An Interpolation Algorithm for computing Dixon Resultants

Ayoola Jinadu and Michael Monagan

Department of Mathematics, Simon Fraser University
Burnaby, British Columbia, V5A 1S6, Canada
ajinadu@sfu.ca and mmonagan@sfu.ca

Abstract. Given a system of polynomial equations with parameters, we present a new algorithm for computing its Dixon resultant R. Our algorithm interpolates the monic square-free factors of R one at a time from monic univariate polynomial images of R using sparse rational function interpolation. In this work, we use a modified version of the sparse multivariate rational function interpolation algorithm of Cuyt and Lee. We have implemented our new Dixon resultant algorithm in Maple with some subroutines coded in C for efficiency. We present timing results comparing our new Dixon resultant algorithm with Zippel's algorithm for interpolating R and a Maple implementation of the Gentleman & Johnson minor expansion algorithm for computing R.

Keywords: Dixon Resultant, Parametric Polynomial Systems, Resultant, Sparse Rational Function Interpolation, Kronecker Substitution

1 Introduction

Let $X = \{x_1, x_2, \cdots, x_n\}$ denote the set of variables and let $Y = \{y_1, y_2, \cdots, y_m\}$ be the set of parameters with $n \geq 2$ and $m \geq 0$. Let $\mathcal{F} = \{f_1, f_2, \cdots, f_n\} \subset \mathbb{Q}[X, Y]$ be a parametric polynomial system where f_i is a polynomial in variables X with coefficients in $\mathbb{Q}[Y]$. Let $I = \langle f_1, f_2, \cdots, f_n \rangle$ be the ideal generated by $\mathcal{F}$. The Dixon resultant [5,6] of $\mathcal{F}$ in x_1 is the determinant of the Dixon matrix (See Section 2) and it is a polynomial in the elimination ideal $I \cap \mathbb{Q}[Y][x_1]$. It is used to eliminate $n-1$ variables from a polynomial system in n variables.

Let $R = \sum_{k=0}^{d} r_k(y_1, \cdots, y_m) x_1^k \in \mathbb{Q}[Y][x_1]$ be the Dixon resultant of $\mathcal{F}$ in x_1 where $d = \deg(R, x_1) > 0$. Let $C = \gcd(r_0, \cdots, r_d)$ be the polynomial content of R. In this paper we will compute the monic square-free factors of R. The monic square-free factorization of R is a factorization of the form $\hat{r} \prod_{j=1}^{l} R_j^j$ such that

1. $\hat{r} = C/L$ for some $L \in \mathbb{Q}[Y]$,
2. each R_j is monic and square-free in $\mathbb{Q}(Y)[x_1]$, i.e., $\gcd(R_j, R_j^{'}) = 1$, and
3. $\gcd(R_i, R_j) = 1$ for $i \neq j$.

This monic square-free factorization exists and it is unique [8, Section 14.6]. Note, the factors R_j are not necessarily irreducible over $\mathbb{Q}$. The monic square-free part S of R is the product of the monic square-free factors R_j, that is, $S = \prod_{j=1}^{l} R_j$.

In this paper we present a new Dixon resultant algorithm that interpolates the monic square-free factors R_j one at a time and does not interpolate R. We interpolate the R_j's because it is cheap to compute a square-free factorization of a monic image of R and the square-free factorization factors will be consistent from one image to the next with high probability. Interpolating the R_j's instead of R results in a huge gain because all unwanted repeated factors and the polynomial content are removed. The advantage of our algorithm over other known polynomial interpolation algorithms [2, 23] is that the number of polynomial terms in R_j to be interpolated is much less than in R. Furthermore, the number of primes used by our algorithm in the sparse interpolation step when we apply the Chinese remainder theorem is reduced. Thus the number of black box [1]probes required to interpolate the monic square-free factors R_j is much fewer than the number required to interpolate R. We give a real example from [14].

Example 1 *[14, robot arms system, page 17] Let*

$$C = \underbrace{-65536\left(al^2+1\right)^8 l_2^8 \left(al^2l_2^2+2al^2l_2l_3+al^2l_3^2+l_2^2-2l_2l_3+l_3^2\right)^4}_{polynomial\ content}$$

$$A_1 = t_1^2+1$$

$$A_2 = (al^2l_1^2+2al^2l_1x-al^2l_2^2-2al^2l_2l_3-al^2l_3^2+al^2x^2+al^2y^2+l_1^2+2l_1x-l_2^2 + 2l_2l_3-l_3^2+x^2+y^2)t_1^2+\left(-4al^2l_1y-4l_1y\right)t_1+al^2l_1^2-2al^2l_1x-al^2l_2^2 - 2al^2l_2l_3-al^2l_3^2+al^2x^2+al^2y^2+l_1^2-2l_1x-l_2^2+2l_2l_3-l_3^2+x^2+y^2$$

$$A_3 = (aa^2+2aal_2)t_1^2+aa^2-4aal_1+2aal_2+4l_1^2-4l_1l_2$$

$$A_4 = (aa^2-2aal_2)t_1^2+aa^2-4aal_1-2aal_2+4l_1^2+4l_1l_2$$

where $X = \{t_1, t_2, b_1, b_2\}$ are the variables, t_1 is the main variable and $Y = \{aa, al, l_1, l_2, l_3, x, y\}$ are the parameters. The Dixon resultant R of the robot arms system in t_1 has $6,924,715$ terms in expanded form and it factors as

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

Our new Dixon resultant algorithm computes R_1, R_2 and R_3 where $R_1 = A_1, R_2 = \text{monic}(A_2, t_1)$ and $R_3 = \text{monic}(A_3A_4, t_1)$. The largest coefficient of R_1, R_2 and R_3 is the leading coefficient of A_2 which has only 14 terms! Notice that R_1 and R_2 are irreducible over $\mathbb{Q}$ but R_3 is not.

Our motivation to investigate Dixon resultants stems from sets of parametric polynomial systems listed in [11,12,14,15]. Lewis tried to solve these polynomial systems using Gröbner bases and Triangular sets in Maple and Magma, but they often failed badly; they took a very long time to execute and often ran out

[1] A black box is a computer program that takes as input a list of integers together with a prime and outputs the evaluation of the represented object modulo the prime. Black box representations are space efficient. The represented object such as a polynomial, a rational function, and a determinant of a matrix of polynomials is assumed to be unknown. A function call to the black box is referred to as a black box probe.

of memory. The failure of these methods is due to the intermediate expression swell caused by the parameters. This led Lewis to develop the Dixon-EDF (Early Detection Factor) algorithm [14] which is a variant of the Gaussian elimination. It is a modified row reduction of the Dixon matrix that factors out the gcd of each pivot row at each step. The Dixon-EDF method is able to detect factors of the Dixon resultant early. One can interrupt it part way to switch to another method. Lewis often switches to the Gentleman & Johnson minor expansion algorithm [7] to finish the computation. The drawback of the Dixon-EDF method is that it is not automatic and expression swell may occur when computing in $\mathbb{Q}[Y, x_1]$.

Our first contribution is a new algorithm that computes the monic square-free factors R_j of R from monic univariate images in x_1 using sparse multivariate rational function interpolation to interpolate the coefficients of R_j in $\mathbb{Q}(Y)$ modulo primes and uses Chinese remaindering and rational number reconstruction [8,18] to recover the rational coefficients of R_j. We have modified the sparse rational function interpolation algorithm of Cuyt and Lee [4] for this purpose. The only interpolation method that has been applied to Dixon resultants that we are aware of was done by Kapur and Saxena in [12]. They used Zippel's sparse interpolation [23] to interpolate R. Zippel's method does $O(m\hat{D}t)$ black box probes for the first image modulo a prime, where m is the number of parameters, $\hat{D} = \deg(R, x_1) + \sum_{i=1}^{m} \deg(R, y_i)$ and $t = \#R$. But one has to recover the integer coefficients of R which may need more primes. Using the support of the result obtained for the first prime, the integer coefficients can be recovered using $O(t)$ probes to the black box for each subsequent prime [23].

Our second contribution is a Maple + C implementation of our algorithm. For our benchmark problem (*Heron5d* system [22]), the Gentleman & Johnson algorithm ran out of space (> 64GB), Zippel's algorithm takes more than 10^5 seconds and our new algorithm takes 23.12 seconds on 1 core.

We provide an overview of our Dixon resultant algorithm. Let $\hat{r} \prod_{j=1}^{l} R_j^j$ be the monic square-free factorization of R. For $1 \le j \le l$, our algorithm will compute each R_j in the form

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

where $\gcd(f_{jk}, g_{jk}) = 1$, $f_{jk}, g_{jk} \in \mathbb{Q}[y_1, y_2, \cdots, y_m]$ and $d_{T_j} = \deg(R_j, x_1)$.

