}\}$ as defined in (17) and $\hat{D} = \deg(\det(M), x_1)$.
Compute $\deg(f_k)$ and $\deg(g_k)$ for $0 \leq k \leq T-1$ as defined in (17).
$e_{\max} \leftarrow \max_{k=0}^{T-1} e_k$ where $e_k = \deg(f_k) + \deg(g_k) + 2$ and assume $e_0 \geq e_1, \cdots \geq e_{T-1}$.
Compute $D_y = [\max_{k=0}^{T-1} (\max(\deg(f_k, y_i), \deg(g_k, y_i)))$ for $1 \leq i \leq m]$.
Compute $E_{f_k} = [\deg(f_{i,k}) : 0 \leq i \leq \deg(f_k))]$ and $E_{g_k} = [\deg(g_{i,k}) : 0 \leq i \leq \deg(g_k))]$ for $0 \leq k \leq T-1$ where $f_{i,k}$ and $g_{i,k}$ are as defined in (18).
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 from $\mathbb{P}_s$ such that $p > \prod_{j=1}^{m} r_j$. // p is the input prime for **BB**.
Let $\beta = (0, 0, \ldots, 0) \in \mathbb{Z}_p^m$ be a basis shift.
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 $i \neq j$ 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 $i \notin \{2, 4, 8, 16, 32, \cdots\}$ **then** next **end**
    for $j = 1, 2, \ldots, i$ **do**
        $A_j \leftarrow \text{RatFun}(H_j, \theta, d_k, e_k, p)$ and set $\frac{N_j}{D_j} := A_j \in \mathbb{Z}_p(z)$
        if $\deg(N_j, z) \neq \deg(f_k)$ or $\deg(D_j, z) \neq \deg(g_k)$ **then**
            return **FAIL** // p is unlucky or β is a bad basis shift or $\alpha^{\hat{s}+i-1}$ is unlucky (See Definitions 35 and 36)
        end
    end
    $F_k \leftarrow \text{BMStep}([\text{coeff}(N_j, z^{\deg(f_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r, D_y)$;
    $G_k \leftarrow \text{BMStep}([\text{coeff}(D_j, z^{\deg(g_k)}) : 1 \leq j \leq i], \alpha, \hat{s}, r, D_y)$;
    //$F_k = f_{\deg(f_k),k} \mod p$ and $G_k = g_{\deg(g_k),k} \mod p$)
    if $F_k \neq$ **FAIL** and $G_k \neq$ **FAIL then**
        $f_k \leftarrow \text{RemoveShift}(F_k, \beta, E_{f_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [N_1, \ldots, N_i], r, , D_y)$
        $g_k \leftarrow \text{RemoveShift}(G_k, \beta, E_{g_k}, \hat{s}, \alpha, [\hat{Y}_1, \ldots, \hat{Y}_i], [D_1, \ldots, D_i], r, , D_y)$
        if $f_k \neq$ **FAIL** and $g_k \neq$ **FAIL** then $k \leftarrow k + 1$ **end**
    end
end
$\hat{S} \leftarrow x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)} x_1^{d_k}$ // $\hat{S} = S \bmod p$.
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 6: GetTerms

Inputs: 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 5, 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 (18) or **FAIL**.

$(\overline{A}, \overline{f}_k, d) \leftarrow (F_k, F_k, \deg(F_k))$.
Set $H = (0, 0, , \ldots, 0) \in \mathbb{Z}_q^{N_{\max}}$.
$\overline{D} \leftarrow [\deg(e) : e \in B_1]$, $\hat{M} \leftarrow [\text{supp}(e) : e \in B_1]$ // supp means support.
for $h = 1, 2, \ldots, |\overline{D}|$ **do**
  $\hat{d} \leftarrow \overline{D}_h$
  if $\beta \neq 0$ **then**
    Pick $\theta \in \mathbb{Z}_q^{d+1}$ at random.
    for $j = 1, 2, \cdots, N_{\max}$ **do**
      for $t = 1, 2, \ldots, d+1$ **do**
        $Z_{j,t} \leftarrow \overline{A}(y_1 = \hat{Y}_{j,1}\theta_t + \beta_1, \ldots, y_m = \hat{Y}_{j,m}\theta_t + \beta_m)$; . $O(md\#\overline{A})$
      end
      Interpolate $\overline{W}_j \in \mathbb{Z}_q[z]$ 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 $\hat{d} \neq 0$ **then**
    $P \leftarrow \left[\text{coeff}(N_j, z^{\hat{d}}) : 1 \le j \le N_{\max}\right]$
    if $\beta \neq 0$ **then**
      for $j = 1, 2, \ldots, N_{\max}$ **do**
        $P_j \leftarrow P_j - \text{coeff}(H_j, z^{\hat{d}})$
      end
    end
    $\hat{m} \leftarrow [\hat{M}_i(\alpha) : 1 \le i \le \hat{n}]$ where $\hat{n} = \#\hat{M}_h$; . . . . . $O(m\hat{n}d)$
    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**
    $(\overline{f}_k, d) \leftarrow (\overline{f}_k + \overline{A}, d-1)$.
  end
end
return $\overline{f}_k$.

Algorithm 8: NewPrime

Inputs: The black box $\mathbf{BB} : \left(\mathbb{Z}_q^{m+1}, q\right) \to \mathbb{Z}_q$ for 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 5 and its prime p where S is as defined in (17), the support $d = \{x_1^{d_0}, \ldots, x_1^{d_T}\}$ and $\hat{D} = \deg(R, x_1) > 0$.

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 (18).
$e_k \leftarrow \deg(f_k) + \deg(g_k) + 2$ 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
  Pick a new 62 bit prime q such that $q \nmid P$. // The black box **BB** uses 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 **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 $\hat{Y}_i = (\alpha_1^{\hat{s}+i-1}, \alpha_2^{\hat{s}+i-1} \cdots, \alpha_m^{\hat{s}+i-1})$. // Implements K_r
    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**
    $(\hat{n}, \hat{M}) \leftarrow (\# f_{\deg(f_k),k}, \text{supp}(f_{\deg(f_k),k}))$
    $(\bar{n}, \bar{M}) \leftarrow (\# g_{\deg(g_k),k}, \text{supp}(g_{\deg(g_k),k}))$
    $\hat{m} \leftarrow [\hat{M}_i(\alpha) : 1 \le i \le \hat{n}]$; ..... $O(m(\deg(f_k)\hat{n})$
    $\bar{m} \leftarrow [\bar{M}_i(\alpha) : 1 \le i \le \bar{n}])$; ..... $O(\deg(g_k)\bar{n}))$
    if $\hat{m}_i = \hat{m}_j$ or $\bar{m}_i = \bar{m}_j$ **then** return **FAIL end.**
    for $j = 1, 2, \ldots, N_{\max}$ **do**
      $B_j \leftarrow \text{RatFun}(H_j, \theta, d_k, e_k, q)$ and set $\frac{N_j}{\bar{N}_j} := B_j \in \mathbb{Z}_q(z)$.
      if $\deg(N_j, z) \ne \deg(f_k)$ or $\deg(\hat{N}_j, z) \ne \deg(g_k)$ **then**
        return **FAIL**
      end
    end
    Let $a_i = \text{LC}(N_j, z)$ for $1 \le i \le \hat{n}$ and $b_i = \text{LC}(\hat{N}_j, z)$ for $1 \le i \le \bar{n}$.
    $F_k \leftarrow$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 to $\hat{F}$ mod P to get $\overline{F}$.
  if $\overline{F} \ne$ **FAIL then** return $\overline{F}$ **else** $(\hat{S}, p) \leftarrow (\hat{F}, q)$ **end**
end

Subroutine 7: RatFun

Inputs: A prime p, a list of univariate polynomials $H = [H_j \in \mathbb{Z}_p[x_1] : 1 \leq j \leq e_{\max}]$, an evaluation point $\theta \in \mathbb{Z}_p^{e_{\max}}$, degrees d_k and e_k such that $2 \leq e_k \leq e_{\max}$.

Output: A univariate rational function $A(z) \in \mathbb{Z}_p(z)$.

$\overline{m}(z) \leftarrow \prod_{i=1}^{e_k}(z - \theta_i) \in \mathbb{Z}_p[z]$; ..... $O(e_k^2)$

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 $A(z) = \frac{N(z)}{D(z)} \in \mathbb{Z}_p(z)$. Normalize $A(z)$ s.t. $\text{coeff}(D(z), z^0) = 1$.

return $A(z)$.

4.2. Probabilistic Test

We determine if the output of our Dixon resultant algorithm is correct using Algorithm 9. It uses a probabilistic strategy to determine if the output returned is correct with high probability.

Algorithm 9: CheckResultant

Inputs: The black box $\mathbf{BB} : \left(\mathbb{Z}_q^{m+1}, q\right) \rightarrow \mathbb{Z}_q$ 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 factor $\hat{S} = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{F_k(y_1,\ldots,y_m)}{G_k(y_1,\ldots,y_m)} x_1^{d_k}$ from Algorithm 5.

Output: true (if $\hat{S}$ is correct) or false otherwise.

repeat
  Pick a 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.
  Compute $\delta_i = \mathbf{BB}\left((\beta_i, \alpha), q\right)$ for $i = 1, 2, \ldots, \deg(R, x_1) + 1$.
  Interpolate $F \in \mathbb{Z}_q[x_1]$ using the points $((\beta_i, \delta_i) : 1 \leq i \leq \deg(R, x_1) + 1)$.
  $J \leftarrow \gcd(F, \partial F/\partial x_1)$
  Set $\overline{S} \leftarrow \text{monic}(F/J)$ the monic square-free part of F in $\mathbb{Z}_q[x_1]$.
until $\deg(F) = \deg(R, x_1)$ *and* $\deg(\overline{S}, x_1) = \deg(\hat{S}, x_1)$ and $q \nmid G_k$ and $G_k(\alpha) \neq 0$ for all k.
if $\hat{S}(x_1, \alpha) \neq \overline{S} \in \mathbb{Z}_q[x_1]$ **then**
  return false
else
  return true
end

Note that it is possible that the output of our Dixon resultant algorithm is incorrect, but our probabilistic test fails to detect that it is incorrect. We give the following example to illustrate this.

