symmetry

Article

Constructing Dixon Matrix for Sparse Polynomial Equations Based on Hybrid and Heuristics Scheme

Guoqiang Deng [1,2], Niuniu Qi [2], Min Tang [2] and Xuefeng Duan [2,*]

1 School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China; d9801242@guet.edu.cn
2 School of Mathematics and Computing Science, Guangxi Colleges and Universities Key Laboratory of Data Analysis and Computation, Guilin University of Electronic Technology, Guilin 541004, China; 19072201023@guet.edu.cn (N.Q.); dengtangmin@guet.edu.cn (M.T.)
* Correspondence: duanxuefeng@guet.edu.cn

Abstract: Solving polynomial equations inevitably faces many severe challenges, such as easily occupying storage space and demanding prohibitively expensive computation resources. There has been considerable interest in exploiting the sparsity to improve computation efficiency, since asymmetry phenomena are prevalent in scientific and engineering fields, especially as most of the systems in real applications have sparse representations. In this paper, we propose an efficient parallel hybrid algorithm for constructing a Dixon matrix. This approach takes advantage of the asymmetry (i.e., sparsity) in variables of the system and introduces a heuristics strategy. Our method supports parallel computation and has been implemented on a multi-core system. Through time-complexity analysis and extensive benchmarks, we show our new algorithm has significantly reduced computation and memory overhead. In addition, performance evaluation via the Fermat–Torricelli point problem demonstrates its effectiveness in combinatorial geometry optimizations.

Keywords: sparsity; hybrid algorithm; successive Sylvester resultant computations; fast recursive algorithm; Dixon matrix

check for updates

Citation: Deng, G.; Qi, N.; Tang, M.; Duan, X. Constructing Dixon Matrix for Sparse Polynomial Equations Based on Hybrid and Heuristics Scheme. *Symmetry* **2022**, *14*, 1174. https://doi.org/10.3390/sym14061174

Academic Editor: Mihai Postolache

Received: 8 May 2022
Accepted: 1 June 2022
Published: 7 June 2022

1. Introduction

As a basic tool in computer algebra and a built-in function of most computer algebra systems, the notion of a resultant is widely used in mathematical theory. A resultant is not only of significance to computer algebra [1], but also plays an important role in biomedicine [2], image processing [3], geographic information [4], satellite trajectory control [5], information security [6] and other scientific and engineering fields [7,8]. The most widely used techniques for solving polynomial systems are Sylvester and Dixon resultants. For example, applying a Dixon resultant to algebraic attacks can quickly solve multivariate polynomial quadratic equations over finite fields. Although these resultant techniques have been extensively studied and improved, they still face many severe challenges regarding their easy occupation of storage space and demand for prohibitively expensive computation resources.

Specifically, a successive Sylvester elimination technique has a shortcoming over a multivariate resultant in that the performance of successive resultant computations is very sensitive to the ordering of variables. Human intervention is required to determine the most efficient ordering, and so they are not automatic methods [9]. Inappropriate choices may cause the extreme intermediate expression to swell, consequently running out of memory before accomplishment [10]. In fact, this method is very inefficient, with the number of variables increasing. Moreover, this technique performs elimination variable by variable, which is inherently sequential.

In contrast, in most cases, the Dixon method is more efficient for directly computing resultant without eliminating variables one at a time. However, it is generally known that the entries of the Dixon matrix are more complicated than the entries of other resultant matrices. As a promising scheme, the fast recursive algorithm of the Dixon matrix (FRDixon for short) [11–16] can greatly improve the computation efficiency of the Dixon matrix. Recently, its effectiveness has been analyzed in detail and proven by [16]. Nevertheless, the size of the Dixon matrix and computational complexity by FRDixon explode exponentially as the number of variables increases. If the size of the Dixon matrix is relatively large, it directly leads to difficulty computing the final resultant.

To sum up, it is difficult to overcome such difficulties mentioned above, whatever the choices of resultants. Notice that the conventional resultants resort to general methods for solving polynomial systems. From another point of view, if we can make use of the features of given systems to design customized algorithms for constructing resultant matrices or computing resultants [10], instead of general-purpose methods, it can be expected that we can obtain a targetable method to solve some intractable problems efficiently. Therefore, the main objective of this article is to make use of the sparsity of systems and advantages of prevailing resultants to design a more suitable scheme according to the features of different systems.

Symmetry/asymmetry phenomena are prevalent in scientific and engineering fields. In computer algebra, especially most problems which arise in geometry, we observe the fact that in the real polynomial systems, variables always do not stand uniformly (i.e., with asymmetry features), such as the three combinatorial geometry problems posed in [10] for Heron's formula in three and four dimensions [17], the problem of mapping of the topographical point to the standard ellipsoid [4], the Fermat–Torricelli point problem on a sphere with an Euclidean metric [18] and bifurcation points of the logistic map [17,19,20]. As we know, in the last decade, there has been considerable interest in employing sparsity to find the solutions in various fields, since most of the systems in real applications actually have sparse representations, such as finding sparse and low-rank solutions [21], parallel GCD algorithm for sparse multivariate polynomials [22], estimating the greatest common divisor (GCD) with sparse coefficients of noise-corrupted polynomials [23] and sparse multivariate interpolation [24].