If more primes are needed to recover the R_j's, one can set up a system of linear equations using the support found with the first prime to solve for the coefficients of f_{jk} and g_{jk} before doing Chinese remaindering. This method costs $O(\sum_{k=1}^{l} \sum_{k=1}^{T_j} d_{j,k}^{\mathbf{3}})$ arithmetic operations in $\mathbb{Z}_p$, where p is the prime and $d_{j,k} = \#f_{jk} + \#g_{jk}$ is the total number of unknowns in the k-th rational coefficient of R_j. Instead, we reduce this cost to $O(\sum_{j=1}^{l} \sum_{k=1}^{T_j} d_{j,k}^{\mathbf{2}})$ arithmetic operations in $\mathbb{Z}_p$ as follows. We pick α and β in $\mathbb{Z}_p^m$ at random, a shift $s \in [1, p-2]$ at random and probe the black box to compute

$$G(\alpha^i, x_1, z) := x_1^{d_{T_j}} + \sum_{k=0}^{T_j - 1} \frac{f_{jk}(z\beta_1 + \alpha_1^{s+i}, \cdots, z\beta_m + \alpha_m^{s+i})}{g_{jk}(z\beta_1 + \alpha_1^{s+i}, \cdots, z\beta_m + \alpha_m^{s+i})} x_1^{d_{j_k}} \in \mathbb{Z}_p(z)[x_1]$$

for $0 \leq i < N$ and $N = \max_{j=1}^{l} \max_{k=0}^{T_j-1} \{\#f_{jk}, \#g_{jk}\}$. Then for $1 \leq j \leq l$, we collect the $\#f_{jk}$ (or $\#g_{jk}$) rational coefficients modulo p from $G(\alpha^i, x_1, 0)$ and set up a shifted transposed Vandermonde system [9, 23] to solve for the the coefficients of f_{jk} and g_{jk} for each R_j.

In a preliminary stage of this work, we first designed our algorithm to interpolate the monic square-free part $S = \prod_{j=1}^{l} R_j$ from the monic univariate images of R in x_1. But we discovered that when $l > 1$, interpolating the R_j's instead of S often reduces the number of black box probes required. These savings are realized because there is a further reduction in the number of terms in the largest polynomial coefficient of R_j to be interpolated compared to the monic product S. Also, the same monic univariate images that yield the first monic square-free factor R_1 can re-used for subsequent monic square-free factors in R.

Table 1 contains the number of black box probes required for interpolating S versus interpolating the monic square-free factors R_j one at a time for the *robot arms* problem [14] and it shows a significant reduction in the number of black box probes when the main variable is t_1, t_2 or b_2.

Main variable	t_1	t_2	b_1	b_2
Interpolating square-free part S	$222,301$	$3,137,373$	$116,741$	$5,531,491$
Interpolating square-free-factors R_j one at a time	$19,241$	$1,210,889$	$116,741$	$1,335,853$
Savings in # of probes	$203,060$	$1,926,484$	0	$4,195,638$
# of terms in the largest coefficient of R_j	14	691	85	624
# of terms in the largest coefficient of S	106	$2,200$	85	$2,388$

Table 1: Interpolating S versus interpolating the square-free factors R_j

Notice in column b_1 that both methods used the same number of black box probes. This is because the number of terms in the largest polynomial coefficient of R_j and S are the same. Thus no savings is realized even though the number of the monic square-free factors for this case is more than 1.

Paper Outline

In Section 2, we present a Dixon resultant formulation for polynomial systems. In Section 3, we give an overview of the rational function interpolation algorithm of Cuyt and Lee [4] and we modify it to use Kronecker substitutions to combat the large prime and unlucky evaluation point problems that occurs when the adopted sparse polynomial algorithm in Cuyt and Lee's method is the Ben-Or/Tiwari sparse polynomial algorithm [2]. Our algorithms without their failure probability bounds are presented in Section 4. In Section 5, we explain how we evaluate the polynomial entries in the Dixon matrix which is the most expensive part of our algorithm and we compare our new Dixon resultant algorithm with a Maple implementation of the Gentleman & Johnson minor expansion algorithm and a Maple+C implementation of Zippel's algorithm to interpolate R on the parametric polynomial systems from [14, 15].

2 Dixon Resultants

Let $\mathbf{x}^{\alpha} = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ and let $\{\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n\}$ be a set of new variables. For each $i \in \{0, 1, 2, \cdots, n\}$, we define $\pi_i(\mathbf{x}^{\alpha}) = \bar{x}_1^{\alpha_1} \bar{x}_2^{\alpha_2} \cdots \bar{x}_i^{\alpha_i} x_{i+1}^{\alpha_{i+1}} x_{i+2}^{\alpha_{i+2}} \cdots x_n^{\alpha_n}$ such that $\pi_0(\mathbf{x}^{\alpha}) = \mathbf{x}^{\alpha}$. Extending the map π_i naturally to polynomials, we have

$$\pi_i(f(x_1, x_2, \cdots, x_n)) = f(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_i, x_{i+1}, x_{i+2} \cdots, x_n).$$

There are three major steps involved in computing the Dixon resultant of a polynomial system. The first step is to construct the cancellation matrix [5, 6]. We refer to the determinant of the cancellation matrix as the Dixon polynomial. The Dixon polynomial acts as the link between the cancellation matrix and the Dixon matrix. Although it is important to select the order of the $n-1$ variables to eliminate because the order affects the size and degree of the Dixon polynomial, we do not focus on the optimal order. Further information about the optimal order can be found in [3, 17].

Definition 2 *Given a polynomial system $\mathcal{F}$, let $X_e = \{x_2, \cdots, x_n\}$ be the set of variables to be eliminated and let x_1 be the main variable to appear in the Dixon resultant. Let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3 \cdots, \bar{x}_n\}$ be the set of the new variables corresponding to X_e. We define the $n \times n$ cancellation matrix*

$$\mathcal{C} = \begin{pmatrix} \pi_0(f_1(X_e)) & \pi_0(f_2(X_e)) \dots & \pi_0(f_n(X_e)) \\ \pi_1(f_1(X_e)) & \pi_1(f_2(X_e)) \dots & \pi_1(f_n(X_e)) \\ \vdots & \vdots & \vdots \\ \pi_{n-1}(f_1(X_e)) & \pi_{n-1}(f_2(X_e)) \dots & \pi_{n-1}(f_n(X_e)) \end{pmatrix}. \tag{1}$$

Definition 3 *Let $P = \prod_{i=1}^{n-1}(X_{e_i} - \overline{X}_{e_i})$ and let*

$$\Delta_{X_e} = \frac{\det(\mathcal{C})}{P}. \tag{2}$$

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

The determinant of the cancellation matrix $\det(\mathcal{C})$ is a multiple of the Dixon polynomial Δ_{X_e}. One must not compute Δ_{X_e} by expanding $\det(\mathcal{C})$ then dividing by P because $\det(\mathcal{C})$ which equals $P \times \Delta_{X_e}$, is much bigger than Δ_{X_e}, since there are 2^{n-1} terms in P when P is expanded. Instead, we follow Lewis [16] and create a new cancellation matrix $\hat{\mathcal{C}}$ using the identity

$$\text{Row } (\hat{\mathcal{C}}_1) = \text{Row}(\mathcal{C}_1), \quad \text{Row } (\hat{\mathcal{C}}_{i+1}) = \frac{\text{Row } (\mathcal{C}_{i+1}) - \text{Row } (\mathcal{C}_i)}{X_{e_i} - \overline{X}_{e_i}} \tag{3}$$

for $i = n-1, n-2 \cdots 1$ and then compute the determinant of $\hat{\mathcal{C}}$ which produces the Dixon polynomial. The second step in Dixon's method is to construct the Dixon matrix from the Dixon polynomial. To do this, one needs to rewrite the Dixon polynomial as a bilinear form. We give the following definition to formalize this.

Definition 4 *Let $\overline{V}$ be a monomial column vector in variables $\overline{X}_e$ when Δ_{X_e} is viewed as a polynomial in variables $\overline{X}_e$ and let V be a monomial row vector in X_e when Δ_{X_e} is viewed as a polynomial in variables X_e. A Dixon polynomial $\Delta_{X_e} \in \mathbb{Q}[Y, x_1][X_e, \overline{X}_e]$ can be written in bilinear form as*

$$\Delta_{X_e} = V D \overline{V} \tag{4}$$

and matrix D is the Dixon matrix with entries in $\mathbb{Q}[Y, x_1]$. The Dixon resultant $R \in \mathbb{Q}[Y, x_1]$ is the determinant of the Dixon matrix D.