Example 21. *Let p and q be 62 bit primes such that $p \neq q$ and let $R = (y_1 + 1)x_1 + (pq + y_1 + 2)$. The correct 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 prime p. Then our algorithm incorrectly outputs

$$\hat{S} = x_1 + \frac{(y_1 + 2)}{y_1 + 1}$$

as the correct answer. Let $H = S - \hat{S} = \dfrac{pq}{y_1 + 1}$. Regardless of the random point α selected in Line 3 of Algorithm 9, if the random prime picked in Line 2 of Algorithm 9 is q then $q|H \Longrightarrow \hat{S}(x_1, \alpha) = x_1 + 2 = \overline{S} \in \mathbb{Z}_q[x_1]$ because $p|pq$. Thus, Algorithm 9 will wrongly verify that we have the correct answer.

We will give a failure probability bound for Algorithm 9 in Theorem 51 when we perform the failure probability analysis of our Dixon resultant algorithm.

4.3. 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 in $\mathbb{Z}_p[x_1]$ using the MQRFR algorithm from [30]. Thus, each probe to the black box is computed using C code. Our C code supports primes up to 63 bits in length.

4.3.1. 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],i=1..n}) mod p;
```

does what we want, however, because we use a Kronecker substitution for the parameters $y_1, y_2, \ldots, y_m$ and want our implementation to handle many parameters 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. To speed up evaluations, we have written a C program to compute $M(\alpha)$ for $p < 2^{63}$ using hardware arithmetic. Since Maple has two representations for polynomials with integer coefficients, the sum-of-products representation and the new POLY representation of Monagan and Pearce [31], and Dixon matrices usually involve both representations, we had to allow for both cases. 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 [16] which replaces division with two multiplications and other cheap operations.

Table 1: CPU 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)

Table 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 timings using `Eval`, and column New Total is the new total timings for both systems when our C code for performing matrix evaluations was used.

4.3.2. 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 in our previous work [19, 20]. Since then, we have re-designed our Dixon resultant algorithm to pre-compute these degrees. The timings reported in Table 2 show the improvement when these total degrees were not precomputed (row Before), and the new timings (row After) obtained when $\deg(f_{i,k})$ and $\deg(g_{i,k)}$ were precomputed in our Dixon resultant algorithm.

Table 2: Improvements when $\deg(f_{i,k})$ and $\deg(g_{i,k)}$ were 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

4.3.3. Timings and Optimizations

We present two benchmark tables (Tables 3 and 5) for our Dixon resultant algorithm. Table 3 contains the names of the real parametric polynomial systems (in Column **System**) on which we tested our code, the number of equations in each system (in 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 of maximal rank (in Column $\dim(D)/$**Rank**).

Timings for comparing our new Dixon resultant algorithm with three other methods for computing R are also reported in Table 3. We report the timings for our new Dixon resultant algorithm in Column **DixonRes**, timings for our Maple implementation of the Gentleman & Johnson minor expansion method in Column **Minor**, timings for our implementation of Zippel's sparse interpolation algorithm in Column **Zippel** and timings of our Maple implementation of the Dixon-EDF algorithm in Column **EDF**. Our implementation of the Dixon-EDF algorithm sorts the matrix M by placing the sparsest columns at the left of the

matrix, removes the gcd of each row before starting the elimination and it has a pivot selection algorithm.

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 also use our C code for evaluating a matrix of polynomials at α modulo a prime p. Table 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 $\#R \gg t_{\max}$. Another reason why this is the case is because more primes are needed to recover integer coefficients of R compared to the R_j's. Our algorithm is able to solve many parametric polynomial systems that other methods cannot solve but 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.

The number of black box probes done to obtain all the needed degree bounds, and the number of probes needed to get the first image of the R_j's by our 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 $\#p_i$. The reader can see that one 62 bit prime is typically enough to recover the R_j's. In Table 5, the number of black box probes used by Zippel's algorithm to interpolate R is denoted by Z-probes.

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 3 and 5, we always compute the determinant of the smallest block after confirming that all the blocks produce the same Dixon resultant. Thus, the number of terms in S and R recorded in Tables 3 and 5 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. These include the block sizes of each Dixon matrix and the number of black box probes required by our algorithm to successfully interpolate the R_j's.

Table 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
Heron2d	3	3/3	$(3 \times 3)/3$	7	6	7	**1**	0.043s	0s	0.005s	0.26s
Heron3d	6	6/6	$(16 \times 14)/13$	23	22	23	**1**	0.099s	0.004s	0.036s	0.044s
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
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
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 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**
Hawes1	4	3/2	$(58 \times 54)/46$	78	3	230	**2**	1.14s	$> 10^3$s	0.591s	1.91s
Hawes2	4	4/5	$(9 \times 8)/8$	671	80	671	**1**	0.910s	0.018s	0.294s	0.072s
Hawes4	6	6/3	$(117 \times 154)/60$	26894	1974	37301	**2**	254.58s	$> 10^4$s	399.72s	1533.60
Hermert	14	14/12	$(20 \times 31)/14$	112	8	5976	**2**	0.114s	1.62s	19.25s	0.088s
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
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
Bricard	6	6/11	$(41 \times 44)/29$	312783	38986	1111775	**2**	5491.80s	226.20s	$> 10^4$s	624.60s

! = ran out of memory, NA= Not Attempted

Table 5: 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
Heron3d	$[6, 7]$	160	1392	-	1	757
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
Alie-5	$[967]$	84477	165504	27584	2	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

"x" = $[89, 93, 93, 94, 98, 100, 100, 107, 118, 123, 124, 124, 129, 131, 133, 141]$ and DNF= Did Not Finish

Table 5 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
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" $= [287, 287, 299, 299, 302, 302, 303, 303, 305, 305, 315, 315, 318, 318, 319, 319, 320, 320, 324, 324, 329, 329, 338, 338, 342, 342, 343, 343, 353, 353, 374, 375]$

5. Failure Probability Analysis

5.1. Introduction

To simplify the failure probability analysis in this section, we make the following assumptions. Let $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Z}[y_1, y_2, \ldots, y_m][x_1, x_2, \ldots, x_n]$ be a parametric polynomial system such that $n \geq 1$ and $m \geq 1$. Let M be a $s \times s$ sub-matrix of the rectangular Dixon matrix D obtained from $\mathcal{F}$, where $s = \text{rank}(D)$, and let the Dixon resultant

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

and suppose its 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 $\hat{d} > 0$. Let the monic square-free factor S be as defined in (17). Let $H = \max_{\hat{f} \in \mathcal{F}} \|\hat{f}\|_\infty$, $d_x = \max_{i=1}^{n} (\max_{\hat{f} \in \mathcal{F}} \deg(\hat{f}, x_i))$ and let $D_y = \max_{i=1}^{m} (\max_{\hat{f} \in \mathcal{F}} \deg(\hat{f}, y_k))$. We also note that FAIL in our analysis indicates that our algorithms have encountered an error (or a problem) or that the returned monic square-free factor is incorrect.

5.2. 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 rest of this paper, let $\mathbb{P}_s$ be the list of pre-computed smooth primes to be used in Line 7 of Algorithm 5 where $p_{s_{\min}} = \min(\mathbb{P}_s)$ and let $\mathbb{P}$ be the list of pre-computed primes (not necessarily smooth) to be used in Line 6 of Algorithm 8 and $p_{\min} = \min(\mathbb{P})$ where $|\mathbb{P}| \geq |\mathbb{P}_s|$ 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 $\mathbb{P}$ and $\mathbb{P}_s$ 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 $|\mathbb{P}|, |\mathbb{P}_s| \geq 10^9$. However, it is not efficient to generate these lists of primes as this computation will take a very long time. We briefly discuss how we generate a random prime from $\mathbb{P}$ and $\mathbb{P}_s$ without creating these 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 picking the prime before or after c. Algorithm RandomSmoothPrime which generates a random smooth prime (not uniformly) $[2^{60}, 2^{63}]$. We estimate there are about 10^{10} smooth primes in this range.

Now we characterize the primes that must be avoided in our algorithm.

Algorithm RandomSmoothPrime
repeat
Pick $q_i \in [750, 2500]$ for $1 \leq i \leq 6$ at random.
Set $p = 1 + \prod_{i=1}^{6} q_i$.
until $p \in (2^{60}, 2^{63})$ and p is prime.

Definition 22. *We say a prime p is **bad** if p divides $\mathrm{LC}(R, x_1)$. If prime p is not bad then we say p **causes missing terms** if p divides any integer coefficient of R_j.*

Example 23. *Suppose the Dixon resultant $R = 15y_2x_1^2 + (7y_1 - 49)x_1 + 7$. So $\mathrm{LC}(R, x_1) = 15y_2$ and its only monic square-free factor is*

$$S = x_1^2 + \frac{(7y_1 - 49)}{15y_2}x_1 + \frac{7}{15y_2}.$$

Clearly, the primes $3, 5$ are bad, and the primes $3, 5, 7$ cause missing terms.

Example 24. *Suppose the Dixon resultant $R = (3137y_2+3)x_1^2+(7y_1+1)x_1+7$ and let*

$$S = x_1^2 + \frac{(7y_1 + 1)}{3137y_2 + 3}x_1 + \frac{7}{3137y_2 + 3}.$$

Notice that $\phi_{3137}(\mathrm{LC}(R, x_1)) = 3 \neq 0$, 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 S for the first prime (See Lines 37-38 of Algorithm 5). If the rational number reconstruction process does not succeed with the first prime, then more primes are used. Our algorithm is designed this way because we do not know the number of primes needed a priori since R is represented by the black box **BB**, and we want to use the fewest number of primes possible. Therefore, in Example 24, if $\phi_{3137}(S)$ is the first image obtained, the rational number reconstruction process will succeed with the input prime $p = 3137$, and Algorithm 5 will return the incorrect answer

$$\hat{S} = x_1^2 + \left(\frac{7}{3}y_1 + \frac{1}{3}\right)x_1 + \frac{7}{3}.$$

Our probabilistically test (Algorithm 9) will catch this error with high probability. We now bound the failure probability of a prime p is bad or p causes missing terms.

Proposition 25. *If p is chosen at random from the list of pre-computed primes*

$\mathbb{P}$ *and* $p_{\min} = \min(\mathbb{P})$ *then*

$$\Pr[\,p \textit{ is bad or causes missing terms}\,] \leq \frac{\log_{p_{\min}} \|R\|_\infty + \#R_j \log_{p_{\min}} \|R_j\|_\infty}{|\mathbb{P}|}.$$

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 $\mathbb{P}$ is at most $\lfloor \log_{p_{\min}} c \rfloor$. So

$$\Pr[\,p \text{ divides } c\,] \leq \frac{\log_{p_{\min}} c}{|\mathbb{P}|}.$$

Clearly, p is bad $\iff p | \mathrm{LC}(R, x_1) \implies p$ divides one term in $\mathrm{LC}(R, x_1)$. Thus,