However, researchers in symbolic computation mainly consider designing the methods to solve an arbitrary given polynomial system and are not aware of such sparse scenarios. In fact, similar studies on sparsity in other fields can go on in the field of solving polynomial systems. In this paper, we propose a new approach to construct a Dixon matrix for solving polynomial equations with sparsity. Through time-complexity analysis and extensive benchmarks, we show our new algorithm has significantly reduced computation and memory overhead.

1.1. Contributions

Our method is combined with Sylvester resultant and fast recursive algorithm FRDixon. It can be partitioned into two phases. In the first phase, we make use of the sparsity of systems to obtain a smaller polynomial system in fewer variables via a Sylvester resultant with fewer computational efforts. In the next step, we consider the multivariate algorithm FRDixon. Since the computational complexity of FRDixon exponentially depends on the number of variables, consequently, via Sylvester resultant elimination, the exported system with fewer variables than operated by FRDixon is more effective than single FRDixon that operates the original system directly. The idea is the genesis of our algorithm employing the Sylvester resultant and FRDixon simultaneously.

The main contributions of this paper are as follows.

1. We take advantage of the sparsity of the system and present a heuristic strategy to determine the most effective elimination ordering and remove part of the variables via Sylvester resultant.
2. We propose a method to improve the fast recursive algorithm of the Dixon matrix, which leads to reduced time and parallelism available.
3. We present a hybrid algorithm employing the methods of 1 and 2 to overcome some computation problems arising in successive Sylvester resultant computations and

FRDixon separately. Meanwhile, we apply parallel computation to speed up these two elimination processes.

4. We implement our hybrid algorithm and parallel version on Maple. Through time-complexity analysis and extensive random benchmarks, we show our new algorithm has significantly reduced computation and memory overhead in the cases of systems with sparsity. In addition, performance evaluation via the Fermat–Torricelli point problem on sphere with Euclidean metric demonstrates our algorithm's effectiveness in terms of the real combinatorial geometry optimization problems.

1.2. Related Work

It is well known that successive Sylvester resultant computations can be used to settle the problem of elimination of variables from a multivariate polynomial system to obtain a smaller polynomial system in fewer variables. Implementations of Sylvester resultants are supported in most of the computer algebra systems. With respect to so-called multivariate resultants, A.L. Dixon [25] proposed a method to simultaneously eliminate variables from multivariate systems, aiming at solving the polynomial equations system by constructing a Dixon matrix and computing its determinant.

Since then, Sylvester and Dixon approaches have been generalized and improved (see [9,11,12,16,17,20,26–34]). In [26], using the Cholesky decomposition, Zhi et al. proposed a method to compute the Sylvester resultant and reduced the time complexity to $O(r(m+n))$, where $m+n$ and r represent the size and numerical rank of the Sylvester matrix, respectively. Kapur et al. [29] extended Dixon's method for the case when Dixon matrix is singular and successfully proved many non-trivial algebraic and geometric identities. In order to improve the efficiency of computing the Dixon resultant, several methods came into existence, such as the Unknown-Order-Change [30], Fast-Matrix-Construction method [9,31], Corner-Cutting method [32], etc. Zhao et al. [11,12] extended Chionh's algorithm [31] to the general case of $n+1$ polynomial equations in n variables by n-degree Sylvester resultant, and proposed the FRDixon algorithm, which initially constructed the Dixon matrix of nine Cyclic equations. In 2017, Qin et al. [16] gave a detailed analysis of the computational complexity of Zhao's recurrence formula setting and applied parallel computation to speed up the recursive procedure [33,34]. To deal with the determinant raised in the Dixon matrix which is too large to compute or factor, some heuristic acceleration techniques were raised to accelerate computation in certain specific cases [17,20].

1.3. Organization

The rest of this article is organized as follows. Section 2 reviews the successive Sylvester resultant computations method and the FRDixon algorithm. In Section 3, a parallel hybrid algorithm which combines the Sylvester resultant and modified FRDixon is developed. Section 4 analyzes the time complexity of our proposed algorithm and conducts a series of numerical experiments. Three sets of random instances and one detailed example are presented to illustrate the application of our method. Finally, a conclusion is reported.

2. Review of Elimination Techniques

In this section, we first review the definition of a Sylvester resultant, which serves as the basis of successive Sylvester resultant computations for solving a system of polynomial equations. Then, we describe the fast recursive algorithm for construction of a Dixon matrix (FRDixon) [11].

All the discussions are stated for a general field $\mathcal{K}[X,A]$, where $X=\{x_1,\ldots,x_n\}$ denotes the set of variables and $A=\{a \,\text{is the parameter}, a\notin X\}$ denotes the set of parameters not belonging to X. Consider a system of $n+1$ polynomial equations

$$\mathrm{Sys}(f_1,\ldots,f_n)=\{f_j(x_1,\ldots,x_n)=\sum_{i_1=0}^{m_{j_1}}\cdots\sum_{i_n=0}^{m_{jn}}a_{j,i_1,\ldots,i_n}x_1^{i_1}\cdots x_n^{i_n}, j=1,\ldots,n+1\} \tag{1}$$

in n variables x_i and a number of coefficients $a_{j,i_1,\ldots,i_n} \in \mathcal{K}[A]$, where m_{j_i} is the degree of the polynomial $f_j(x_1,\ldots,x_n)$ with respect to x_i. The objective is to construct the resultant matrix of polynomial equation system (1).