Example 5 *Let $\mathcal{F} = \{x_2^2 + x_3^2 - y_3^2,\ (x_2 - y_1)^2 + x_3^2 - y_2^2,\ -x_3 y_1 + 2x_1\}$ with variables $X = \{x_1, x_2, x_3\}$ and parameters $Y = \{y_1, y_2, y_3\}$. Let $X_e = \{x_2, x_3\}$ be the variables to be eliminated and let $\overline{X}_e = \{\bar{x}_2, \bar{x}_3\}$ be the new variables corresponding to X_e. Using the identity 3, it follows that the cancellation matrix*

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

and the Dixon polynomial

$$\Delta_{X_e} = y_1 \left(-2x_2 y_1 + y_1^2 - y_2^2 + y_3^2\right) \bar{x}_2 + y_1 \left(x_2 y_1^2 - x_2\, y_2^2 + x_2\, y_3^2 - 2y_1\, y_3^2 + 4x_1 x_3\right) + y_1 \left(-2x_3 y_1 + 4x_1\right) \bar{x}_3.$$

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

$$V D \overline{V} = \begin{bmatrix} x_2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} -2y_1^2 & 0 & y_1^3 - y_1 y_2^2 + y_1 y_3^2 \\ 0 & -2y_1^2 & 4x_1 y_1 \\ y_1^3 - y_1 y_2^2 + y_1 y_3^2 & 4x_1 y_1 & -2y_1^2 y_3^2 \end{bmatrix} \begin{bmatrix} \bar{x}_2 \\ \bar{x}_3 \\ 1 \end{bmatrix}.$$

Finally, the Dixon resultant $R = \det(D)$ is

$$2y_1^4(16x_1^2 + y_1^4 - 2y_1^2 y_2^2 - 2y_1^2 y_3^2 + y_2^4 - 2y_2^2 y_3^2 + y_3^4).$$

The last step of the Dixon's method is to compute the determinant of the Dixon matrix D. Unfortunately, the Dixon matrix obtained may be rectangular thus eliminating the possibility of computing its determinant or it may be singular thus providing no information about the solutions of $\mathcal{F}$. Dixon's method was originally designed to compute Dixon resultants of $n + 1$ generic n-degree polynomials in n variables. However, for geometric problems arising in practice, the Dixon resultant is almost always zero because these systems do not have a generic degree shape [11]. These problems were addressed by Kapur, Saxena and Yang in [11]. They proved that the determinant of any maximal minor M of the Dixon matrix D is an element of the elimination ideal $I \cap \mathbb{Q}[Y][x_1]$. Thus, once a Dixon matrix D is constructed, we find any minor of D of maximal rank, and compute its determinant. Hence, the requirement for $\mathcal{F}$ to be generic n-degree in Dixon's method is no longer necessary.

Our idea to select a maximal minor M of a Dixon matrix D proceeds as follows. We pick a 62 bit prime p and choose $\beta \in \mathbb{Z}_p^{m+1}$ at random. Then we compute $B = D(\beta)$ and identify a maximal minor from B in D with high probability. This requires Gaussian elimination over $\mathbb{Z}_p$ only and in contrast to Kapur, Saxena and Yang [11] crucially avoids doing polynomial arithmetic in $\mathbb{Q}[Y, x_1]$.

3 Modified Interpolation using Kronecker substitution

Let $f = \sum_{k=1}^{t} a_k M_k(x_1, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_n]$ with $a_k \neq 0$ be a sparse polynomial. The Ben-Or/Tiwari algorithm [2] interpolates f using $2T$ 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. In this work, the Ben-Or/Tiwari algorithm is the preferred polynomial algorithm for the Cuyt and Lee's rational function interpolation algorithm [4] because it requires the fewest number of black box probes.

Let $m_i = M_i(2, 3, \cdots, p_n)$ be the monomial evaluations. The Ben-Or/Tiwari algorithm is done 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 [9].

We avoid these problems in the Cuyt and Lee sparse multivariate rational function interpolation algorithm by using Kronecker substitution to map a multivariate rational function into a univariate rational function and we evaluate at powers of a generator of $\mathbb{Z}_p^*$ instead of powers of prime $(2^j, 3^j, \cdots, p_n^j)$. To invert a Kronecker substitution, we need to know the partial degrees of a multivariate rational function $A = f/g$ for all variables involved.

3.1 Partial Degrees of $A = f/g$ in each variable

Let $A = f/g$ be a rational function in variables $y_1, \cdots, y_m$. Let $d_{f_i} \geq \deg(f, y_i)$ and $d_{g_i} \geq \deg(g, y_i)$ be partial degree bounds. Let A be viewed as

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

such that $f, g \in \mathbb{Z}[y_1, y_2, \cdots, y_{i-1}, y_{i+1}, y_{i+2}, \cdots, y_m][y_i]$. Let p be a prime and let z be a new variable. Let $\alpha = (\alpha_1, \cdots, \alpha_{i-1}, \alpha_{i+1}, \cdots, \alpha_m) \in (\mathbb{Z}_p \setminus \{0\})^{m-1}$ be selected at random. To obtain partial degree bounds for each d_{f_i} and d_{g_i}, we use enough distinct points for z selected at random from $\mathbb{Z}_p \setminus \{0\}$ and compute

$$H_i(z) := H_{f_i}/H_{g_i} = A(\alpha_1, \cdots, \alpha_{i-1}, \theta z + \beta, \alpha_{i+1}, \cdots, \alpha_n) \in \mathbb{Z}_p(z)$$

such that $d_{f_i} = \deg(H_{f_i}, z)$ and $d_{g_i} = \deg(H_{g_i}, z)$ where $\beta, \theta \in \mathbb{Z}_p$ are chosen at random. Observe that if $\mathrm{LC}(H_{f_i}, z)(\alpha) = 0$ or $\mathrm{LC}(H_{g_i}, z)(\alpha) = 0$ then the wrong partial degrees would be obtained. For example, let $A = f/g = \frac{(2-y_3)y_1^2y_2+y_1}{y_1+y_2}$ and suppose we want to determine $\deg(A, y_1)$. Let prime $p = 3137$ and let z be a new variable. Let $H_1(z) := H_{f_1}/H_{g_1} = A(\theta z+\beta, \alpha_2, \alpha_3) = \frac{(2-\alpha_3)(\theta z+\beta)^2\alpha_2+\theta z+\beta}{\theta z+\beta+\alpha_2}$. Observe that if $\alpha_3 = 2$ then $\mathrm{LC}(H_{f_1}, z)(\alpha_2, 2) = 0$ for any $\beta, \theta, \alpha_2 \in \mathbb{Z}_p$. The wrong partial degree bound of f will be returned in this case since $H_1(z) = A(\theta z + \beta, \alpha_2, 2) = \frac{\theta z+\beta}{\theta z+\alpha_2+\beta}$. Thus it is important that we pick prime $p \gg \deg f \deg g$ and α randomly.

3.2 Algorithm by Cuyt and Lee

Let $\mathbb{K}$ be a field and let $A = f/g \in \mathbb{K}(y_1, \cdots, y_m)$ be a sparse multivariate rational function with $\gcd(f, g) = 1$. Cuyt and Lee's rational function algorithm [4] reduces interpolation of sparse rational functions to sparse polynomials interpolation. The first step in their algorithm is to introduce a homogenizing variable z to form the auxiliary rational function

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

In the case when constant terms g_0 and f_0 are both zero, one has to pick $\beta \in (\mathbb{K} \setminus \{0\})^m$ and perform a basis shift to obtain auxiliary rational function $\hat{A}(y_1, \cdots, y_m, z) := A(y_1 z + \beta_1, \cdots, y_m z + \beta_m)$ so that

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

The basis shift forces the auxiliary rational function to have non-zero constant terms f_0 and g_0. This is important because their method normalizes on f_0 or g_0. That is they write

$$\hat{A}(y_1, \cdots, y_m, z) = \frac{\frac{f_0}{g_0} + \frac{f_1(y_1,\cdots,y_m)}{g_0}z + \cdots + \frac{f_{\deg f}(y_1,\cdots,y_m)}{g_0}z^{\deg f}}{1 + \frac{g_1(y_1,\cdots,y_m)}{g_0}z + \cdots + \frac{g_{\deg g}(y_1,\cdots,y_m)}{g_0}z^{\deg g}} \in \mathbb{K}[y_1 \cdots, y_m](z).$$

Thus for a black box rational function $A = f/g$, we interpolate $\hat{A}$ using univariate dense auxiliary rational functions

$$\hat{A}(\alpha^j, z) = \frac{\frac{f_0}{g_0} + \frac{f_1(\alpha^j)}{g_0}z + \cdots + \frac{f_{\deg f}(\alpha)}{g_0}z^{\deg f}}{1 + \frac{g_1(\alpha^j)}{g_0}z + \cdots + \frac{g_{\deg g}(\alpha^j)}{g_0}z^{\deg g}} \in \mathbb{K}(z)$$

for $j = 0, 1, 2, \cdots$. To interpolate $\hat{A}(\alpha^j, z)$ we use $\deg f + \deg g + 2$ black box probes on z. Since the sparsity of $A = f/g$ is destroyed by the basis shift, Cuyt and Lee adjust the coefficients of the lower degree terms in the numerator and denominator of $\hat{A}(\alpha^j, z)$ by the contributions from the higher degree terms before the coefficients interpolation step is performed. We will show how to do this in our Dixon resultant algorithm (See Subroutine Remove-Shift on page 12). Thus using an appropriate sparse polynomial interpolation algorithm such as [2, 23], the adjusted coefficients of the auxiliary rational functions produces the desired rational function $A = f/g$ that was represented by a black box.