$$\Pr[p \text{ is bad}] \leq \Pr[p \text{ divides one term in } \mathrm{LC}(R, x_1)] \leq \frac{\log_{p_{\min}} \|R\|_\infty}{|\mathbb{P}|}. \tag{19}$$

Furthermore, the probability that p causes missing terms is at most

$$\frac{\#R_j \log_{p_{\min}} (\|R_j\|_\infty)}{|\mathbb{P}|}. \tag{20}$$

Adding (19) and (20) completes our proof. □

5.3. Evaluation Points

After selecting a random smooth prime from $\mathbb{P}_s$, the next major 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 1). 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 26. *Let p be a prime that is not bad. Let $\alpha \in \mathbb{Z}_p^m$ be an evaluation point. We say that $\alpha \in \mathbb{Z}_p^m$ is **bad** if $\mathrm{LC}(R, x_1)(\alpha) = 0$. 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 27. *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 $p \nmid a \Rightarrow p \nmid \mathrm{LC}(R, x_1) = (y_1 - a)$. 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 28. *Suppose prime p is not bad. If $\alpha \in \mathbb{Z}_p^m$ is chosen at random then*

$$\Pr[\,\alpha \textit{ is bad or causes missing terms}\,] \leq \frac{nmsD_y + n^2 s^2 m d_x D_y}{p}. \tag{21}$$

Proof. Using Lemma 2 and Theorem 14(ii), we have that

$$\Pr[\,\alpha \text{ is bad}\,] = \Pr[\mathrm{LC}(R, x_1)(\alpha) = 0] \leq \frac{\deg(\mathrm{LC}(R, x_1))}{p} \leq \frac{nmsD_y}{p}. \quad (22)$$

Now we address the case when the evaluation point α causes missing terms. Let N be the number of monic square-free factors R_j to be interpolated and let

$$\Delta(y_1, \ldots, y_m) = \prod_{j=1}^{N} \prod_{k=0}^{T_j - 1} f_{jk}.$$

Since T_j is the number of the rational function coefficients in each R_j, we have

$$\left(\sum_{j=1}^{N} T_j\right) \leq \left(\sum_{j=1}^{N} d_{T_j}\right) \leq \sum_{j=1}^{N} \deg(R_j, x_1) \leq \deg(R, x_1) \leq nsd_x.$$

Clearly, $\deg(\Delta) = \sum_{j=1}^{N} \sum_{k=0}^{T_j - 1} \deg(f_{jk})$. Thus, using Theorem 14, we get

$$\deg(\Delta) \leq \sum_{j=1}^{N} \left(T_j \sum_{i=1}^{m} nsD_y\right) \leq n^2 s^2 m d_x D_y.$$

Therefore, by Lemma 2,

$$\Pr[\,\alpha \text{ causes missing terms}] = \Pr[\Delta(\alpha) = 0] \leq \frac{\deg(\Delta)}{p} \leq \frac{n^2 s^2 m d_x D_y}{p}. \quad (23)$$

Adding (22) and (23) completes our proof. □

5.4. Monic Univariate Polynomial Images of R

Recall that Subroutine 1 (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 1 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 1 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 degrees $[d_0, \ldots, d_T]$ as defined in (17) are the same as the degrees of the support of the

interpolated monic polynomial images H in Subroutine 1, then we know we have a correct univariate monic polynomial image of R with high probability. We detect this in Line 7 of Subroutine 1. Otherwise, we interpolate a monic square-free factor that have missing terms. We now find the probability that Subroutine 1 returns FAIL.

Lemma 29. *Assume the degrees $[d_0, \ldots, d_T]$ from $\{x^{d_0}, \ldots, x^{d_T}\}$ as defined in (17) 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 $\mathbb{P}$ and $p_{\min} = \min(\mathbb{P})$ then the probability that Subroutine 1 returns FAIL is at most*

$$e_{\max} \left(\frac{nmsD_y + n^2 s^2 m d_x D_y}{p} + \frac{nsd_x \log_{p_{\min}} (\|S\|_\infty \|R\|_\infty)}{|\mathbb{P}|} \right).$$

Proof. There are two sources of failure in Subroutine 1. 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 $\deg(B_j, x_1) < \deg(R, x_1)$ in Line 5 of Subroutine 1. Thus,

$$\begin{aligned}
&\Pr[\text{prime } p \text{ or evaluation point } Z_j \text{ is bad in Line 5 of Subroutine 1}] \\
&\leq \Pr[p \text{ divides } \mathrm{LC}(R, x_1)] + \Pr[\mathrm{LC}(R, x_1)(Z_j) = 0] \\
&\leq \Pr[\, p \text{ divides one term in } \mathrm{LC}(R, x_1)] + \Pr[\mathrm{LC}(R, x_1)(Z_j) = 0] \\
&\leq \frac{\log_{p_{\min}} \|\mathrm{LC}(R, x_1)\|_\infty}{|\mathbb{P}|} + \frac{\deg(\mathrm{LC}(R, x_1))}{p} \leq \frac{\log_{p_{\min}} \|R\|_\infty}{N} + \underbrace{\frac{nmsD_y}{p}}_{\text{by (22)}}. \quad (24)
\end{aligned}$$

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 $\mathrm{supp}(H_j) \neq \{x^{d_0}, \ldots, x^{d_T}\}$ in Line 7 of Subroutine 1 where H_j is the monic square-free part of the univariate polynomial B_j. Since T is the number of polynomials f_k in S, we have that $T \leq \deg(R, x_1) \leq nsd_x$. Thus

$$\begin{aligned}
&\Pr[\, Z_j \text{ or } p \text{ causes missing terms in Line 7 of Subroutine 1}] \\
&\leq \Pr[\text{Any } f_k(Z_j) = 0] + \Pr[\, p \text{ divides any } f_k \text{ in } S] \\
&\leq Pr[\text{Any } f_k(Z_j) = 0] + \Pr[p \text{ divides one term of } f_k \text{ in } S] \\
&\leq \Pr[\prod_{k=0}^{T-1} f_k(Z_j) = 0] + \frac{T \log_{p_{\min}} \|S\|_\infty}{|\mathbb{P}|} \\
&\leq \underbrace{\frac{n^2 s^2 m d_x D_y}{p}}_{\text{by (23)}} + \frac{nsd_x \log_{p_{\min}} \|S\|_\infty}{|\mathbb{P}|}. \quad (25)
\end{aligned}$$

Hence $\Pr[\text{ Subroutine 1 returns FAIL}] \leq e_{\max}((24) + (25))$. □

Remark 30. *If Algorithm 5 calls Subroutine 1 then the list of primes $\mathbb{P}$ is replaced with the list of smooth primes $\mathbb{P}_s$, so the above theorem works accordingly.*

5.5. Unlucky Content

Definition 31. *Let the Dixon resultant* $R = \sum_{i=1}^{t_r} \bar{r}_i(y_1, \ldots, y_m)x_1^{e_i}$ *with* $t_r \geq 2$. *Let* p *be a prime such that* $\phi_p(\bar{r}_i) \neq 0$ *for all* i. *Let the polynomial content of* R *be denoted by* $C = \gcd(\bar{r}_1, \bar{r}_2 \ldots, \bar{r}_{t_r})$. *We say* p ***causes unlucky content*** *if*

$$\gcd\left(\phi_p\left(\frac{\bar{r}_1}{C}\right), \phi_p\left(\frac{\bar{r}_2}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{t_r}}{C}\right)\right) \neq 1.$$

Example 32. *Let* $p \neq 2$ *be a sufficiently large prime and let*

$$R = (2y_1^2 + 2y_1 + 2py_1)x_1 + (2y_1^2 + 2y_1)$$

where $\bar{r}_2 = (2y_1^2 + 2y_1 + 2py_1)$ *and* $\bar{r}_1 = (2y_1^2 + 2y_1)$. *Notice that* $C = 2y_1$, *but*

$$\gcd\left(\phi_p\left(\frac{\bar{r}_1}{C}\right), \phi_p\left(\frac{\bar{r}_2}{C}\right)\right) = y_1 + 1 \neq 1.$$

Thus, our algorithm will incorrectly output $\hat{S} = x_1 + 1$ *instead of*

$$S = x_1 + \frac{y_1 + 1}{y_1 + p + 1},$$

which is the correct answer because p *caused an unlucky content.*

We cannot detect in advance the occurrence of an unlucky content in our algorithm due to the black box representation of the Dixon resultant R. But it can be detected after our algorithm terminates using our probabilistic test (Algorithm 9) which was presented in Subsection 4.2.

Theorem 33. *Suppose the Dixon resultant* $R = \sum_{i=1}^{t_r} \bar{r}_i(y_1, \ldots, y_m)x_1^{e_i}$ *where* $t_r \geq 2, N_t = \max_{i=1}^{t_r}(\#\bar{r}_i)$. *Let* p *be a prime chosen at random from the list of pre-computed primes* $\mathbb{P}$ *and* $p_{\min} = \min(\mathbb{P})$. *Suppose* $\bar{r}_i \in \mathbb{Z}[y_2, \ldots, y_m][y_1]$. *If* $\phi_p(\mathrm{LC}(\bar{r}_i, y_1)) \neq 0$ *for all* i *then the probability that* p *causes unlucky content*

$$< \frac{2nsD_y \log_{p_{\min}}((2nsN_tD_y)\|R\|_\infty)}{|\mathbb{P}|}.$$

Proof. Observe that

$$\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_1}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{t_r}}{C}\right)\right)\right) > 0 \Longrightarrow \deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_i}{C}\right), \phi_p\left(\frac{\bar{r}_j}{C}\right)\right)\right) > 0$$

for any $1 \leq i \neq j \leq t_r$, 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

$$\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_1}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{t_r}}{C}\right)\right)\right) > 0 \Longrightarrow \deg(\gcd(\phi_p(\bar{r}_i), \phi_p(\bar{r}_j)), y_1) > 0$$

$\implies \phi_p(\overline{R}_s) = 0$ where $\overline{R}_s$ is the Sylvester resultant of $\bar{r}_i, \bar{r}_j$ in y_1. Therefore,

$$\Pr[\deg\left(\gcd\left(\phi_p\left(\frac{\bar{r}_0}{C}\right), \ldots, \phi_p\left(\frac{\bar{r}_{\hat{d}}}{C}\right)\right)\right) > 0] \leq \Pr[\phi_p(\overline{R}_s) = 0] \leq \frac{\log_{p_{\min}} \|\overline{R}_s\|_\infty}{|\mathbb{P}|}.$$

To complete our proof, we obtain a bound for $\|\overline{R}_s\|_\infty$ as follows. Let $\hat{S}$ be the Sylvester matrix whose entries are coefficients of $\bar{r}_i$ and $\bar{r}_j$ in y_1. Thus, $\overline{R}_s = \det(\hat{S})$. Using Theorem 14, the dimension of $\hat{S}$ denoted by $\dim(\hat{S}) \leq \deg(\bar{r}_i, y_1) + \deg(\bar{r}_j, y_1) \leq 2\deg(R, y_1) \leq 2nsD_y$. Thus,

$$\|\overline{R}_s\|_\infty < (\dim(\hat{S})^{\frac{1}{2}} \max_{j,k}(\#\hat{S}_{jk}) \|\hat{S}_{jk}\|_\infty)^{2nsD_y} < (2nsD_yN_t\|R\|_\infty)^{2nsD_y}$$

by Theorem 4, and we are done. □

Example 34. *For the robot arms system from [26] listed in Appendix A we have* $n = 4$, $s = 16$, $D_y = 2$ *so that* $2nsD_y = 256$, $N_t = 145185$ *and* $\|R\|_\infty = 93296724912960039813120 = 9.33 \times 10^{22}$. *If* $\mathbb{P} = \{$*the set 62 bit primes*$\}$, *since* $|\mathbb{P}| \approx 5.28 \times 10^{16}$ *it follows from Theorem 33 that the probability that* p *causes unlucky content is* $< 8.06 \times 10^{-15} < 2^{-46}$.

5.6. 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 S where S is as defined in (17) is reduced to a univariate rational function interpolation modulo a prime 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

$$K_r(f_k/g_k) = \frac{f_k(y, y^{r_1}, y^{r_1r_2}, \ldots, y^{r_1r_2\cdots r_{m-1}})}{g_k(y, y^{r_1}, y^{r_1r_2}, \ldots, y^{r_1r_2\cdots r_{m-1}})} \in \mathbb{Z}_p(y)$$

where $r_i > \max\left(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i))\right)$, prime $p > \prod_{i=1}^m r_i$ and polynomials f_k and g_k are as defined in (17). Let α be a generator for $\mathbb{Z}_p^*$ and let $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ be a basis shift as described in Lines 8-13 of Algorithm 5. Let

$$F_k(y, z, \beta) := \frac{f_k^\beta(y, z)}{g_k^\beta(y, z)} = \frac{f_k(zy + \beta_1, zy^{r_1} + \beta_2, \ldots, zy^{(r_1r_2\cdots r_{m-1})} + \beta_m)}{g_k(zy + \beta_1, zy^{r_1} + \beta_2, \ldots, zy^{(r_1r_2\cdots r_{m-1})} + \beta_m)} \in \mathbb{Z}_p(y)(z).$$

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 7 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 5.

Definition 35. *A prime* p *is said to be* ***unlucky*** *if* $p|\mathrm{LC}(f_k^\beta(y, z), z)$ *or* $p|\mathrm{LC}(g_k^\beta(y, z), z)$ *for any* k.

Definition 36. *Suppose p is not an unlucky prime. We say that $\alpha \in \mathbb{Z}_p \setminus \{0\}$ is an **unlucky evaluation point** if $\deg(f_k^\beta(\alpha, z), z) < \deg(f_k, z)$ or $\deg(g_k^\beta(\alpha, z), z) < \deg(g_k, z)$ for any k. That is, $\mathrm{LC}(f_k^\beta, z)(\alpha) = 0$ or $\mathrm{LC}(g_k^\beta, z)(\alpha) = 0$. We say that $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ is a **bad basis shift** if $\gcd(f_k, g_k) = 1$ and α is not an unlucky evaluation point but $\deg(\gcd(f_k^\beta(\alpha, z), g_k^\beta(\alpha, z))) > 0$ for any k.*

Example 37. *Let $f_1/g_1 = (y_1 + y_2 + y_3)/(qy_2^2 + y_1 + y_3)$. Let prime $p > 10$ and let $r = (2, 3)$ where $r_i > \max\{\deg(f_1, y_i), \deg(g_1, y_i)\}$, so*

$$K_r(f_1/g_1) = \frac{f_1(y, y^2, y^6)}{g_1(y, y^2, y^6)} = \frac{y^6 + y^2 + y}{y^6 + qy^4 + y} \in \mathbb{Z}_p(y).$$

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 $\beta = (5, 2, 3)$. Then

$$F_1(y, z, \beta) = \frac{f_1^\beta(y, z)}{g_1^\beta(y, z)} = \frac{10 + \left(y^6 + y^2 + y\right) z}{qy^4z^2 + (y^6 + y)\, z + 8 + 4q} \in \mathbb{Z}_p[y](z).$$