2.1. Elimination via Sylvester Resultant

In this subsection, we introduce the elimination process variable by variable based on the Sylvester resultant. The classical Sylvester resultant is used to the elimination of systems of two polynomials in one variable. Consider the polynomials f and g:

$$\begin{aligned} f &= a_m x^m + a_{m-1}x^{m-1} + \cdots + a_0, \\ g &= b_l x^l + b_{l-1}x^{l-1} + \cdots + b_0, \end{aligned}$$

where m(>0) and l(>0) are the degrees of polynomials f and g in x, respectively. Recall that the Sylvester matrix of f and g in x is the matrix of the form

$$S = \begin{pmatrix} a_m & a_{m-1} & \ldots & a_0 & & & \\ & a_m & a_{m-1} & \ldots & a_0 & & \\ & & \ldots & \ldots & \ldots & \ldots & \\ & & & a_m & a_{m-1} & \ldots & a_0 \\ b_l & b_{l-1} & \ldots & b_0 & & & \\ & b_l & b_{l-1} & \ldots & b_0 & & \\ & & \ldots & \ldots & \ldots & \ldots & \\ & & & b_l & b_{l-1} & \ldots & b_0 \end{pmatrix}$$

and the resultant of f and g in x is defined as the determinant of matrix S.

Let $\mathrm{res}(f_i, f_j, x_k)$ represent the Sylvester resultant of the polynomial f_i and f_j in x_k. By computing the Sylvester resultant of f_i and other polynomials $f_2,\ldots,f_n$ with respect to x_1, respectively, we obtain the system $\mathrm{Sys}(x_2,\ldots,x_n)$ denoted by

$$\begin{cases} f_{1,2} = \mathrm{res}(f_1, f_2, x_1) \\ f_{1,3} = \mathrm{res}(f_1, f_3, x_1) \\ \qquad\quad \ldots \\ f_{1,n+1} = \mathrm{res}(f_1, f_{n+1}, x_1) \end{cases}$$

which contains n equations in $n-1$ variables $x_2,\ldots,x_n$, i.e., the variable x_1 is removed from the original system (1). This procedure may be repeated until we have determined the sequence of polynomial systems $\mathrm{Sys}(x_3,\ldots,x_n), \mathrm{Sys}(x_4,\ldots,x_n),\ldots,\mathrm{Sys}(x_n)$. Obviously, the above elimination procedure yields the final resultant $\mathrm{Sys}(x_n)$ for system (1).

Similar to the procedure above, we can eliminate $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n$ in any order by successive Sylvester resultant computations and finally have a resultant in variable $x_i (i \in \{1,\ldots,n\})$. In the worst cases, this method requires $O(n^2)$ times Sylvester resultant computations.

2.2. Fast Recursive Algorithm of the Dixon Matrix (FRDixon)

We now give the key technique of the fast recursive algorithm for constructing the Dixon matrix in [11]. To avoid the computation of polynomial division, the technique of truncated formal power series (see [31]) is employed in the FRDixon algorithm.

Let $a_i(y) (i = 0,1,\ldots,n)$ be the polynomial in y. It is obvious that $\sum_{i=0}^{n} a_i(y)x^i = 0$ when $x = y$. Therefore, the quotient $\sum_{i=0}^{n} a_i(y)x^i/(x-y)$ is a polynomial in y. We denote $1/(x-y)$ by series form $\sum_{u=1}^{\infty} x^{-u}y^{u-1}$. Hence,

$$\frac{\sum_{i=0}^{n} a_i(y)x^i}{x-y} = \sum_{i=0}^{n} a_i(y)x^i \sum_{u=1}^{i} x^{-u}y^{u-1} + \sum_{i=0}^{n} a_i(y)x^i \sum_{u=i+1}^{\infty} x^{-u}y^{u-1}, \tag{2}$$

here $\sum_{u=1}^{0} x^{-u}y^{u-1}$ is taken as zero. Since the powers of x of the second term are all negatives, the left side of Equation (2) is a polynomial if, and only if, the second term of the right side in (2) is equal to zero. Hence, the equation

$$\frac{\sum_{i=0}^{n} a_i(y)x^i}{x-y} = \sum_{i=1}^{n} a_i(y)x^i \sum_{k=1}^{i-1} x^k y^{i-1-k} \tag{3}$$

holds.

The FRDixon algorithm is based on the following ideas: employing the technique of truncated formal power series for reducing the Dixon matrix construction problem into a set of sub-Dixon matrix construction problems with fewer variables, and using the Sylvester resultant matrix and the Dixon matrix with $k-1$ ($3 \le k \le n$) variables to represent the Dixon matrix with k variables via matrix block computation. A recursive process for constructing the Dixon matrix is then proposed. For more details, one can refer to [11,12,16].