3.3 Kronecker Substitution

Using a Kronecker substitution in Cuyt and Lee's method, we reduce the problem of interpolating a sparse multivariate rational function into a univariate rational function interpolation.

Definition 6 *Let $\mathbb{K}$ be an integral domain and let $A = f/g \in \mathbb{K}(y_1, \cdots, y_m)$. Let $r = (r_1, r_2, \cdots, r_{m-1}) \in \mathbb{Z}^{m-1}$ with $r_i > 0$. Let $K_r : \mathbb{K}(y_1, \cdots, y_m) \to \mathbb{K}(y)$ be the Kronecker substitution*

$$K_r(A) = \frac{f(y, y^{r_1}, y^{r_1 r_2}, \cdots, y^{r_1 r_2 \cdots r_{m-1}})}{g(y, y^{r_1}, y^{r_1 r_2}, \cdots, y^{r_1 r_2 \cdots r_{m-1}})}.$$

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 $K_r(A) = 0 \iff f = 0$.

Definition 7 *Let $\mathbb{K}$ be a field and let $A = f/g \in \mathbb{K}(y_1, \cdots, y_m)$ such that $\gcd(f, g) = 1$. Let z be the homogenizing variable and let $r = (r_1, \cdots, r_{m-1})$ with $r_i > d_i = \max\{(\deg f, y_i), \deg(g, y_i)\}$. Let K_r be the Kronecker substitution and let*

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

Following the presentation of auxiliary rational functions in [4], we need to guarantee the existence of a constant term in the denominator of $F(y, z)$. Thus we use a basis shift $\beta \in (\mathbb{K} \setminus \{0\})^m$ and instead define an auxiliary rational function

$$F(y, z) := \frac{f(zy + \beta_1, zy^{r_1} + \beta_2, \cdots, zy^{r_1 r_2 \cdots r_{m-1}} + \beta_m)}{g(zy + \beta_1, zy^{r_1} + \beta_2, \cdots, zy^{r_1 r_2 \cdots r_{m-1}} + \beta_m)} \in \mathbb{K}[y](z) \qquad (5)$$

with Kronecker substitution K_r.

Although the degree of the mapped rational function $K_r(A)$ is exponential in y, the degree of the auxiliary functions with Kronecker substitution $F(y, z)$ in z through which $K_r(A)$ 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 recover the exponents in y we require prime $p > \prod_{i=1}^{m} r_i$.

Example 8 *Let*

$$A = f/g = \frac{y_1 + y_2 + y_3}{y_1 + y_3} \in \mathbb{Z}_{3137}(y_1, y_2, y_3).$$

Observe that $d_i = \max\{\deg(f, y_i), deg(g, y_i)\} = 1$ for $1 \leq i \leq 2$. Let $r = (2, 2)$ where $r_i > d_i$ and let $\beta = (2, 3, 5)$ be a basis shift. Let $K_r(A) = A(y, y^2, y^4) = \frac{y^4+y^2+y}{y^4+y}$. Then $F(y, z) = A(zy+2, zy^2+3, zy^4+5) = \frac{(y^4+y^2+y)z+10}{(y^4+y)z+7} \in \mathbb{Z}_{3137}[y](z)$ is an auxiliary rational function with Kronecker substitution K_r.

3.4 Bad Evaluation Points

Definition 9 *Let p be prime and let $A = f/g \in \mathbb{Z}_p(y_1, \cdots, y_m)$ with $\gcd(f, g) = 1$. Let $\alpha \in \mathbb{Z}_p \setminus \{0\}$ and $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ with $A(\beta) \in \mathbb{Z}_p$. Let $i \geq 0$ and let*

$$F(y^i, z) := \frac{f_i(y, z)}{g_i(y, z)} = \frac{f(zy^i + \beta_1, zy^{(r_1)i} + \beta_2, \cdots, zy^{(r_1 r_2 \cdots r_{m-1})i} + \beta_m)}{g(zy^i + \beta_1, zy^{(r_1)i} + \beta_2, \cdots, zy^{(r_1 r_2 \cdots r_{m-1})i} + \beta_m)} \in \mathbb{Z}_p[y](z)$$

be the i-th auxiliary rational function with Kronecker substitution K_r. We say that $\alpha \in \mathbb{Z}_p$ is a bad evaluation point if $\deg(f_i(\alpha, z)) < \deg f$ or $\deg(g_i(\alpha, z)) < \deg g$. That is $\mathrm{LC}(f_i, z)(y = \alpha) = 0$ or $\mathrm{LC}(g_i, z)(y = \alpha) = 0$.

Example 10 *Let*

$$A = f/g = \frac{2891y_1 + y_2 + y_3}{y_2^2 + y_1 + y_3} \in \mathbb{Z}_{3137}(y_1, y_2, y_3).$$

Clearly $\gcd(f, g) = 1$. *The rational function* $A = f/g$ *does not have a constant term in the numerator or denominator. Let* $\beta = (5, 2, 3) \in \mathbb{Z}_{3137}$ *serve as the basis shift for* A. *Let* $r = (2, 3)$ *and let* $K_r(A) = A(y, y^2, y^6) = \frac{y^6+y^2+2891y}{y^6+y^4+y}$. *Then an auxiliary rational function* $F(y, z)$ *with Kronecker substitution* K_r *is*

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

If $\alpha = 3$ *is randomly picked in* $\mathbb{Z}_{3137}^*$, *then the auxiliary rational function*

$$F(3, z) = \frac{f_1(\alpha, z)}{g_1(\alpha, z)} = \frac{1108}{z^2 + 2217z + 1162} \in \mathbb{Z}_{3137}(z).$$

Thus $\deg(f_1(\alpha, z)) < 1$ *which implies that* $\alpha = 3$ *is a bad evaluation point.*

We avoid bad evaluation points with high probability in our Dixon resultant algorithm by picking any generator $\alpha \in \mathbb{Z}_p^*$ and a random shift $s \in [1, p-2]$ where p is prime and instead compute $F(\alpha^{s+j}, z)$ for $j = 0, 1, 2, \cdots$ [9].

4 The Dixon Resultant Algorithm

For the purpose of description in this paper, 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. The implementation of our algorithm handles more than one monic square-free factor. Let

$$S = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \cdots, y_m)}{g_k(y_1, \cdots, y_m)} x_1^{d_k}. \tag{6}$$

Definition 11 *Let* M *be a Dixon matrix of polynomials in* $\mathbb{Z}[x_1, y_1, \cdots, y_m]$. *For our algorithms, a black box* $\boldsymbol{BB}: \mathbb{Z}_p^{m+1} \to \mathbb{Z}_p$ *is a program that takes a prime* p *and* $\alpha \in \mathbb{Z}_p^{m+1}$ *as inputs and outputs* $\det(M(\alpha)) \mod p$.

The implication of the black box representation of $\det(M)$ is that information such as number of terms and variable degrees are unknown. The degree bounds needed are degrees $[d_0, \cdots, d_T]$ as defined in Equation 6, total degree bounds for the rational function coefficients $\frac{f_k(y_1,\cdots,y_m)}{g_k(y_1,\cdots,y_m)}$ and the maximum partial degrees $\max\left(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i))\right)$ of S with respect to each variable y_i. For lack of space, we will not present the algorithms to compute these degree bounds.

We now present our Dixon resultant algorithm labelled as algorithm Dixon-Res. It calls Algorithms SparseKron, MQRFR and Subroutines PolyInterp, RatFun, Remove-Shift, VanderSolver, BMStep. The MQRFR algorithm is the Maximal Quotient Rational Function Reconstruction algorithm in [13, page 186].

Algorithm 1: DixonRes

Input: A prime p and a black box $\mathbf{BB} : \mathbb{Z}_p^{m+1} \to \mathbb{Z}_p$
Output: A square-free factor $\overline{S} \in \mathbb{Q}[x_1, y_1, \cdots, y_m]$ of R or **FAIL**.