If $p|q$ then $p|\mathrm{LC}(g_1^\beta(y, z), z) = qy^4$. Thus, for $\alpha \in \mathbb{Z}_p^m$, we have that p is an unlucky prime since $\deg(g_1^\beta(\alpha, z)) < \deg(g_1) = 2$.

Example 38. *Let $f_1/g_1 = (2891y_1+y_2+y_3)/(y_2^2+y_1+y_3)$. Notice $\gcd(f, g) = 1$. Let $p = 3137$ and let $\beta = (5, 2, 3) \in \mathbb{Z}_{3137}$ serve as the basis shift for f_1/g_1. Let $r = (2, 3)$ $r_i > \max\{\deg(f_1, y_i), \deg(g_1, y_i)\}$. Then*

$$K_r(f_1/g_1) = \frac{f_1(y, y^2, y^6)}{g_1(y, y^2, y^6)} = \frac{y^6 + y^2 + 2891y}{y^6 + y^4 + y} \in \mathbb{Z}_{3137}(y).$$

Thus,

$$F_1(y, z, \beta) = \frac{f_1^\beta(y, z)}{g_1^\beta(y, z)} = \frac{1912 + (y^6 + y^2 + 2891y)z}{12 + y^4z^2 + (y^6 + 4y^2 + y)z} \in \mathbb{Z}_{3137}[y](z),$$

$\mathrm{LC}(f_k^\beta(y, z), z) = y^6 + y^2 + 2891y$ and $\mathrm{LC}(g_k^\beta(y, z), z) = y^6 + 4y^2 + y$. Clearly, $p = 3137$ is not an unlucky prime. If $\alpha = 3$ is chosen from $\mathbb{Z}_{3137}^*$, then

$$F_1(3, z, \beta) = \frac{f_1^\beta(3, z)}{g_1^\beta(3, z)} = \frac{1912}{81z^2 + 768z + 12} \in \mathbb{Z}_{3137}(z).$$

Therefore, $\deg(f_1(\alpha, z)) < 1$ because $\mathrm{LC}(f_1^\beta(y, z), z)(3) = 9411 = 3 \times 3137 \equiv 0 \pmod p$ which implies that $\alpha = 3$ is an unlucky evaluation point.

To avoid the occurrence of unlucky evaluation points with high probability in Algorithm 5, 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 23 of Algorithm 5. Line 25 detects unlucky evaluation points, a bad basis shift and a prime p that is unlucky.

Example 39. *Let p be a prime and let $f_1/g_1 = y_1/(y_1y_2+y_3y_2) \in \mathbb{Z}_p(y_1,y_2,y_3)$. Let $r=(2,2)$ where $r_i > \max\{\deg(f_1,y_i),\deg(g_1,y_i)\} = 1$ for $1 \le i \le 3$. Then*

$$K_r(f_1/g_1) = \frac{f_1(y,y^2,y^4)}{g_1(y,y^2,y^4)} = \frac{y}{(y+y^4)y^2} = \frac{y}{y^3+y^6}.$$

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

$$\mathcal{R} = \operatorname{res}(f_1^\beta(\alpha,z), g_1^\beta(\alpha,z), z) = \alpha^2(\alpha^3\beta_1-\beta_3)(\alpha\beta_1-\beta_2) \neq 0$$

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 40. *Let N_a be the number of auxiliary rational functions needed by Algorithm 5 to interpolate the monic square-free factor S of R. If a smooth prime p is chosen at random from the list of pre-computed primes $\mathbb{P}_s$ and $p_{s_{\min}} = \min(P_s)$ then the probability that Algorithm 5 returns FAIL in Line 25 is at most*

$$\frac{2nsd_x(1+nsD_y)^m}{p-1} + \frac{N_an^3s^3m^2D_y^2d_x}{p-1} + \frac{2N_ansd_x\log_{p_{s_{\min}}}\|S\|_\infty}{|\mathbb{P}_s|}.$$

Proof. There are 3 causes of FAIL in Line 25 of Algorithm 5. They are unlucky evaluation points, a bad basis shift and an unlucky prime p. We remark that all 3 failure causes are direct consequence of attempting to interpolate auxiliary rational functions A_j in Line 23 of Algorithm 5 when it calls Subroutine Ratfun.

1. Unlucky evaluation point case:. Let

$$\Delta(y) = \prod_{k=0}^{T-1} \mathrm{LC}(f_k^\beta(y,z),z)\mathrm{LC}(g_k^\beta(y,z),z) \in \mathbb{Z}_p[y].$$

For $0 \le j \le N_a-1$, the evaluation point $\alpha^{\hat{s}+j-1}$ in Line 17 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, $\alpha^{\hat{s}+j-1}$ is an unlucky evaluation point $\iff$ $\Delta(\alpha^{\hat{s}+j-1}) = 0$. Since T is the number of numerator polynomials f_k in S, we have that $T \le \deg(R,x_1) \le nsd_x$. Also, recall that the r_i's from the Kronecker map K_r applied on f_k and g_k satisfy

$$r_i = 1 + \max_{k=0}^{T-1}(\deg(f_k,y_i),\deg(g_k,y_i)) \le 1+\deg(R,y_i) \le 1+nsD_y.$$

Thus $\deg(\Delta(y)) = 2T\prod_{i=1}^{m} r_i \leq 2nsd_x(1+nsD_y)^m$. Therefore, the probability that $\alpha^{\hat{s}+j-1}$ is unlucky for $0 \leq j \leq N_a - 1$ is at most

$$\frac{N_a \deg(\Delta)}{p-1} \leq \frac{2N_a nsd_x(1+nsD_y)^m}{p-1}. \tag{26}$$

2. Bad basis shift case:. Suppose $\theta_j := \alpha^{\hat{s}+j-1}$ is not unlucky for $1 \leq j \leq N_a$. Let $w_1, w_2, \cdots w_m$ be new variables and let $G_{kj} \in \mathbb{Z}_p(w_1, w_2, \ldots, w_m)(z)$ and

$$G_{kj} = \frac{\hat{f}_{k_j}(w_1, \ldots, w_m)}{\hat{g}_{k_j}(w_1, \ldots, w_m)} = \frac{f_k(\theta_j z + w_1, \ldots, z\theta_j^{(r_1 r_2 \cdots r_{m-1})} + w_m)}{g_k(\theta_j z + w_1, \ldots, z\theta_j^{(r_1 r_2 \cdots r_{m-1})} + w_m)}.$$

Recall that a basis shift $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ does not affect the leading coefficients of auxiliary functions, so $\mathrm{LC}(\hat{f}_{k_j}, z)(\beta) \neq 0$ and $\mathrm{LC}(\hat{g}_{k_j}, z)(\beta) \neq 0$. 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 $\iff \deg(\gcd(\hat{f}_{k_j}(z,\beta), \hat{g}_{k_j}(z,\beta)) > 0$ for any k and $j \iff \Delta(\beta) = 0$. Using the Bezout bound, we have

$$\deg(\overline{R}_{kj}) \leq \deg(\hat{f}_{k_j})\deg(\hat{g}_{k_j}) \leq \deg(f_k)\deg(g_k) \leq \left(\sum_{k=1}^{m} \deg(R, y_k)\right)^2 \leq n^2m^2s^2D_y^2.$$

Hence,

$$\deg(\Delta) \leq \sum_{k=0}^{T-1}\sum_{j=1}^{N_a} \deg(\overline{R}_{kj}) \leq N_a T \left(\sum_{k=1}^{m} \deg(R, y_k)\right)^2 \leq n^3s^3m^2D_y^2d_xN_a$$

since $T \leq nsd_x$ and $\sum_{k=1}^{m} \deg(R, y_k) \leq nmsD_y$ by Theorem 14. Therefore,

$$\Pr\left[\beta \text{ is bad basis shift}\right] = \Pr[\Delta(\beta) = 0] \leq \frac{n^3s^3m^2D_y^2d_xN_a}{p-1}. \tag{27}$$

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 the rational function A_j in Line 23 of Algorithm 5 in z to drop lower than $\deg(f_k)$ or $\deg(g_k)$. Since $\deg(f_{\deg(f),k}) = \deg(f_k)$ and $\deg(g_{\deg(g),k}) = \deg(g_k)$, we have

$$\begin{aligned}
&\Pr[p \text{ is unlucky in } A_j] \leq \Pr[\,p \,\text{divides } f_{k,\deg(f)} \text{ or } g_{\deg(g),k} \text{ in } S \text{ for } 0 \leq k \leq T-1] \\
&\leq \Pr[p \text{ divides one integer coefficient of } f_{k,\deg(f)} \text{ or } g_{k,\deg(g)} \text{ in } S \text{ for } 0 \leq k \leq T-1] \\
&\leq \frac{2T\log_{p_{s_{\min}}}(\|S\|_\infty)}{N_s} \leq \frac{2nsd_x\log_{p_{s_{\min}}}\|S\|_\infty}{N_s}
\end{aligned}$$

since the number of polynomials f_k and g_k in S is $T \leq \deg(R, x_1) \leq nsd_x$. Thus,

$$\Pr[p \text{ is unlucky in } A_j \text{ for } 1 \leq j \leq N_a] \leq \frac{2N_a nsd_x \log_{p_{s_{\min}}} \|S\|_\infty}{|\mathbb{P}_s|}. \tag{28}$$

Adding (26), (27) and (28) completes our proof. □

Remark 41. *The exponential factor* $(1+nsD_y)^m$ *in* (26) *is caused by the Kronecker substitution* K_r*. For the robot arms system from [26] listed in Appendix A we have* $n = 4$*,* $m = 7$ $s = 16$*,* $D_y = 2$*, so* $(1 + nsD_y)^m = 594,467,302,491,009$ *which is large. This factor allows* $\Delta(y)$ *to have* $\deg(\Delta(y))$ *roots in* $\mathbb{Z}_p$*. In practice, we do not encounter unlucky evaluation points because the average number of roots of a random* $\Delta(y)$ *over* $\mathbb{Z}_p$ *is* 1 *[36, Chapter 4]. Also, since the Dixon resultant* R *has three square-free factors* $R_1 = A_1, R_2 = A_2$ *and* $R_3 = A_3A_4$ *(see Example 1), and the maximum partial degrees of* R_1, R_2 *and* R_3 *in the parameters* $y_1, y_2, \ldots, y_7$ *are* $4, 2, 4, 2, 2, 2, 2$ *and* 2*, we use* $r = (5, 3, 5, 3, 3, 3, 3)$ *for the Kronecker substitution and the exponential factor becomes* $3^5 \times 5^2 = 6075$*.*

5.7. Discovering the size and supports of the polynomials $K_r(f_{i,k})$ *and* $K_r(g_{i,k})$

We now aim to obtain the probability of failure of finding the correct support for the univariate polynomials $K_r(f_{i,k}) \in \mathbb{Z}_p[y]$ and $K_r(g_{i,k}) \in \mathbb{Z}_p[y]$ where K_r is the Kronecker substitution and $f_{i,k}$ and $g_{i,k}$ are as defined in (18). These supports along with their sizes (number of terms) are used to determine the multivariate polynomials $f_{i,k}$ and $g_{i,k}$ in our Dixon resultant algorithm. Subroutine BMStep (Subroutine 2) was designed to get these univariate polynomials when it receives an input prime p, and an input array J containing a sequence of coefficients from the univariate auxiliary rational functions in z such that $|J| = i$ 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 degree condition

$$\deg(\lambda) < \frac{i}{2} \tag{29}$$

is satisfied, and the number of roots of $\lambda(z)$ over $\mathbb{Z}_p$ yields the number of terms in $K_r(f_{i,k}) \in \mathbb{Z}_p[y]$ or $K_r(g_{i,k}) \in \mathbb{Z}_p[y]$ to be interpolated with high probability. However, it is possible that an incorrect $\lambda(z)$ is produced even if the condition (29) is satisfied. Thus, the wrong number of terms, and consequently the wrong polynomials $f_{i,k}$ or $g_{i,k}$ are obtained. To obtain a failure probability bound, we state the following result proved in Hu [18].

Theorem 42. *[18, 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*

$$1 - \frac{t(t+1)\deg(f)}{2(p-1)}.$$

Following the definition of Kaltofen, Lee and Lobo in [24], 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 $\lambda(z)$ and the correct number of terms whenever we run the BMA on an input of length i containing a sequence of points with $i > 2t$.

In practice, an input sequence of length $2t$ would yield a feedback polynomial of degree t (which is also the number of terms in the polynomial f to be interpolated). Therefore, our stopping condition (29) definitely obeys Theorem 42 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 2. As we have said earlier, the condition (29) may be satisfied, but still the wrong feedback polynomial is obtained because a zero discrepancy is encountered when $i < 2t$.

We give the following example to illustrate this possible failure.

Example 43. *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* $\hat{s} = 0$. *By computing* $v_j = f(\alpha^{\hat{s}+j})$ *for* $0 \leq j \leq 12$, *we obtain*

$$v = [59, 84, 8, 0, 64, 0, 96, 64, 94, 76, 85, 88].$$

Define $W_i := [v_1, v_2, \ldots, v_i]$. *Running the BMA on inputs* W_i *for points* $i = 2, 4, 6, 10, 12$ *yield the feedback polynomials* $\lambda(z)$ *recorded in the following table.*

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, Line 2 of Subroutine 2 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 algorithm at termination when we check if the returned answer is incorrect. Our Dixon resultant algorithm also checks that # roots of $\lambda(z)$ is equal to the degree of $\lambda(z)$ (see Line 4 of Subroutine 2).

Theorem 44. *Let N_a be the number of auxiliary rational functions needed by Algorithm 5 to interpolate the monic square-free factor S. If prime p is chosen at random from the list of pre-computed primes $\mathbb{P}_s$ and $p_{s_{\min}} = \min(P_s)$ then the probability that Algorithm 5 returns FAIL or an incorrect answer in Lines 28 or 29 or 31 or 32 is at most*

$$\frac{nmsd_x(N_a+1)^2(1+nsD_y)^{m+1}}{p-1}.$$

Proof. Since N_a is the required number of auxiliary rational functions needed by Algorithm 5, it follows that Line 2 of Subroutine 2 will never return FAIL. However, the feedback polynomial $\lambda(z) \in \mathbb{Z}_p[z]$ generated by the Berlekamp-Massey Algorithm in Subroutine 2 to find $K_r(f_{i,k}) \in \mathbb{Z}_p[y]$ or $K_r(g_{i,k}) \in \mathbb{Z}_p[y]$ might be wrong if the number of roots of $\lambda(z)$ is not equal to $\deg(\lambda)$ or after inverting the Kronecker substitution K_r, $\deg(f_{i,k}, y_j)$ or $\deg(g_{i,k}, y_j)$ is greater than $\max_{k=0}^{T-1}(\deg(f_k, y_j), \deg(g_k, y_j))$ so Line 4 or Line 7 of Subroutine 2 will return FAIL which causes Algorithm 5 to return FAIL in either Lines 28 or 29 or 31 or 32. Even if Line 4 or Line 7 of Subroutine 2 does not return FAIL, we might still have a feedback polynomial that terminates too early, so an incorrect $K_r(f_{i,k}) \in \mathbb{Z}_p[y]$ or $K_r(g_{i,k}) \in \mathbb{Z}_p[y]$ may be produced.