3. A Hybrid Algorithm for Constructing a Dixon Matrix

In this section, we will propose a new method to construct the Dixon matrix. Additionally, our proposed scheme is a parallel hybrid approach utilizing both the Sylvester resultant and the modified FRDixon algorithm, and making our algorithm applicable in either random polynomial systems or problems in reality.

The time complexity of our algorithm for construction of the Dixon matrix of $\mathrm{Sys}(f_1,\ldots,f_{n+1})$ defined by (1) is $O(\bar{m}^3(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3 \prod_{i=2}^{t}\tilde{m}_i^3)$ (numerical type determinant) or $O(\bar{m}!(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3 \prod_{i=2}^{t}\tilde{m}_i^3)$ (symbolic type determinant), where $\bar{m}$ is the degree bound of the polynomial, $\tilde{m}_i$ denotes the maximum degree of x_i in all polynomials $f_1,\ldots,f_{n+1}$, n denotes the number of variables and t denotes the number of variables in the polynomial system after the Sylvester resultant eliminates some variables. See Section 4.1 for details of the proof. In comparison, our method is more efficient than the existing methods for solving sparse polynomial systems.

Our method can be partitioned into two phases. In the first phase, we eliminate part of variables by Sylvester resultant from the original system, and then in the second phase, we construct the resultant matrix of the system derived from the first phase by a variant version of FRDixon.

3.1. Sylvester Elimination by Heuristic Strategy

A successive Sylvester elimination technique requires $n-1$ computations to remove a variable for a non-sparse system. If the sparse condition is satisfied, we can obtain a smaller system with the least computation costs.

In this subsection, we describe a heuristic strategy to determine which variable should be eliminated first. According to the degrees of this variable in each polynomial of the system, we can give the optimal combinational relationships of polynomials for removing it by Sylvester resultant.

If there are some variables only appearing in a few equations of system (1), the computation cost of eliminating such variables from (1) will be relatively small. In such a case, our approach to decide the elimination ordering and then remove them via Sylvester resultant in the first phase is fairly straightforward.

Let d_{j_i} represent whether the variable x_i appears in polynomial f_j. That is,

$$d_{j_i} = \begin{cases} 1 & \text{if } x_i \text{ is in variable set of } f_j \\ 0 & \text{otherwise} \end{cases}$$

For each x_i, we compute the sum of d_{j_i} for $j = 1,\ldots,n+1$ denoted by

$$e_i = \sum_{j=1}^{n+1} d_{j_i}, \quad i = 1,2,\ldots,n. \tag{4}$$

Assume x_k denotes the first variable to be eliminated from system (1). Then, for $j = 1, \dots, k-1, k+1, \dots, n$, it satisfies

$$(e_k < e_j) \text{ or } (e_k = e_j \text{ and } m_k \le m_j) \tag{5}$$

m_i means the maximum degree of x_i in all polynomials $f_1, \dots, f_{n+1}$. Once x_k is picked, we move to adjust the ordering of $f_1, \dots, f_{n+1}$. Let $\text{ord}(f_j)$ for $j = 1, 2, \dots, n+1$ be the order of f_j after rearrangement. We require that if $m_{j_k} < m_{i_k}$, then $\text{ord}(f_j) < \text{ord}(f_i)$. At this stage, the ordering of polynomials is rearranged according to the degree m_{j_k}; we denote the rearrangement system by $\text{Sys}(p_1, \dots, p_{n+1})$.

From now on, the technique of successive Sylvester resultant computations is operated on the newly adjusted system $\text{Sys}(p_1, \dots, p_{n+1})$. Assume the first l polynomials do not contain variable x_k. In such a case, we only need to eliminate x_k from the last $(n+1-l)$ polynomials $p_{l+1}, \dots, p_{n+1}$ by computing the resultants $\text{res}(p_{l+1}, p_{l+2}, x_k), \dots, \text{res}(p_{l+1}, p_{n+1}, x_k)$. Let

$$p_{l+1,j} = \text{res}(p_{l+1}, p_j, x_k), \quad l+2 \le j \le n+1.$$

Then, a new polynomial system

$$\{p_1, \dots, p_l, p_{l+1,l+2}, \dots, p_{l+1,n+1}\} \tag{6}$$

with $n-1$ variables is obtained.

We can proceed with eliminating a variable from (6) in way similar to that shown above. This approach in the first phase is illustrated in Example 1.