Compute T and $d = [d_0, \cdots, d_T]$ as defined in 6 and $\hat{D} = \deg(\det M, x_1)$.
Compute $e_k = \deg(f_k) + \deg(g_k) + 2$ for $0 \le k \le T-1$.
Let $e_{\max} = \max_{k=0}^{T-1}\{e_k\}$ and assume that $e_0 \ge e_1, \cdots \ge e_{T-1}$.
Compute $D_{y_i} = \max\left(\max_{k=0}^{T-1}(\deg(f_k, y_i), \deg(g_k, y_i))\right)$ for $1 \le i \le m-1$.
Initialize $r_i = D_{y_i} + 1$ for $1 \le i \le m$. // Prime $p > \prod_{j=1}^{m} r_i$.
Let $K_r : \mathbb{Z}_p(y_1, \cdots, y_m)[x_1] \to \mathbb{Z}_p(y)[x_1]$ be the Kronecker substitution $K_r(S)$ where S is as defined in 6 and $r = (r_1, r_2, \cdots, r_{m-1})$.
Pick a random basis shift $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$ such that $\mathbf{BB}(\beta) \in \mathbb{Z}_p$.
Pick a random shift $s \in [1, p-2]$ and any generator α for $\mathbb{Z}_p^*$.
Let z be the homogenizing variable.
Pick $\theta \in \mathbb{Z}_p^{e_{\max}}$ and $\delta \in \mathbb{Z}_p^{\hat{D}+1}$ at random.
$k \leftarrow 0$
for $i = 1, 2, \cdots$ **while** $k \le T-1$ **do**
  $\hat{Y}_i \leftarrow (\alpha^{s+i-1}, \alpha^{(s+i-1)r_1}, \cdots \alpha^{(s+i-1)(r_1 r_2 \cdots r_{m-1})})$.
  Let $Z_i = [\hat{Y}_i\theta_j + \beta \in \mathbb{Z}_p^m : 1 \le j \le e_{\max}]$ be the evaluation points.
  // Compute the monic univariate images $H_i \in \mathbb{Z}_p[x_1]$.
  $H_i \leftarrow \text{PolyInterp}(\mathbf{BB}, Z_i, \delta, d_T, e_{\max})$ // $|H_i| = e_{\max}$
  if $H_i = \mathbf{FAIL}$ return **FAIL end**
  if $i \in \{2, 4, 6, 10, 16, 26, \cdots\}$ **then**
    for $j = 1, 2, \ldots, i$ **do**
      // Compute auxiliary functions $A_j(\alpha^{s+j-1}, z) = N_j/\hat{N}_j \in \mathbb{Z}_p(z)$ with Kronecker substitution K_r. Let $A_j = A_j(\alpha^{s+j-1}, z)$.
      $A_j \leftarrow \text{RatFun}(H_j, \theta, d_k, e_k, p)$
      if $\deg(N_j, z) \ne \deg(f_k)$ *or* $\deg(\hat{N}_j, z) \ne \deg(g_k)$ **then**
        return **FAIL** // α^{s+j-1} is a bad evaluation point
      end
    end
    $F_k \leftarrow \text{BMStep}([\text{coeff}(N_j, z^{\deg(f_k)}) : 1 \le j \le i], \alpha, s, r)$.
    $G_k \leftarrow \text{BMStep}([\text{coeff}(\hat{N}_j, z^{\deg(g_k)}) : 1 \le j \le i], \alpha, s, r)$
    if $F_k \ne \mathbf{FAIL}$ *and* $G_k \ne \mathbf{FAIL}$ **then**
      $f_k \leftarrow \text{Remove-Shift}(F_k, [\hat{Y}_1, \cdots, \hat{Y}_i], [N_1, \cdots, N_i], \alpha, s, \beta, r)$
      $g_k \leftarrow \text{Remove-Shift}(G_k, [\hat{Y}_1, \cdots, \hat{Y}_i], [\hat{N}_1, \cdots, \hat{N}_i], \alpha, s, \beta, r)$
      if $f_k \ne \mathbf{FAIL}$ *and* $g_k \ne \mathbf{FAIL}$ **then**
        $k \leftarrow k+1$// We have interpolated the k-th coefficient of S.
      end
    end
  end
end
$\hat{S} \leftarrow x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1, \cdots, y_m)}{g_k(y_1, \cdots, y_m)} x_1^{d_k}$ // $\hat{S} = S \mod p$ where S is as defined in 6
$L \leftarrow \text{LCM}\{g_k \in \mathbb{Z}_p[y_1, y_2, \cdots, y_m] : 0 \le k \le T-1\}$
$\overline{M} \leftarrow \hat{S} \times L \in \mathbb{Z}_p[x_1, y_1, y_2, \cdots, y_m]$. // Clear the denominators
Apply rational number reconstruction to the coefficients of $\overline{M} \mod p$ to get $\overline{S}$
if $\overline{S} \ne \text{FAIL}$ **then**
  return $\overline{S}$
else
  $\overline{S} \leftarrow \text{SparseKron}(\mathbf{BB}, \hat{S}, \overline{M}, \{(\deg f_k, \deg g_k) : 0 \le k \le T-1\}, e_{\max}, \hat{D}, d_T)$
  if $\overline{S} \ne \text{FAIL}$ **then return** $\overline{S}$ else **return** FAIL **end**
end

Subroutine 2: Remove-Shift : The effect of the basis shift β is corrected

Input: $\overline{F} \in \mathbb{Z}_p[y_1, \cdots, y_m]$, basis shift $\beta \in (\mathbb{Z}_p \setminus \{0\})^m$, shift $s \in [1, p-2]$ and r which defines the Kronecker substitution K_r.
Input: $[\hat{Y}_j \in \mathbb{Z}_p^m : 1 \le j \le i], [N_j \in \mathbb{Z}_p[z] : 1 \le j \le i]$ and a generator α for $\mathbb{Z}_p^*$
Output: $f_k \in \mathbb{Z}_p[y_1, \cdots, y_m]$ or **FAIL**

$(\overline{A}, f_k) \leftarrow (\overline{F}, \overline{F})$
Initialize $\Gamma_j = 0$ for $j = 1, 2, \cdots, i$.
for $d = \deg(\overline{F}) - 1, \deg(\overline{F}) - 2, \cdots, 0$ **do**
  if $\overline{A} \neq 0$ **then**
    Pick $\theta \in \mathbb{Z}_p^{d+2}$ at random.
    for $j = 1, 2, \cdots, i$ **do**
      Compute polynomial evaluations :
      $\{Z_{j,t} = \overline{A}(\hat{Y}_{j,1}\theta_t + \beta_1, \cdots, \hat{Y}_{j,m}\theta_t + \beta_m) \mod p : 1 \le t \le d+2\}$.
      Interpolate $\overline{W}_j \in \mathbb{Z}_p[z]$ using points $(\theta_t, Z_{j,t} : 1 \le t \le d+2)$.
      $\Gamma_j \leftarrow \Gamma_j + \overline{W}_j$
    end
  end
  if $d \neq 0$ **then**
    Compute $P = \left[\text{coeff}(N_j, z^d) - \text{coeff}(\Gamma_j, z^d) \mod p : 1 \le j \le i\right]$.
    // The P_j's are adjusted to correct the effect of the basis shift β.//
    if $[P_j = 0 : 1 \le j \le i]$ **then**
      $\overline{A} = 0$ // There is no monomial of total degree d.
    else
      $\overline{A} \leftarrow \text{BMStep}([P_1, \cdots, P_i], \alpha, s, r)$. // $\overline{A} \in \mathbb{Z}_p[y_1, \cdots, y_m]$.
      if $\overline{A} =$ **FAIL then** return **FAIL end** // More P_j's are needed.
    end
  else
    $\overline{A} \leftarrow \text{coeff}(N_1, z^0) - \text{coeff}(\Gamma_1, z^0) \mod p$// We get the constant term.
  end
  $f_k \leftarrow f_k + \overline{A}$.
end
return f_k.

Subroutine 3: BMStep

Input: $P = [P_j \in \mathbb{Z}_p : 1 \le j \le i]$, i is even, $\alpha \in \mathbb{Z}_p$, shift $s \in [1, p-2]$ and r which defines the Kronecker substitution K_r.
Output: $\bar{F} \in \mathbb{Z}_p[y_1, y_2, \cdots, y_m]$ or **FAIL**.