Using Theorem 42, the probability of getting FAIL or an incorrect $K_r(f_{i,k})$ or $K_r(g_{i,k})$, for all i and k in Subroutine 2 is at most

$$\begin{aligned}
&\sum_{k=0}^{T-1} \frac{\sum_{i=0}^{\deg(f_k)} \#f_{i,k}(\#f_{i,k}+1)\deg(K_r(f_{i,k})) + \sum_{i=0}^{\deg(g_k)} \#g_{i,k}(\#g_{i,k}+1)\deg(K_r(g_{i,k}))}{2(p-1)} \\
&\leq \sum_{k=0}^{T-1} \frac{\sum_{i=0}^{\deg(f_k)} N_a(N_a+1)(1+nsD_y)^m}{2(p-1)} + \frac{\sum_{i=0}^{\deg(g_k)} N_a(N_a+1)(1+nsD_y)^m}{2(p-1)} \\
&\leq \frac{(N_a+1)^2(1+nsD_y)^m \sum_{k=0}^{T-1}(1+\deg(f_k))}{2(p-1)} + \frac{(N_a+1)^2(1+nsD_y)^m \sum_{k=0}^{T-1}(1+\deg(g_k))}{2(p-1)} \\
&\leq \frac{2T(N_a+1)^2(1+nsD_y)^m(1+nmsD_y)}{2p-1} \leq \frac{nmsd_x(N_a+1)^2(1+nsD_y)^{m+1}}{p-1}
\end{aligned}$$

since $N_a \geq \#K_r(f_{i,k}), \#K_r(g_{i,k})$, and $(1+nsD_y)^m \geq \deg(K_r(f_{i,k})), \deg(K_r(g_{i,k}))$, and $T \leq \deg(R, x_1) \leq nsd_x$ by Theorem 14, and our result follows. □

5.8. Monomial Evaluations

We have to solve for the coefficients of the polynomials $f_{i,k}$ and $g_{i,k}$ in Algorithm 8, when more primes are required to interpolate the monic square-free factor S. Algorithm 8 uses the support obtained from the first image to solve for the coefficients of a new image of the monic square-free factor 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 25 and 37 of Algorithm 8 both detect the occurrence of getting the same monomial evaluation. Thus, we need to obtain a failure probability bound for this case.

Lemma 45. *Let q be a prime chosen at random from the list of pre-computed primes $\mathbb{P}$ to be used by Algorithm 8 in order to get a new image of the monic square-free factor S. Let $p_{\min} = \min(\mathbb{P})$. Let $f_{i,k}, g_{i,k}, f_k, g_k$ be as defined in* (17) *and* (18)*. Let $\hat{N}_{\max} = \max_{k=0}^{T-1} \left(\max_{i=0}^{\deg(f_k)} \{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)} \{\#g_{i,k}\} \right)$. Then the probability that Algorithm 8 returns FAIL in Line 25 or 37 is at most*

$$\frac{nsd_x \hat{N}_{\max}^2 (nmsD_y + 1)^2}{(q-1)}.$$

Proof. Let the support of $f_{i,k}$ 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 $\hat{m}_j = H_j(\alpha)$ be the j-th monomial evaluation where $\alpha \in (\mathbb{Z}_q \setminus \{0\})^m$ is the evaluation point picked at random in Line 13. By Lemma 2, we have

$$\Pr[\hat{m}_l = \hat{m}_j : 1 \leq l \neq j \leq \#f_{i,k}] = \Pr[J(\hat{Y}_i) = 0] \leq \frac{\binom{\#f_{i,k}}{2} \deg(f_{i,k})}{q-1}.$$

The same argument can be repeated for the $g_{i,k}$ polynomials. Thus, if the monomial evaluations obtained in Line 25 of Algorithm 8 or the monomial evaluations obtained in Line 24 of Subroutine 6 are not distinct, since $\deg(f_{i,k}), \deg(g_{i,k}) \leq \sum_{k=1}^{m} \deg(R, y_k) \leq nmsD_y$ and $T \leq \deg(R, y_1) \leq nsd_x$ by Theorem 14, we have

$$\begin{aligned}
&\Pr[\text{Algorithm 8 returns FAIL in Line 25 or 37}] \\
&\leq \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(f_k)} \binom{\#f_{i,k}}{2} \deg(f_{i,k})}{q-1} + \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(g_k)} \binom{\#g_{i,k}}{2} \deg(g_{i,k})}{q-1} \\
&\leq \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(f_k)} \hat{N}_{\max}^2 nmsD_y}{2(q-1)} + \frac{\sum_{k=0}^{T-1} \sum_{i=0}^{\deg(g_k)} \hat{N}_{\max}^2 nmsD_y}{2(q-1)} \\
&\leq \frac{2T\hat{N}_{\max}^2 nmsD_y(nmsD_y + 1)}{2(q-1)} \leq \frac{nsd_x \hat{N}_{\max}^2 (nmsD_y + 1)^2}{(q-1)}.
\end{aligned}$$

□

5.9. Univariate Rational Functions without a Kronecker Substitution

We remind the reader that Algorithm 8 does not use a Kronecker substitution K_r as it uses the support discovered by Algorithm 5.

Theorem 46. *Let q be an additional prime chosen at random from the list of pre-computed primes $\mathbb{P}$ to be used by Algorithm 8, in order to get a new image of the monic square-free factor S. Let $p_{\min} = \min(\mathbb{P})$ 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 $f_{i,k}, g_{i,k}, f_k, g_k$ are as defined in* (17)

and (18). *The probability that Algorithm 8 returns FAIL in Line 29 is at most*

$$\frac{2n^2s^2md_xD_y\hat{N}_{\max}}{q-1}+\frac{n^3s^3m^2D_y^2d_x\hat{N}_{\max}}{q-1}+\frac{2\hat{N}_{\max}nsd_x\log_{p_{\min}}(\|S\|_\infty)}{|\mathbb{P}|}.$$

Proof. Similar to Theorem 40, we have three causes of FAIL in Line 29 of Algorithm 8. The failure causes are the presence of unlucky 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 27. Note that auxiliary rational functions B_j are different from the A_j's interpolated in Algorithm 5 because a Kronecker substitution is not used. Let $\Delta=\prod_{k=0}^{T-1}\mathrm{LC}(f_k^\beta,z)\mathrm{LC}(g_k^\beta,z)\in\mathbb{Z}_q[y_1,y_2,\ldots,y_m]$ where

$$\frac{f_k^\beta(y_1,y_2,\ldots,y_m,z)}{g_k^\beta(y_1,y_2,\ldots,y_m,z)}=\frac{f_k(y_1z+\beta_1,\ldots,y_mz+\beta_m)}{g_k(y_1z+\beta_1,\ldots,y_mz+\beta_m)}.$$

Observe that $\deg(\Delta)=\sum_{k=0}^{T-1}\deg(f_k)+\deg(g_k)\leq 2T\sum_{k=1}^{m}\deg(R,y_k)\leq 2nsd_x(nsmD_y)\leq 2n^2s^2mD_yd_x$. 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 evaluation 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, $\hat{Y}_j$ is unlucky $\iff\Delta(\hat{Y}_j)=0$. Therefore,

$$\Pr[\hat{Y}_j\text{ in Line 15 is unlucky for }0\leq j\leq\hat{N}_{\max}-1]\leq\frac{\hat{N}_{\max}\deg(\Delta)}{q-1}\leq\frac{2\hat{N}_{\max}n^2s^2mD_yd_x}{q-1}.$$

Thus, using (27) and (28), we get

$$\Pr[\text{basis shift }\beta\text{ picked at random in Line 10 is bad}]\leq\frac{n^3s^3m^2D_y^2d_x\hat{N}_{\max}}{q-1},$$

and

$$\Pr[\text{prime }q\text{ is unlucky for any }B_j\text{ where }1\leq j\leq\hat{N}_{\max}]\leq\frac{2\hat{N}_{\max}nsd_x\log_{p_{\min}}\|S\|_\infty}{|\mathbb{P}|}.$$

Adding the above three probabilities completes our proof.

□

Remark 47. *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 5 or many subsequent primes in Algorithm 8 will not be accounted for, because Monagan's maximal quotient reconstruction algorithm [30] is used in our implementation, and it will always succeed with a probability of one when the input prime p or product of our input primes $p=\prod_{q\in P}q>9h^2$ where $h=\max_{k=0}^{T-1}\left(\max_{\frac{n_{k_1}}{d_{k_1}}\in f_k\ \text{and}\ \frac{n_{k_2}}{d_{k_2}}\in g_k}(|n_kd_k|)\right)$.*

5.10. Main Results

Our main technical results are presented in this section.

Theorem 48. *Suppose Algorithm 5 only needs one prime p to interpolate the monic square-free factor S. Let N_a be the number of auxiliary rational functions needed to interpolate S. If all the degrees pre-computed in Lines 1-6 are correct and p is selected at random from the pre-computed list of primes $\mathbb{P}_s$ and $p_{s_{\min}} = \min(\mathbb{P}_s)$ then the probability that Algorithm 5 returns FAIL is at most*

$$\frac{4N_a n^2 s^2 m D_y d_x \log_{p_{s_{\min}}}(\|S\|_\infty \|R\|_\infty) + 2N_a nsd_x \log_{p_{s_{\min}}} \|S\|_\infty}{|\mathbb{P}_s|} + \frac{9N_a n^3 s^3 m^2 D_y^2 d_x}{p_{s_{\min}} - 1} + \frac{nsd_x(1+nsD_y)^m(2+m(N_a+1)^2(1+nsD_y))}{p_{s_{\min}}-1}.$$

Proof. In Lemma 29, the number of points needed to perform a univariate rational interpolation $e_{\max} \leq 4nsmD_y$. Thus, the probability that Algorithm 8 returns FAIL in Lines 20 or 25 or 30 or 33 is at most

$$\underbrace{\frac{8N_a n^3 s^3 m^2 D_y^2 d_x}{p} + \frac{4N_a n^2 s^2 m d_x D_y \log_{p_{s_{\min}}}(\|S\|_\infty \|R\|_\infty)}{|\mathbb{P}_s|}}_{\text{by Lemma 29}} + \underbrace{\frac{nmsd_x(N_a+1)^2(1+nsD_y)^{m+1}}{p-1}}_{\text{by Theorem 44}}$$
$$+ \underbrace{\frac{N_a n^3 s^3 m^2 D_y^2 d_x}{p-1} + \frac{2N_a nsd_x \log_{p_{s_{\min}}} \|S\|_\infty}{|\mathbb{P}_s|} + \frac{2nsd_x(1+nsD_y)^m}{p-1}}_{\text{by Theorem 40}}.$$

Our result follows by simplifying the above bounds and using $p \geq p_{s_{\min}}$. □

Theorem 49. *Suppose Algorithm 5 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 $\mathbb{P}$ to be used by Algorithm 8 and $p_{\min} = \min(\mathbb{P})$. Let N_a be the number of auxiliary rational functions needed to interpolate S. Then the probability that Algorithm 8 returns* FAIL is at most

$$\frac{4N_a n^2 s^2 m D_y d_x \log_{p_{s_{\min}}}(\|S\|_\infty \|R\|_\infty) + 2N_a nsd_x \log_{p_{\min}} \|S\|_\infty}{|\mathbb{P}|} + \frac{9N_a n^3 s^3 m^2 D_y^2 d_x}{(p_{\min}-1)} + \frac{nsd_x \hat{N}_a^2 (nmsD_y+1)^2 + 2n^2 s^2 m d_x D_y N_a}{(p_{\min}-1)}.$$

Proof. Recall that the number of points needed to perform univariate rational interpolation is $e_{\max} = 2 + \max_{k=0}^{T-1}(\deg(f_k) + \deg(g_k)) \leq 4nsmD_y$. Also, $N_a \geq \max_{k=0}^{T-1}(\max_{i=0}^{\deg(f_k)}\{\#f_{i,k}\}, \max_{i=0}^{\deg(g_k)}\{\#g_{i,k}\})$. Thus, the probability that Al-

gorithm 8 returns FAIL in Lines 18 or 25 or 29 or 37

$$\leq \underbrace{\frac{8N_a n^3 s^3 m^2 D_y^2 d_x}{q} + \frac{4N_a n^2 s^2 m d_x D_y \log_{p_{s_{\min}}}(\|S\|_\infty \|R\|_\infty)}{|\mathbb{P}|}}_{\text{by Lemma 29}} + \underbrace{\frac{nsd_x \hat{N}_a^2 (nmsD_y+1)^2}{(q-1)}}_{\text{by Lemma 45}}$$
$$+ \underbrace{\frac{N_a n^3 s^3 m^2 D_y^2 d_x}{q-1} + \frac{2N_a nsd_x \log_{p_{\min}}\|S\|_\infty}{|\mathbb{P}|} + \frac{2n^2 s^2 m d_x D_y N_a}{q-1}}_{\text{by Lemma 46}}$$

and our result follows using $q \geq p_{\min}$. □

Lemma 50. *Suppose Algorithms 5 and 8 are modified to interpolate N 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 primes needed by Algorithm 5 are selected from the list of smooth primes $\mathbb{P}_s$ such that $p_{s_{\min}} = \min(\mathbb{P}_s)$ and the (not necessarily smooth) primes if needed by Algorithm 8 are selected at random from the list of pre-computed primes $\mathbb{P}$ such that $p_{\min} = \min(\mathbb{P})$ and $p_{\min} \geq p_{s_{\min}}$.*

A. If Algorithm 5 only needs one prime to interpolate all the monic square-free factors then the probability that Algorithm 5 returns FAIL is at most

$$\frac{4N_a n^2 s^2 m D_y d_x \log_{p_{s_{\min}}}(\max_{j=1}^N \|R_j\|_\infty \|R\|_\infty) + 2N_a nsd_x \log_{p_{s_{\min}}} \max_{j=1}^N \|R_j\|_\infty}{|\mathbb{P}_s|}$$
$$+ \frac{9N_a n^3 s^3 m^2 D_y^2 d_x}{p_{s_{\min}} - 1} + \frac{nsd_x(1+nsD_y)^m(2+m(N_a+1)^2(1+nsD_y))}{p_{s_{\min}}-1}.$$

B. Suppose Algorithm 5 needs an additional prime to interpolate all the monic square-free factors R_j. Then the probability that Algorithm 8 returns FAIL

$$\leq \frac{4N_a n^2 s^2 m D_y d_x \log_{p_{s_{\min}}}(\max_{j=1}^N \|R_j\|_\infty \|R\|_\infty) + 2N_a nsd_x \log_{p_{s_{\min}}} \max_{j=1}^N \|R_j\|_\infty}{|\mathbb{P}_s|}$$
$$+ \frac{9N_a n^3 s^3 m^2 D_y^2 d_x}{p_{s_{\min}} - 1} + \frac{nsd_x \hat{N}_a^2 (nmsD_y+1)^2 + 2n^2 s^2 m d_x D_y N_a}{p_{s_{\min}}-1}.$$

Now, we give a failure probability bound for our probabilistic test, which verifies the correctness of the output of our Dixon resultant algorithm.

Theorem 51. *Let H_p be the product of all the primes needed by our Dixon resultant algorithm (Algorithm 5) to reconstruct the coefficients of its returned output using rational number reconstruction. If the prime used by our probabilistic test (Algorithm 9) is selected at random from the list of primes $\mathbb{P}$ and*

$p_{\min} = \min(\mathbb{P})$ *then the probability that our probabilistic test* fails is at most

$$\frac{2n^2s^2md_xD_y}{p_{\min}} + \frac{nsd_x \log_{p_{\min}}\left(\frac{2||S||_\infty\sqrt{H_p}}{3}\right)}{|\mathbb{P}|}.$$