Example 1. *Given a system*

$$\begin{aligned} f_1 &= -3{x_3}^2 - x_1x_2x_3 + x_1^2x_2 + 9x_2x_3 \\ f_2 &= x_2{x_3}^2 - 3x_2^2 - x_3 + 5x_2 + 11 \\ f_3 &= -x_2^2x_3 - 7x_1x_2 - x_3^2 + 2x_2 - 9 \\ f_4 &= -x_2x_3^2 - x_2x_3 - x_2^2 + x_3 \end{aligned}$$

we want to eliminate a variable with the least computation cost. For each variable x_i, count the number of times of x_i appearing in $\text{Sys}(f_1, \dots, f_4)$ according to (4). We find that the minimum of $\{e_1, e_2, e_3\}$ is 2 corresponding to x_1. Therefore, x_1 is chosen to be eliminated first and $\{f_1, f_2, f_3, f_4\}$ can be rearranged by $m_{j_1} (j = 1, \dots, 4)$. The rearranged system is

$$\{p_1 = f_2, p_2 = f_4, p_3 = f_3, p_4 = f_1\}.$$

Since x_1 does not appear in p_1 and p_2, eliminating x_1 by computing $\text{res}(p_3, p_4, x_1)$, we obtain the system of 3 equations and 2 variables as

$$\begin{aligned} &p_1 = f_2 = x_2{x_3}^2 - 3x_2^2 - x_3 + 5x_2 + 11, \quad p_2 = f_4 = -x_2x_3^2 - x_2x_3 - x_2^2 + x_3, \\ &p_{3,4} = x_2^5x_3^2 + 7x_2^4x_3^2 + 2x_2^3x_3^3 - 4x_2^4x_3 + 7x_2^2x_3^3 + x_2x_3^4 + 445x_2^3x_3 - 151x_2^2x_3^2 + 4x_2^3 \ . \\ &\quad +63x_2^2x_3 + 18x_2x_3^2 - 36x_2^2 + 81x_2 \end{aligned}$$

Note that if x_3 is chosen to be eliminated first instead of x_1, this yields a much larger system:

$$\begin{aligned} f_{1,2} = \quad & \text{res}(f_1, f_2, x_3) = x_1^4x_2^4 - 3x_1^2x_2^5 - x_1^3x_2^3 - 13x_1^2x_2^4 + 54x_1x_2^5 + 50x_1^2x_2^3 - 90x_1x_2^4 - \\ & 243x_2^5 + 66x_1^2x_2^2 - 207x_1x_2^3 + 486x_2^4 - 3x_1^2x_2 + 15x_1x_2^2 + 702x_2^3 + 33x_1x_2 - \\ & 504x_2^2 + 693x_2 + 1089, \\ f_{1,3} = \quad & \text{res}(f_1, f_3, x_3) = x_1^3x_2^4 - 3x_1^2x_2^5 + x_1^4x_2^2 + 7x_1^3x_2^3 - 30x_1^2x_2^4 + 42x_1^3x_2^2 - 128x_1^2x_2^3 + \\ & 195x_1x_2^4 + 438x_1^2x_2^2 + 576x_1x_2^3 - 54x_2^4 + 54x_1^2x_2 - 414x_1x_2^2 + 81x_2^3 + 1134x_1x_2 \\ & +765x_2^2 - 324x_2 + 729, \\ f_{1,4} = \quad & \text{res}(f_1, f_4, x_3) = x_2(x_1^4x_2^3 + x_1^3x_2^3 + x_1^2x_2^4 - x_1^3x_2^2 - 3x_1^2x_2^3 - 18x_1x_2^4 + 6x_1^2x_2^2 - \\ & 3x_1x_2^3 + 81x_2^4 + 6x_1^2x_2 + 3x_1x_2^2 + 36x_2^3 - 3x_1^2 - 27x_2^2). \end{aligned}$$

Remark 1. *An attractive feature of the hybrid algorithm is that it reflects the polynomial sparsity of the system. Hence, our overall algorithm is sensitive to the sparsity of variables appearing in its original representation of a given system.*

3.2. Construction of Dixon Matrix by Improved FRDixon

At this stage, assume that we have eliminated $n-t$ variables via successive Sylvester elimination techniques by making use of a heuristic strategy. The derived system is denoted by $\mathrm{Sys}(q_1,\ldots,q_{t+1})$,

$$\mathrm{Sys}(q_1,\ldots,q_{t+1}) = \{q_j(\tilde{x}_1,\ldots,\tilde{x}_t) = \sum_{i_1=0}^{\tilde{m}_{j_1}} \cdots \sum_{i_t=0}^{\tilde{m}_{j_t}} \tilde{a}_{j,i_1,\ldots,i_t}\tilde{x}_1^{i_1}\cdots\tilde{x}_t^{i_t}, j=1,\ldots,t+1\}, \tag{7}$$

where $\tilde{m}_{j_i}$ denotes the degree of variable $\tilde{x}_i$ in $\{q_1,q_2,\ldots,q_t+1\}$. $\tilde{a}_{j,i_1,\ldots,i_t}$ denotes the coefficients of monomial.

The rest of the work constructs a Dixon matrix using an improved FRDixon algorithm. In our new version, we replace the computations of the product of Sylvester matrix $S_i(i=0,\ldots,\tilde{m}_1-1)$ and block matrix $F_j(j=0,\ldots,t\tilde{m}_1-1)$ to the sum of products of a set of matrices with smaller size, which leads to reduced time and parallelism available.