Run the Berlekamp-Massey algorithm on P to obtain the polynomial $\lambda(z)$.
if $\deg(\lambda, z) = \frac{i}{2}$ **then** return **FAIL end** // More images are needed
Compute the roots of $\lambda(z)$ 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 $t = |\hat{m}|$.
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 $M = [\, y^{e_i} : i = 1, 2 \cdots, t\,]$ // These are the monomials
$F \leftarrow$VanderSolver $(\hat{m}, [P_1, \cdots P_t], s, M)$ // $F \in \mathbb{Z}_p[y]$.
$\bar{F} \leftarrow K_r^{-1}(F) \in \mathbb{Z}_p[y_1, \cdots, y_m]$.// Invert the Kronecker map K_r.
return $\bar{F}$

We use the Berlekamp-Massey algorithm [1] to find the term bounds for the leading term polynomials in $f_k(y_1, \cdots, y_m)$ and $g_k(y_1, \cdots, y_m)$ by computing the corresponding feedback polynomial $\lambda(z)$ after $i = 2, 4, 6, \cdots, \cdots$ points and we wait until $\deg(\lambda, z) < \frac{i}{2}$. The condition $\deg(\lambda, z) < \frac{i}{2}$ ensures that $\lambda(z)$ is correct with high probability. This process of determining these term bounds is done by the two calls to Subroutine BMStep in Lines 24 and 25 of Algorithm DixonRes. If Subroutine BMStep succeeds in getting the correct term bound with high probability then the output F_k or G_k is not equal to FAIL. By design it follows that the polynomials F_k or G_k are the highest degree terms in the numerator and denominator of $\frac{f_k(y_1,\cdots,y_m)}{g_k(y_1,\cdots,y_m)}$.

Next, Algorithm DixonRes sends the leading term polynomials F_k and G_k to Subroutine Remove-Shift in Lines 27 and 28 to interpolate other lower degree polynomial terms in $f_k(y_1, \cdots, y_m)$ and $g_k(y_1, \cdots, y_m)$. However, the term bound that was sufficient for interpolating the leading term polynomials might be too small for other lower degree polynomial terms in $f_k(y_1, \cdots, y_m)$ and $g_k(y_1, \cdots, y_m)$. If this happens then Subroutine Remove-Shift will output FAIL. Thus more univariate images and auxiliary rational functions are computed in Algorithm DixonRes and a new term bound is found.

We need to solve shifted transposed Vandermonde systems using Subroutine VanderSolver [9] because Algorithms DixonRes and SparseKron randomized their evaluation points with a shift $s \in [1, p-2]$. To solve the shifted transposed Vandermonde system

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

where $\hat{m}_i$ are the monomial evaluations, we use Zippel's $O(t^2)$ algorithm [23] to first solve the transposed Vandermonde system

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

which yields $c = W^{-1}B$. Notice that $V = WD$ where D is a $t \times t$ diagonal matrix with entries $D_{ii} = \hat{m}_i^s$. Thus we obtain the unknown coefficients a_i using $a_i = \hat{m}_i^{-s} c_i$ since $Va = B \Longrightarrow (WD)a = B \Longrightarrow (Da) = W^{-1}B = c \Longrightarrow a = D^{-1}c$.

Subroutine 4: RatFun : Rational function interpolation using MQRFR [13]

Input: $H = [H_j \in \mathbb{Z}_p[x_1] : 1 \leq j \leq e_{\max}], \theta \in \mathbb{Z}_p^{e_{\max}}$ and degrees d_k, e_k.

Output: $A(z) = \frac{N(z)}{\hat{N}(z)} \in \mathbb{Z}_p(z)$ such that $\hat{N}(z) = 1 + \sum_{j=1}^{\deg(\hat{N},z)} a_j z^j \in \mathbb{Z}_p[z]$.

1 $m(z) \leftarrow \prod_{i=1}^{e_k}(z - \theta_i) \in \mathbb{Z}_p[z]$.
2 Interpolate $U \in \mathbb{Z}_p[z]$ using points $(\theta_i, \text{coeff}(H_i, x_1^{d_k}) : 1 \leq i \leq e_k)$.
3 $A(z) \leftarrow \text{MQRFR}(m, U, p)$
4 **return** $A(z)$.

Algorithm 5: SparseKron

Comment: If Algorithm DixonRes does not succeed in getting the square free factor, then SparseKron gets more images using the support from Algorithm DixonRes with new primes, performs Chinese remaindering + rational number reconstruction.

Input: $\hat{S} = x_1^{d_T} + \sum_{k=0}^{T-1} \frac{f_k(y_1,y_2,\cdots,y_m)}{g_k(y_1,y_2,\cdots,y_m)} x_1^{d_k} \in \mathbb{Z}_{p_1}(y_1,\cdots,y_m)[x_1]$

Input: $\overline{M} \in \mathbb{Z}_{p_1}[x_1, y_1, \cdots, y_m]$ where prime p_1 is from Algorithm DixonRes

Input: Degree bounds $\{(\deg(f_k), \deg(g_k)) : 0 \le k \le T-1\}$ and $e_{\max}$.

Input: $\hat{D} = \deg(\det M, x_1) + 1$, and $d_T = \deg(S, x_1)$ as defined in 6.

Input: Black box $\mathbf{BB} : \mathbb{Z}_q^{m+1} \to \mathbb{Z}_q$ where $q \neq p_1$.

Output: Square-free factor $\bar{F} \in \mathbb{Q}[x_1, y_1, \cdots, y_m]$ or **FAIL**.

Let $N = \max_{k=0}^{T-1}\{\#f_k, \#g_k\}$.
$(p, P) \leftarrow (p_1, p_1)$
do
  Get a new 62 bit prime $q > P$.
  Pick $\beta, \alpha \in (\mathbb{Z}_q \setminus \{0\})^m$, $\delta \in \mathbb{Z}_q^{\hat{D}}, \theta \in \mathbb{Z}_q^{e_{\max}}$ and $s \in [1, q-2]$ at random.
  for $i = 0, 1, \cdots, N-1$ **do**
    Set $\alpha_i^* = (\alpha_1^{s+i}, \alpha_2^{s+i} \cdots, \alpha_m^{s+i})$.
    Let $Z_i = [\beta\theta_j + \alpha_i^* \in \mathbb{Z}_q^m : 1 \le j \le e_{\max}]$ be the evaluation points.
    $H_i \leftarrow \text{PolyInterp}(\mathbf{BB}, Z_i, \delta, d_T, e_{\max})$
    if $H_i = \mathbf{FAIL}$ **then**
      return **FAIL**
    end
  end
  for $k = 0, 1, \cdots, T-1$ **do**
    $\hat{m} \leftarrow [\hat{M}_i(\alpha) : 1 \le i \le n]$ where $n = \#f_k$ and $\hat{M} = supp(f_k)$.
    $\bar{m} \leftarrow [\bar{M}_i(\alpha) : 1 \le i \le \bar{n}]$ where $\bar{n} = \#g_k$ and $\bar{M} = supp(g_k)$.
    if *the monomial evaluations* $\hat{m}_i$ *or* $\bar{m}_i$ *are not distinct* **then**
      return **FAIL.**
    end
    for $j = 0, 1, 2, \cdots, N-1$ **do**
      $B_j \leftarrow \text{RatFun}(H_j, d_k, \theta, e_k, q)$ // $B_j = N_j(z)/\hat{N}_j(z) \in \mathbb{Z}_q(z)$.
      if $\deg(N_j, z) \neq \deg(f_k)$ *or* $\deg(\hat{N}_j, z) \neq \deg(g_k)$ **then**
        return **FAIL.**
      end
      $U_j(z) \leftarrow N_j(z) \times \text{LC}(\hat{N}_j, z)$
      $V_j(z) \leftarrow \hat{N}_j(z) \times \text{LC}(\hat{N}_j, z)$
      $(a_j, b_j) \leftarrow (U_j(0), V_j(0))$ // $a_j, b_j \in \mathbb{Z}_q$
      $F_k \leftarrow \text{VanderSolver}(\hat{m}, [a_1, \cdots, a_n], s, \hat{M})$.
      $G_k \leftarrow \text{VanderSolver}(\bar{m}, [b_1, \cdots, b_{\bar{n}}], s, \bar{M})$.
    end
  end
  $\hat{S} \leftarrow x_1^{d_T} + \sum_{k=0}^{T-1} \frac{F_k(y_1,y_2,\cdots,y_m)}{G_k(y_1,y_2,\cdots,y_m)} x_1^{d_k} \in \mathbb{Z}_q(y_1,\cdots,y_m)[x_1]$
  $L \leftarrow \text{LCM}\{G_k \in \mathbb{Z}_q[y_1, y_2, \cdots, y_m] : 0 \le k \le T-1\}$
  $\underline{M} \leftarrow \hat{S} \times L \in \mathbb{Z}_q[x_1, y_1, y_2, \cdots, y_m]$. // Clear the denominators.
  Solve $\hat{F} \equiv \overline{M} \mod p$ and $\hat{F} \equiv \underline{M} \mod q$ using the Chinese remainder algorithm.
  $p \leftarrow p \times q$
  Apply rational number reconstruction to the coefficients of $\hat{F}$ mod p to get $\overline{F}$
  if $\overline{F} \neq \mathbf{FAIL}$ **then** return $\overline{F}$ **end**
  $(\overline{M}, P) \leftarrow (\hat{F}, q)$
end