Proof. Let $S = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1,\ldots,y_m)}{g_k(y_1,\ldots,y_m)} x_1^{d_k} \in \mathbb{Z}(y_1, y_2, \ldots, y_m)[x_1]$ defined in (17) be the correct monic square-free factor of R with $f_k, g_k \neq 0$. Let

$$\hat{S} = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{F_k(y_1, \ldots, y_m)}{G_k(y_1, \ldots, y_m)} x_1^{d_k} \in \mathbb{Z}(y_1, y_2, \ldots, y_m)[x_1]$$

be the output of our Dixon resultant algorithm, and suppose $S \neq \hat{S}$. Since

$$\deg(F_k, y_i), \deg(G_k, y_i) \leq \max_{k=0}^{T-1}(\max(\deg(f_k, y_i), \deg(g_k, y_i)))$$

because of our check in Line 7 of Subroutine 2, using Theorem 14, we have

$$\deg(F_k, y_i), \deg(G_k, y_i) \leq \deg(R, y_i) \leq nsD_y.$$

Let

$$H = S - \hat{S} = \sum_{k=0}^{T-1} \frac{(f_kG_k - F_kg_k)}{g_kG_k} x_1^{d_k} \in \mathbb{Z}(y_1, y_2, \ldots, y_m)[x_1].$$

Recall that our Dixon resultant algorithm uses Monagan's maximal quotient reconstruction algorithm which succeeds with a probability of one if $H_p > 9h^2$ where $h = \max_{k=0}^{T-1}\left(\max_{\frac{n_{k_1}}{d_{k_1}} \in F_k \text{ and } \frac{n_{k_2}}{d_{k_2}} \in G_k}(|n_kd_k|)\right)$. Therefore, $h < \frac{\sqrt{H_p}}{3}$.

Let q be the random prime selected in Line 2 of Algorithm 9 and let $\alpha \in \mathbb{Z}_q^m$ be the random point selected in Line 3. Notice that Algorithm 9 fails if $S \neq \hat{S}$ but $H(\alpha) = 0$ or $q|H$. Let

$$E = \prod_{k=0}^{T-1}(f_kG_k - F_kg_k) \in \mathbb{Z}[y_1, \ldots, y_m].$$

Observe that $H(\alpha) = 0 \iff E(\alpha) = 0$ and $g_k(\alpha) \neq 0$ and $G_k(\alpha) \neq 0$. Furthermore, $q|H \iff q|(f_kG_k - F_kg_k)$ and $q \nmid g_k$ and $q \nmid G_k$ for all k. We remind the reader that Line 9 of Algorithm 9 ensures that the conditions $g_k(\alpha) \neq 0$, $G_k(\alpha) \neq 0$, $q \nmid g_k$ and $q \nmid G_k$ are met. Since $T \leq nsd_x$ by Theorem 14, it follows that the probability that our probabilistic test fails is at most

$$
\begin{aligned}
&\Pr[E(\alpha)=0] + \Pr[\,q \text{ divides any } f_kG_k - F_kg_k \text{ in } H] \\
&\leq \frac{\deg(E)}{q} + \Pr[\,q \text{ divides one term in any } f_kG_k - F_kg_k \text{ of } H] \\
&\leq \frac{\sum_{k=0}^{T-1} \deg(f_kG_k - F_kg_k)}{q} + \frac{T\log_{p_{\min}}||f_kG_k - F_kg_k||_\infty}{|\mathbb{P}|} \\
&\leq \frac{\sum_{k=0}^{T-1}\sum_{i=1}^{m} \deg(f_kG_k - F_kg_k, y_i)}{q} + \frac{T\log_{p_{\min}}||f_kG_k - F_kg_k||_\infty}{|\mathbb{P}|} \\
&\leq \frac{Tm(2nsD_y)}{q} + \frac{T\log_{p_{\min}}(||f_k||_\infty||G_k||_\infty + ||F_k||_\infty||g_k||_\infty)}{|\mathbb{P}|} \\
&\leq \frac{2n^2s^2md_xD_y}{p_{\min}} + \frac{nsd_x\log_{p_{\min}}\left(\frac{2||S||_\infty\sqrt{H_p}}{3}\right)}{|\mathbb{P}|}.
\end{aligned}
$$

□

5.11. The cost and number of black box probes required by our algorithm

In this section, we estimate the cost of a black box probe 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 was always correct, i.e., $\text{supp}(\hat{S}) = \text{supp}(S)$.

Theorem 52. *Let M be the non-singular $s \times s$ Dixon matrix obtained from a parametric polynomial system $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\} \subset \mathbb{Z}[y_1, \ldots, y_m][x_1, \ldots, x_n]$ such that $\#M_{ij} \leq T_{\max}$ and $\|M_{ij}\|_\infty \leq B^h$. Let $\hat{D} = \max(d_x, D_y)$. Let p be a prime from the list of primes $\mathbb{P}$ satisfying $B < p < 2B$. Then a black box probe costs $O(s^2hT_{\max} + s^2nm\hat{D}T_{\max} + s^3)$ arithmetic operations in $\mathbb{Z}_p$.*

Proof. Let $M_{ij} = \sum_{k=1}^{T_{\max}} a_k M_{ij_k}(x_1, y_1, \ldots, y_m)$. The cost of computing M_{ij} mod p is $O(hT_{\max})$, so M mod p costs $O(s^2hT_{\max})$. Let $\alpha \in \mathbb{Z}_p^{m+1}$ be an evaluation point. The number of multiplications performed to compute $M_{ij_k}(\alpha)$ is $mn\hat{D}$ since the maximum partial degree of M_{ij} is at most $n\hat{D}_{\max}$. For $1 \leq k \leq T_{\max}$, all monomial evaluations $M_{ij_k}(\alpha)$ are computed using $O(nm\hat{D}T_{\max})$ multiplications and $T_{\max}$ multiplications for the product $a_kM_{ij_k}(\alpha)$. Hence the cost of evaluating M is $O(s^2nm\hat{D}T_{\max})$. The cost of the determinant computation using Gaussian elimination over $\mathbb{Z}_p$ is $O(s^3)$. Thus one black box probe costs $O(s^2T_{\max}h + s^2mn\hat{D}T_{\max} + s^3)$.

□

Theorem 53. *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 the monic square-free factor S are as defined in (17), (18). *Let H be the number of primes needed by Algorithm 8 to reconstruct the coefficients of S using rational number reconstruction. The number of black box probes required by our Dixon resultant 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 23 of Algorithm 5, we need to use $e_{\max}$ coefficients from the monic polynomial images of R in x_1 obtained in Line 19 of Algorithm 5. The size of the supports $\#f_{i,k}$ and $\#g_{i,k}$ are unknown, and they will be discovered in Line 1 of Subroutine BMStep using the Berlekamp-Massey Algorithm. 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 $\#g_{j,k}$ where $0 \le i < \deg(f)$ and $0 \le j < \deg(g)$. 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. Concluding Remarks

We have developed a new sparse rational function interpolation method which uses a Kronecker substituion and a new set of randomized points in the Cuyt and Lee's method and the Ben-Or/Tiwari algorithm. 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 tested our new Dixon resultant code on many real parametric polynomial systems that emerged from practical problems. Our benchmarks showed that our new algorithm is much faster than other algorithms (Zippel's sparse interpolation, Gentleman & Johnson algorithm and Dixon-EDF method) for computing the Dixon resultant R in expanded form whenever R is large while it has relatively small square-free factors or a large polynomial content.

We have identified all the causes of failure in our new Dixon resultant algorithm and presented a thorough analysis of failure probabilities.

Acknowledgements

This work was supported by the National Science and Engineering Research Council of Canada.

References

[1] Atti, N. B. and Lombardi, H. and Diaz-Toca G. M.: The Berlekamp-Massey algorithm revisited. *AAECC* ***17***, 4 (2006), pp. 75–82.

[2] Aubry, P., Lazard, D., Moreno Maza, M. On the Theories of Triangular Sets. *J. Symbolic Comp.* **28**:105–124, Springer, 1999.

[3] Bareiss, E.H.: Sylvesters' identity and multistep integer preserving Gaussian elimination. *Math. Comp.* **22**(103):565–578, 1968.

[4] Ben-Or, M., and Tiwari, P.: A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. *Proceedings of STOC '20*, pp. 301–309, ACM, 1988.

[5] Berkowitz, S.J.: On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.* **18**(3):147–150, 1984.

[6] Cox, D., Little, J., and O'Shea, D. *Ideals, Varieties and Algorithms* 3rd ed. Springer, 2007.

[7] Cuyt, A., and Lee, W.-S.: Sparse Interpolation of Multivariate Rational Functions. *J. Theoretical Comp. Sci.* ***412***, pp. 1445–1456, Elsevier, 2011.

[8] Dixon, A.: On a form of the Eliminant of Two Quantics. Proceedings of the London Mathematical Society **2**, (1908), pp. 468–478.

[9] Dixon, A.: The eliminant of three Quantics in Two Independent Variables. Proceedings of the London Mathematical Society **2**, (1909), pp. 49–69.

[10] Edmonds, J.: Systems of Distinct Representatives and Linear Algebra *J. Research, Mathematics and Mathematical Physics* **71B**(4):241–245, 1967.

[11] Gelfond A.: *Transcendental and Algebraic Numbers.* GITTL, Moscow, 1952; English translation by Leo F. Boron, Dover, New York, 1960

[12] Gentleman, W. M., and Johnson, S. C.: The Evaluation of Determinants by Expansion by Minors and the General Problem of Substitution. *Mathematics of Computation* **28**(126):543–548, 1974.

[13] Kapur, D., Saxena, T., and Yang, L.: Algebraic and Geometric Reasoning using Dixon Resultants. *Proceedings of ISSAC '94* , pp. 99–107, ACM, 1994.

[14] Kapur, D., and Saxena, T.: Comparison of Various Multivariate Resultant formulations. *Proceedings of ISSAC '95* , pp. 187–194, ACM, 1995.

[15] Gerhard, J., and Von zur Gathen, J.: *Modern Computer Algebra.* Cambridge University Press, 2013.

[16] Möller, N., and Grandlund T.: Improved Division by Invariant Integers. *Transactions on Computers* **60**(2):165–175, IEEE, 2011.

[17] Hu, J., and Monagan, M.: A fast parallel sparse polynomial GCD algorithm. *Journal of Symbolic Computation*, **105**(1):28–63, 2021

[18] Hu, J.: Computing polynomial greatest common divisors using sparse interpolation. PhD Thesis, Simon Fraser University, 2018.

[19] Jinadu, A., and Monagan, M.: An Interpolation Algorithm for computing Dixon Resultants. *Proceedings of CASC '2022,* LNCS **13366**:185-205, Springer, 2022.

[20] Jinadu, A., and Monagan, M.: A new interpolation algorithm for computing Dixon Resultants. *Commun. in Computer Algebra* **56**(3):88-91, 2022.

[21] Jinadu, A.: Solving parametric systems using Dixon resultants and sparse interpolation tools. PhD Thesis, Simon Fraser University, 2023.

[22] Kaltofen, E. Fifteen years after DSC and WLSS2. In *Proc. of PASCO 2010*, pp. 10–17, ACM, 2010.

[23] Kaltofen, E. , and Trager, B. 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, Elsevier, 1990.

[24] Kaltofen, E. , Lee, W. , and Lobo, A.: Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. *Proceedings of ISSAC 2000, pp. 192–201, ACM, 2000.*

[25] Lewis, R.: Comparison of the greatest common divisor (GCD) in several systems, 2004. URL https://home.bway.net/lewis/fermat/gcdcomp.

[26] Lewis, R.: Dixon-EDF: The Premier Method for Solution of Parametric Polynomial Systems. *Special Sessions in ACA*, pp. 237–256, Springer

[27] Lewis, R.: Resultants, Implicit Parameterizations, and Intersections of Surfaces. *Proceedings of ICMS 2018*, LNCS **10931**:310–318, Springer, 2018.

[28] Lewis, R.: Image Analysis: Identification of Objects via Polynomial Systems. Proceedings of ICMS 2018, LNCS **10931**:305–309, Springer, 2018.

[29] Lewis, R.: Private Communication, 2018.

[30] Monagan, M.: Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '2004*, pp. 243–249, ACM, 2004.

[31] Monagan, M. and Pearce R. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Communications of Computer Algebra*, **48**(4):166–186, ACM, 2014.

[32] Minimair, M. Computing the Dixon Resultant with the Maple Package DR. *Applications of Computer Algebra. ACA 2015. Springer Proceedings in Mathematics & Statistics, vol.* ***195****,* Springer.

[33] Schwartz, J: Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, **27**:701–717 (1980)

[34] Shanks, D: Class number, a theory of factorization, and genera In Proc. Symp. Math. Soc., **20**:415–440, 1971.

[35] S. Pohlig and M. Hellman: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. on Information Theory*, **24**:106–110, IEEE, 1978.

[36] Schmidt, W.: Equations over finite fields: an elementary approach. **vol. 536**. Springer, 2006.

[37] Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. *Proceedings of EUROSAM '79* , pp. 216–226, Springer, 1979.

[38] Zippel, R.: Interpolating Polynomials from their Values. *J. Symbolic Computation* **9**:375–403, Springer, 1990.

Appendix A

The robot arms system from Lewis [26] in Maple and Magma input format. There are four unknowns x_1, x_2, x_3, x_4 and seven parameters $y_1, y_2, y_3, y_4, y_5, y_6, y_7$.