Specifically, S_i is expressed by

$$S_i = \left(\begin{array}{ccccccc}
\tilde{a}_{1,i,0,\ldots,0} & \cdots & \tilde{a}_{t+1,i,0,\ldots,0} & & & & \\
\tilde{a}_{1,i,0,\ldots,1} & \cdots & \tilde{a}_{t+1,i,0,\ldots,1} & \ddots & & & \\
\vdots & \cdots & \vdots & \ddots & \tilde{a}_{1,i,0,\ldots,0} & \cdots & \tilde{a}_{t+1,i,0,\ldots,0} \\
\tilde{a}_{1,i,\tilde{m}_2,\ldots,\tilde{m}_t} & \cdots & \tilde{a}_{t+1,i,\tilde{m}_2,\ldots,\tilde{m}_t} & \ddots & \vdots & \cdots & \vdots \\
 & & & \ddots & \tilde{a}_{1,i,\tilde{m}_2,\ldots,\tilde{m}_t-1} & \cdots & \tilde{a}_{t+1,i,\tilde{m}_2,\ldots,\tilde{m}_t-1} \\
 & & & & \tilde{a}_{1,i,\tilde{m}_2,\ldots,\tilde{m}_t} & \cdots & \tilde{a}_{t+1,i,\tilde{m}_2,\ldots,\tilde{m}_t}
\end{array}\right),$$

where $\tilde{a}_{j,i,i_2,\ldots,i_t}$ is exacted from the coefficients of polynomials $q_1,\ldots,q_{t+1}$. The order of S_i is $t!\prod_{l=2}^{t}\tilde{m}_l \times (t+1)(t-1)!\prod_{l=2}^{t}\tilde{m}_l$.

The block matrix F_j is constructed by

$$F_j = \begin{pmatrix}
DD_{1,j+1}(1,:) \\
\vdots \\
DD_{t+1,j+1}(1,:) \\
\vdots \\
DD_{1,j+1}((t-1)!\Pi_{l=2}^{t}\tilde{m}_l,:) \\
\vdots \\
DD_{t+1,j+1}((t-1)!\Pi_{l=2}^{t}\tilde{m}_l,:)
\end{pmatrix},$$

where $DD_{k,j+1}(l,:)$ denotes the l-row of $DD_{k,j+1}$ defined by (14). The order of F_j is $(t+1)(t-1)!\prod_{l=2}^{t}\tilde{m}_l \times (t-1)!\prod_{l=2}^{t}\tilde{m}_l$.

In the original version, S_i and F_j are computed, respectively. The improved algorithm considers $S_i \cdot F_j$ as the sum of products of a set of $P_{k,i} \cdot DD_{k,j+1}$; that is

$$S_i \cdot F_j = \sum_{k=1}^{t+1} P_{k,i} \cdot DD_{k,j+1}, \quad 0 \le i \le \tilde{m}_1 - 1, \quad 0 \le j \le t\tilde{m}_1 - 1, \tag{8}$$

where $P_{k,i}$ is the $t!\prod_{l=2}^{t}\tilde{m}_l \times (t-1)!\prod_{l=2}^{t}\tilde{m}_l$ matrix defined by (13).

In our theoretical analysis in Theorem 1 with an improved FRDixon algorithm, we find that the key advantages to this matrix decomposition are as follows.

- There is no need to construct matrix F_j for $j = 0, \ldots, t\tilde{m}_1 - 1$ explicitly. This is in contrast with the original FRDixon, which requires computing matrix F_j from $DD_{k,j+1}$ one by one.
- Compared to S_i, matrix $P_{k,j}$ with a smaller size can be computed independently and, consequently, has the advantage of working in parallel.
- Decomposition of (8) leads to reduced time.

Based on above analysis, we now give a description of Algorithm 1.

Algorithm 1: Improved FRDixon algorithm.

Input: $\mathrm{Sys}(q_1, \ldots, q_{t+1})$: multivariate polynomial system with $t+1$ equations and t variables $\tilde{x}_1, \ldots, \tilde{x}_t$.

Output: $\mathrm{Dixon}(q_1, \ldots, q_{t+1})$: Dixon matrix of $\mathrm{Sys}(q_1, \ldots, q_{t+1})$.

Step 1. (Decompose Dixon polynomial $\delta(q_1, \ldots, q_{t+1})$ into a set of sub-Dixon polynomials.)

By introducing the new variables $\bar{x}_1, \ldots, \bar{x}_t$ to $q_j(\tilde{x}_1, \ldots, \tilde{x}_t)$, we form the Dixon polynomial $\delta(q_1, \ldots, q_{t+1})$ defined as

$$\delta(q_1, \ldots, q_{t+1}) = \prod_{i=1}^{t} \frac{1}{\bar{x}_i - \tilde{x}_i} \begin{vmatrix} q_1(\tilde{x}_1, \ldots, \tilde{x}_t) & q_2(\tilde{x}_1, \ldots, \tilde{x}_t) & \ldots & q_{t+1}(\tilde{x}_1, \ldots, \tilde{x}_t) \\ q_1(\bar{x}_1, \ldots, \tilde{x}_t) & q_2(\bar{x}_1, \ldots, \tilde{x}_t) & \ldots & q_{t+1}(\bar{x}_1, \ldots, \tilde{x}_t) \\ \vdots & \vdots & \vdots & \vdots \\ q_1(\bar{x}_1, \ldots, \bar{x}_t) & q_2(\bar{x}_1, \ldots, \bar{x}_t) & \ldots & q_{t+1}(\bar{x}_1, \ldots, \bar{x}_t) \end{vmatrix} \tag{9}$$