Subroutine 6: VanderSolver

Input: Vectors $\hat{m}, b \in \mathbb{Z}_p^t$, shift $s \in [1, p-2]$ and monomials $[M_1, \cdots, M_t]$
Output: $F \in \mathbb{Z}_p[y_1, \cdots, y_m]$

Let $V_{ij} = \hat{m}_i^{s+j-1}$ for $1 \leq i, j \leq t$.
Solve $Va = b$ for the coefficients a_i using Zippel's $O(t^2)$ algorithm [24].
return $F = \sum_{i=1}^{t} a_i M_i$

Subroutine 7: PolyInterp

Input: Black box $\mathbf{BB} : \mathbb{Z}_p^{m+1} \to \mathbb{Z}_p$.
Input: $Z = [Z_j \in \mathbb{Z}_p^m : 1 \leq j \leq e_{\max}], \delta \in \mathbb{Z}_p^{\hat{D}+1}$, degree $d_T = \deg(\det S, x_1)$.
Output: $H = [\text{monic}(H_j) \in \mathbb{Z}_p[x_1] : 1 \leq j \leq e_{\max}]$ or **FAIL**.

for $j = 1, 2, \ldots, e_{max}$ **do**
  Compute $G_j = (\mathbf{BB}(\delta_i, Z_j) : 1 \leq i \leq \hat{D}+1)$.
  Interpolate $B_j \in \mathbb{Z}_p[x_1]$ using points $(\delta_i, G_{j,i} : 1 \leq i \leq \hat{D}+1)$.
  Compute the square-free part $H_j = B_j / \gcd(B_j, B_j')$.
  if $\deg(H_j, x_1) \neq d_T$ **then** return **FAIL end**
end
return $[\text{monic}(H_1), \cdots, \text{monic}(H_{e_{\max}})]$.

5 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 $\mathbb{Z}_p$ and performing dense rational function interpolation using the MQRFR algorithm modulo a prime [13]. Thus each probe to the black box is computed using C code. Our C code supports primes up to 63 bits in length.

5.1 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 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 $\det(M(\alpha))$ mod p for many $\alpha \in \mathbb{Z}_p^n$. Often, over 80% of the time is spent computing $M(\alpha)$ mod p. The Maple command

```
> Eval(M,{seq(z[i]=alpha[i]}) mod p;
```

does what we want, however, because we want our implementation to handle many variables and fail with low probability, we want to use the largest primes the hardware can support which are 63 bit primes if we use signed 64 bit integers. Unfortunately, `Eval` uses hardware arithmetic for $p < 2^{31}$, otherwise, it uses software arithmetic which is relatively very slow. Also, `Eval` evaluates each polynomial in M independently, that is, if $M_{1,1} = 2z_1^3 z_2$ and $M_{2,2} = z_1^3 + 5z_3$

say, `Eval` computes α_1^3 twice. To speed up evaluations we have written a C program to compute $M(\alpha)$ for $p < 2^{63}$ using hardware arithmetic. In Maple, we first precompute a vector of degrees

$$D = \left[\max_{1 \le i,j \le t} \deg(M_{ij}, z_k) \ : \ 1 \le k \le n \right].$$

For each $\alpha \in \mathbb{Z}_p^n$ we call our C program from Maple with inputs M, α, D, p. To save multiplications our C program first computes power arrays

$$P_k = \left[\alpha_k^i \ : \ 0 \le i \le D_k \right] \quad \text{for} \quad 1 \le k \le n$$

then uses these P_k to evaluate $M_{i,j}(\alpha)$ for $1 \le i, j \le t$. Maple uses two data structures for polynomials, the SUM-OF-PROD data structure and the POLY data structure. POLY was added to Maple in 2013 by Monagan and Pearce [19] to speed up polynomial arithmetic. Figure 1 shows the POLY data structure for the polynomial $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$. Figure 2 shows how the same polynomial is represented in the SUM-OF-PROD data structure. All boxes in Figures 1 and 2 represent arrays. The first entry in each box is a header word; it encodes the object type and the array length.

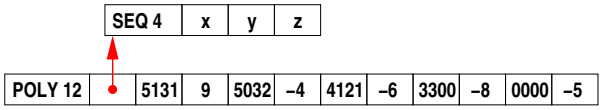

Fig. 1: Maple's POLY representation for $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$.

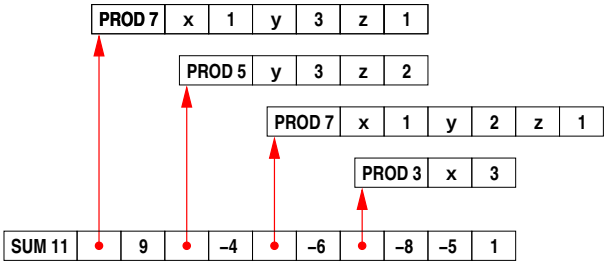

Fig. 2: Maple's SUM-OF-PROD representation for $f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$.

In POLY, if $M = z_1^{d_1} z_2^{d_2} \cdots z_n^{d_n}$ is a monomial in f, then M is encoded as the integer $d2^{nb} + \sum_{i=0}^{n-1} 2^{ib} d_i$ where $d = \sum_{i=1}^{n} d_i$ and $b = \lfloor 64/(n+1) \rfloor$. For example, the monomial xy^3z with $d = 5, b = 16, n = 3$ is encoded as the integer $5 \cdot 2^{48} + 2^{32} + 3 \cdot 2^{16} + 1$. This is depicted as 5131 in Figure 1. This encoding allows Maple to compare two monomials in the graded monomial ordering using

a single 64 bit integer comparison. Also, provided overflow does not occur, Maple can multiply two monomials using a single 64 bit integer addition.

When does Maple use POLY instead of SUM-OF-PROD? If a polynomial f has (i) all integer coefficients, (ii) more than one term, (iii) is not linear, and (iv) all monomials in f can be encoded in a 64 bit integer using B bits for d_i and $64 - nB$ bits for d, then it is encoded using POLY otherwise the SUM-OF-PROD representation is used. In a typical Dixon matrix both representations are used so we have to handle both and we need to know the details of both representations.

Also important for efficiency is how to multiply in $\mathbb{Z}_p$. We do not use the hardware division instruction which is very slow. Instead we use Roman Pearce's assembler implementation of Möller and Granlund [20] which replaces division with two multiplications and other cheap operations.

Table 2: 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 2 shows the improvement obtained using our C code for evaluating a Dixon matrix M at integer points modulo a prime for both Tot and Heron5d systems. Column `Eval` contains the timings using `Eval` command and column C-Eval is the timings for the case when our C code was used. Column Determinant is the amount of time spent computing the determinant of integer matrices modulo a prime. Column Total contains the total CPU timings using `Eval` and column New Total is the new total CPU timings for both polynomial systems when the C code for matrix evaluation was used.

5.2 Timings

We present two tables for our Dixon resultant algorithm. Table 3 contains basic information about the polynomial systems that is stored on the web at `www.cecm.sfu.ca/~mmonagan/code/DixonRes`. This web address also contain our Maple and C codes and they are freely available for use. Table 3 includes timings comparing three methods. Columns $1-4$ contain names of the polynomial systems, the number of equations in each system, the dimension of the Dixon matrix D and the rank of its maximal minor M respectively. The number of terms in the product of all the monic square-free factors in expanded form when the denominators are cleared is denoted by $\#S$, the number of terms in R labelled $\#R$ is in column 7 and column 6 labelled $t_{\max} = \max(\#f_{jk}, \#g_{jk})$. In column 8 named as DRes, we report the timings of our Dixon resultant algorithm. Column 9 contain timings of an efficient Maple implementation of the Gentleman & Johnson minor expansion method. The timings of a hybrid implementation of Zippel's sparse algorithm in Maple + C are given in column 11. All our experiments were performed on an Intel Xeon E5-2680 v2 processor using 1 core. The first prime used in our code is the 62 bit prime $p = (2^{50})(61)(67) + 1$.

The Gentleman & Johnson minor expansion algorithm uses a lot of space. To reduce space and speed it up, we first divide each row i of the Dixon matrix M by the gcd of the entries in row i. Then we permute the Dixon matrix M by putting the sparsest columns at the left of the matrix. We call this method the cleaned version of the Gentleman & Johnson method. The timings for it are presented in column 10 labelled as Cleaned.

Our DixonRes algorithm outperforms Zippel's sparse interpolation. This was expected because $\#R$ is much larger than $t_{\max}$. Another reason is because more primes are needed to recover integer coefficients in R compared to the R_j's. Our algorithm is not always faster than the Gentleman & Johnson algorithm. The evaluation cost of the Dixon matrix is still the bottleneck of our algorithm while the determinant computation takes roughly 10% of the total time.