```
f1 := (x2^2+1)*(x1^2+1)*(x4^2+1)*(x3^2+1)*y1+2*y3*x1^2*x4^2*x3^2
-2*y3*x2^2*x4^2*x3^2+2*y4*x2^2*x1^2*x3^2-2*y4*x2^2*x1^2*x4^2+2*y3*x1^2*x3^2
-2*y3*x2^2*x3^2+2*y4*x1^2*x3^2+2*y4*x2^2*x3^2+2*y3*x1^2*x4^2-2*y3*x2^2*x4^2
-2*y4*x1^2*x4^2-2*y4*x2^2*x4^2+2*y4*x3^2-2*y4*x4^2+2*y3*x1^2-2*y3*x2^2;

f2 := 2*x1^2*x2^2*x3^2*x4*y4-2*x1^2*x2^2*x3*x4^2*y4+2*x1^2*x2*x3^2*x4^2*y3
-2*x1*x2^2*x3^2*x4^2*y3-2*x1^2*x2^2*x3*y4+2*x1^2*x2^2*x4*y4+2*x1^2*x2*x3^2*y3
+2*x1^2*x2*x4^2*y3+2*x1^2*x3^2*x4*y4-2*x1^2*x3*x4^2*y4-2*x1*x2^2*x3^2*y3
-2*x1*x2^2*x4^2*y3-2*x1*x3^2*x4^2*y3+2*x2^2*x3^2*x4*y4-2*x2^2*x3*x4^2*y4
+2*x2*x3^2*x4^2*y3+2*x1^2*x2*y3-2*x1^2*x3*y4+2*x1^2*x4*y4-2*x1*x2^2*y3
-2*x1*x3^2*y3-2*x1*x4^2*y3-2*x2^2*x3*y4+2*x2^2*x4*y4+2*x2*x3^2*y3+2*x2*x4^2*y3
+2*x3^2*x4*y4-2*x3*x4^2*y4-2*x1*y3+2*x2*y3-2*x3*y4+2*x4*y4;