By comparing the form of the left side in Equation (3), we conclude that Dixon polynomial $\delta(q_1, \ldots, q_{t+1})$ has a structure identical to (3). Hence, we can use the technique of truncated formal power series to split (9) into several sub-Dixon polynomials:

$$\begin{aligned} \delta(q_1, \ldots, q_{t+1}) = & \sum_{u_1=0}^{t\tilde{m}_1-1} \sum_{i=u_1+1}^{t\tilde{m}_1} \tilde{x}_1^{i-1-u_1} \bar{x}_1^{u_1} ((-1)^0 q_1 \sum_{i_1+\cdots+i_t=i} \delta(q_{2,i_1}, q_{3,i_2}, \ldots, q_{t+1,i_t}) \\ & +(-1)^1 q_2 \sum_{i_1+\cdots+i_t=i} \delta(q_{1,i_1}, q_{3,i_2}, \ldots, q_{t+1,i_t}) + \cdots \\ & +(-1)^t q_{t+1} \sum_{i_1+\cdots+i_t=i} \delta(q_{1,i_1}, q_{2,i_2}, \ldots, q_{t,i_t})), \end{aligned} \tag{10}$$

where $\tilde{m}_i = \max\{\tilde{m}_{j_i}, j = 1, \ldots, t+1\}$ for $i = 1, \ldots, t$. If terms of q_j are collected with respect to $\tilde{x}_k$, q_j can be written as a univariate polynomial in $\tilde{x}_k$,

$$q_j = \sum_{i=0}^{\tilde{m}_{j_k}} q_{j,i}(\tilde{x}_1, \ldots, \tilde{x}_{k-1}, \tilde{x}_{k+1}, \ldots, \tilde{x}_t) \tilde{x}_k^i,$$

then $q_{j,i}(\tilde{x}_1, \ldots, \tilde{x}_{k-1}, \tilde{x}_{k+1}, \ldots, \tilde{x}_t)$ is regarded as the coefficient polynomial from $\mathcal{K}[\tilde{x}_1, \ldots, \tilde{x}_{k-1}, \tilde{x}_{k+1}, \ldots, \tilde{x}_t]$ with respect to $\tilde{x}_k^i$. This procedure reduces the original Dixon polynomial $\delta(q_1, \ldots, q_{t+1})$ into several sub-Dixon polynomials $\delta(q_{1,i_1}, \ldots, q_{k-1,i_{k-1}}, q_{k+1,i_k}, \ldots, q_{t+1,i_t})$ with fewer variables and fewer polynomials.

Step 2. (Express the Dixon polynomial in terms of Dixon matrix.)

Deduce the recursive formula for Dixon matrix, express sub-Dixon polynomials in Dixon matrix from

$$\begin{aligned} & \sum_{i_1+\cdots+i_t=i} \delta(q_{1,i_1}, \ldots, q_{k-1,i_{k-1}}, q_{k+1,i_k}, \ldots, q_{t+1,i_t}) \\ & = [1, \cdots, \prod_{l=2}^{t} \tilde{x}_l^{(l-1)\tilde{m}_l-1}] \times \sum_{i_1+\cdots+i_t=i} \mathrm{Dixon}(q_{1,i_1}, \ldots, q_{k-1,i_{k-1}}, q_{k+1,i_k}, \ldots, q_{t+1,i_t}) \times \\ & \quad [1, \cdots, \prod_{l=2}^{t} \tilde{x}_l^{(t-l+1)\tilde{m}_l-1}]^T, \end{aligned} \tag{11}$$

Algorithm 1: *Cont.*

for $k = 1, \dots, t+1$ and $i = 1, \dots, t\tilde{m}_1$. Then, substitute (11) into (10); we can obtain the Dixon matrix representation of (10) as follows,

$$\begin{aligned}
&\delta(q_1, \dots, q_{t+1}) \\
&= \sum_{u_1=0}^{t\tilde{m}_1-1} \sum_{i=u_1+1}^{t\tilde{m}_1} \tilde{x}_1^{i-1-u_1} \bar{x}_1^{u_1} (q_1 \cdot [1, \cdots, \prod_{l=2}^{t} \tilde{x}_l^{(l-1)\tilde{m}_l-1}] \cdot \sum_{i_1+\cdots+i_t=i} \mathrm{Dixon}(q_{2,i_1}, \dots, q_{t+1,i_t}) \\
&+ \cdots + (-1)^t q_{t+1} \cdot [1, \cdots, \prod_{l=2}^{t} \tilde{x}_l^{(l-1)\tilde{m}_l-1}] \cdot \sum_{i_1+\cdots+i_t=i} \mathrm{Dixon}(q_{1,i_1}, \dots, q_{t,i_t})) \\
&\times [1, \cdots, \prod_{l=2}^{t} \bar{x}_l^{(t-l+1)\tilde{m}_l-1}]^T.
\end{aligned} \tag{12}$$