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 2 and 3, we always compute the determinant of the smallest block after confirming that all blocks produce the same Dixon resultant. So, both $\#S$ and $\#R$ are the number of terms due to the determinant of the smallest block obtained. However for the Tot system, the 25×25 block matrix did not produce all the monic square-free factors R_j so we had to compute the determinant of the 31×31 block matrix. Details about the block structure of the Dixon matrices are provided in Table 4.

Table 3: DixonRes versus Minor Expansion and Zippel's Interpolation

System	$\#$**Eq**	n/m	$\dim D/$**Rank**	$\#S$	$t_{\max}$	$\#R$	**DRes**	**Minor**	**Cleaned**	**Zippel**
Robot-t_1	4	4/7	$(32 \times 48)/20$	450	14	6924715	7.34s	2562.6s	188.4s	$> 10^5$s
Robot-t_2	4	4/7	$(32 \times 48)/20$	13016	691	16963876	316.99s	!	2559.6s	$> 10^5$s
Robot-b_1	4	4/7	$(32 \times 48)/20$	334	85	6385205	27.78s	182.4s	15.15s	$> 10^5$s
Robot-b_2	4	4/7	$(32 \times 48)/20$	11737	624	16801877	241.61s	!	2452.8s	$> 10^5$s
Heron5d	15	14/16	$(707 \times 514)/399$	823	822	12167689	23.12s	!	!	$> 10^5$s
Flex-v1	3	3/15	$(8 \times 8)/8$	5685	2481	45773	201s	5.09s	NA	308684.76s
Flex-v2	3	3/15	$(8 \times 8)/8$	12101	2517	45773	461.4s	5.02s	NA	308684.76s
Perimeter	6	6/4	$(16 \times 16)/16$	1980	303	9698	49.97s	18.23	NA	2360.27s
Pose	4	4/8	$(13 \times 13)/12$	24068	8800	24068	461.4s	4.48s	NA	21996.25s
Pendulum	3	2/3	$(40 \times 40)/33$	4667	243	19899	45.46s	1721.50s	NA	2105.321s
Tot	4	4/5	$(85 \times 94)/56$	8930	348	52982	82.11s	!	!	17370.07s
Image3d	10	10/9	$(178 \times 152)/130$	130	84	1456	2.34s	1.04s	NA	53.68s
Heron3d	6	5/7	$(16 \times 14)/13$	23	22	90	0.411s	0.014s	NA	0.738s
Nachtwey	6	6/5	$(11 \times 18)/11$	244	106	244	7.23s	0.424s	NA	5.36s
Storti	6	5/2	$(24 \times 113)/20$	12	4	32	0.177s	229.945s	NA	0.053s

! = ran out of memory, NA= Not Attempted

In Table 4, we provide details about block sizes of each Dixon matrix M and the number of black box probes required by our Dixon resultant algorithm to successfully interpolate the R_j's. The quantity Q in Table 4 is the number of black box probes done to obtain all degree bounds needed by Algorithm DixonRes. In Table 4, the quantity p_1 is the number of probes needed to get the first image of the R_j's . If the rational number reconstruction process fails on the first image, then more primes are needed. The number of black box probes used

Table 4: Block structure and # of probes used by Algorithm DixonRes and Zippel's interpolation

System	**Block Structure**	Q	p_1	p_2	Zippel-probes
Robot-t_1	$[8, 8]$	3641	13000	-	-
Robot-t_2	$[12]$	5685	705796	-	-
Robot-b_1	$[8, 8]$	3901	91000	-	-
Robot-b_2	$[12]$	5489	529984	-	-
Heron5d	$[49, 52, 48, 50, 49, 53, 50, 48]$	307	62928	-	-
Flex-v1	$[8]$	1693	588060	-	3310871
Flex-v2	$[8]$	5017	2664948	-	3310871
Perimeter	$[16]$	1243	225828	-	230773
Pose	$[12]$	1072	525636	-	569513
Pendulum	$[17, 16]$	8971	114920	-	128322
Tot	$[31, 25]$	4261	420000	-	742099
Image3d	$[13, 14, 14, 15, 18, 19, 18, 19]$	401	12320	-	29415
Heron3d	$[6, 7]$	133	1392	-	3071
Nachtwey	$[11]$	576	39780	18020	12983
Storti	$[20]$	273	816	-	343

for the second prime is p_2. One prime is typically enough to interpolate the R_j's. Zippel-probes represents the number of probes used by Zippel's algorithm to interpolate R. Note that the block structure depends on the variable elimination order. For example, we record that the block structure for Robot-b_1 is $[8, 8]$. For a different variable elimination order, we get $[8, 4, 4]$.

6 Conclusion

We have designed and implemented a new Dixon resultant algorithm that computes the monic square-free factors of the Dixon resultant R of a parametric polynomial system using sparse interpolation tools. We have shown that there is a huge reduction in the number of terms when the monic square-free factors of R are interpolated instead of interpolating R. We have also shown that a Kronecker substitution can be used to reduce the problem of interpolating a multivariate rational function using Cuyt and Lee's method to a univariate rational function interpolation.

We implemented our algorithms in Maple and implemented several subroutines in C including the evaluation of the Dixon matrix modulo a prime. Our benchmarks showed that our algorithms is much faster than Zippel's sparse interpolation. We are currently working on the analysis of the failure probability of our new Dixon resultant algorithm.

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

3. Chtcherba, A. D., and Kapur, D.: On the Efficiency and Optimality of Dixon-Based Resultant Methods. *Proceedings of ISSAC '2002*, pp. 29–36, ACM, 2002.
4. Cuyt, A., and Lee, W.-S.: Sparse Interpolation of Multivariate Rational Functions. *J. Theoretical Comp. Sci.* ***412***, pp. 1445–1456, Elsevier, 2011.
5. Dixon, A.: On a form of the Eliminant of Two Quantics. *Proceedings of the London Mathematical Society* ***2***, (1908), pp. 468–478.
6. Dixon, A.: The eliminant of three Quantics in Two Independent Variables. *Proceedings of the London Mathematical Society* ***2***, (1909), pp. 49–69.
7. 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 (1974), pp. 543–548.
8. Gerhard, J., and Von zur Gathen, J.: *Modern Computer Algebra.* Cambridge University Press, 2013.
9. Hu, J., and Monagan, M.: A fast parallel sparse polynomial GCD algorithm. *Proceedings of ISSAC '2016*, pp. 271–278, ACM, 2016.
10. Kapur, D., and Saxena, T.: Extraneous Factors in the Dixon Resultant Formulation. *Proceedings of ISSAC '97* pp. 141–148, ACM, 1997.
11. Kapur, D., Saxena, T., and Yang, L.: Algebraic and Geometric Reasoning using Dixon Resultants. *Proceedings of ISSAC '94*, pp. 99–107, ACM, 1994.
12. Kapur, D., and Saxena, T.: Comparison of Various Multivariate Resultant formulations. *Proceedings of ISSAC '95* , pp. 187–194, ACM, 1995.
13. Khodadad, S., and Monagan, M.: Fast Rational Function Reconstruction. *Proceedings of ISSAC '2006*, pp. 184–190, ACM, 2006.
14. Lewis, R.: Dixon-EDF: The Premier Method for Solution of Parametric Polynomial Systems. *Special Sessions in Applications of Computer Algebra* (2015), Springer, pp. 237–256.
15. Lewis, R.: Resultants, Implicit Parameterizations, and Intersections of Surfaces. *International Congress on Mathematical Software* (2018), Springer, pp. 310–318.
16. Lewis, R.: Private Communication.
17. Lewis, R.: New Heuristics and Extensions of the Dixon Resultant for Solving Polynomial Systems. *Applications of Computer Algebra, Montreal, Canada* (2019), pp. 16–20.
18. Monagan, M.: Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '2004*, pp. 243–249, ACM, 2004.
19. Monagan, M., and Pearce R.: The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Communications in Computer Algebra*, **48**(4):166–186, ACM, 2014.
20. Möller, N., and Grandlund T.: Improved Division by Invariant Integers. *Transactions on Computers* **60**(2):165–175, IEEE, 2011.
21. Storti, D.: Algebraic Skeleton Transform: A symbolic computation challenge, *Submitted to Faculty Papers and Data, Mechanical Engineering, ResearchWorks Archive. http://hdl.handle.net/1773/48587*
22. Tot, J.: Private Communication.
23. Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. *Proceedings of EUROSAM '79* , pp. 216–226, (1979), Springer-Verlag, 1979.
24. Zippel, R.: Interpolating Polynomials from their Values. *Journal of Symbolic Computation* ***9*** (1990), Springer, pp. 375–403.