f3 := y3+y4-y5-y5*y2^2*x1^2*x3^2-y4*y2^2*x1^2*x3^2-y3*y2^2*x1^2*x3^2
-4*y5*y2*x1^2*x3-(x1^2+1)*(x3^2+1)*(y2^2+1)*y6+y5*y2^2*x1^2-y4*y2^2*x3^2
-y5*y2^2*x3^2+y3*y2^2*x3^2-y3*y2^2*x1^2-4*y5*y2*x3+y5*x1^2*x3^2
-y4*x1^2*x3^2+y4*y2^2*x1^2-y3*x1^2*x3^2-y4*x3^2-y3*x1^2+y5*x3^2
+y3*x3^2-y5*x1^2+y4*x1^2+y5*y2^2+y4*y2^2+y3*y2^2;

f4 := -(x1^2+1)*(x3^2+1)*(y2^2+1)*y7+2*y3*y2^2*x1*x3^2+2*y4*y2^2*x1^2*x3
+2*y5*y2^2*x1^2*x3-2*y5*y2*x1^2*x3^2+2*y4*y2^2*x3+2*y5*y2^2*x3
```

```
+2*y3*y2^2*x1-2*y5*y2*x3^2+2*y5*y2*x1^2+2*y3*x1*x3^2+2*y4*x1^2*x3
-2*y5*x1^2*x3+2*y5*y2+2*y4*x3-2*y5*x3+2*y3*x1;
```