Step 3. (Construct the matrix $P_{k,i}$.)
Extract the coefficients of $q_1, \dots, q_{t+1}$ to construct the matrix

$$P_{k,i} = \begin{pmatrix} \tilde{a}_{k,i,0,\dots,0} & & & \\ \tilde{a}_{k,i,0,\dots,1} & \ddots & & \\ \vdots & \ddots & \ddots & \tilde{a}_{k,i,0,\dots,0} \\ \tilde{a}_{k,i,\tilde{m}_2,\dots,\tilde{m}_t} & \ddots & \ddots & \vdots \\ & & \ddots & \tilde{a}_{k,i,\tilde{m}_2,\dots,\tilde{m}_t-1} \\ & & & \tilde{a}_{k,i,\tilde{m}_2,\dots,\tilde{m}_t} \end{pmatrix}, \tag{13}$$

where $k = 1, \dots, t+1$ and $i = 0, \dots, \tilde{m}_1 - 1$. From (13), it is easy to see that $P_{k,i}$ is a block matrix. The order of $P_{k,i}$ is $t! \prod_{l=2}^{t} \tilde{m}_l \times (t-1)! \prod_{l=2}^{t} \tilde{m}_l$.
Step 4. (Construct the matrix $DD_{k,j+1}$.)
Compute the sum of sub-Dixon matrices corresponding to sub-Dixon polynomials in (12), denoted by

$$DD_{k,j+1} = \sum_{i_1+\cdots+i_t=j+1} \mathrm{Dixon}(q_{1,i_1}, \dots, q_{k-1,i_{k-1}}, q_{k+1,i_k}, \dots, q_{t+1,i_t}). \tag{14}$$

where $k = 1, \dots, t+1$ and $j = 0, \dots, t\tilde{m}_1 - 1$. The order of $DD_{k,j+1}$ is $(t-1)! \prod_{l=2}^{t} \tilde{m}_l \times (t-1)! \prod_{l=2}^{t} \tilde{m}_l$.
Step 5. (Compute $S_i \cdot F_j$.)
From (13) and (14), compute the product of $S_i \cdot F_j$ by (8).
Step 6. (Construct $\mathrm{Dixon}(q_1, \dots, q_{t+1})$ using $S_i \cdot F_j$.)
From the evaluations of $S_i \cdot F_j$ for $0 \le i \le \tilde{m}_1 - 1$ and $0 \le j \le t\tilde{m}_1 - 1$, construct Dixon matrix,

$$\begin{aligned}
\mathrm{Dixon}(q_1, \dots, q_{t+1}) &= \begin{pmatrix} D_{0,0} & \cdots & D_{0,t\tilde{m}_1-1} \\ \vdots & \vdots & \vdots \\ D_{\tilde{m}_1-1,0} & \cdots & D_{\tilde{m}_1-1,t\tilde{m}_1-1} \end{pmatrix} \\
&= \begin{pmatrix} S_0 & & \\ \vdots & \ddots & \\ S_{\tilde{m}_1-1} & \cdots & S_0 \end{pmatrix} \cdot \begin{pmatrix} F_0 & \cdots & F_{(t-1)\tilde{m}_1} & \cdots & F_{t\tilde{m}_1-1} \\ \vdots & \iddots & \iddots & \iddots & \\ F_{\tilde{m}_1-1} & \cdots & F_{t\tilde{m}_1-1} & & \end{pmatrix},
\end{aligned}$$

where

$$D_{i,j} = \sum_{k=0}^{\min\{i, t\tilde{m}_1-1-j\}} S_{i-k} \cdot F_{j+k} \tag{15}$$

is a block matrix of order $t! \prod_{l=2}^{t} \tilde{m}_l \times (t-1)! \prod_{l=2}^{t} \tilde{m}_l$.

3.3. The Parallel Hybrid Algorithm

In Sections 3.1 and 3.2, we describe the two phases involved in our hybrid algorithm. Now, the overall algorithm is presented.

Remark 2. *The Algorithm 2 presented corresponds to our sequential implementation. Further parallel is available. In particular,*

- *In step 2, once x_k is determined to be eliminated, we simultaneously have at our disposal the computations $p_{l+1,l+2}, \dots, p_{l+1,n+1}$. Hence, $\mathrm{res}(p_{l+1}, p_j, x_k)$ can be obtained in parallel.*
- *In step 3 and step 4, $P_{k,i}$ and $DD_{k,j+1}$ can each be obtained independently. Hence, the computations of $P_{k,i}$ and $DD_{k,j+1}$ can be carried out in parallel.*
- *In step 5, once $P_{k,i}$ and $DD_{k,j+1}$ are known to us, we can compute $D_{i,j}$ immediately. Hence, the initialization of $D_{i,j}$ can be performed in parallel.*
- *In step 6, recursive operation is carried out on each anti-diagonal line as $\mathrm{Dixon}(q_1, \dots, q_{t+1})$ can also be performed in parallel.*

Algorithm 2: Hybrid algorithm.

```
Input:  $\mathrm{Sys}(f_1, \dots, f_{n+1})$: multivariate polynomial system with $n+1$
        equations and $n$ variables $x_1, \dots, x_n$ over $\mathcal{K}[X, A]$.
Output: $\mathrm{Dixon}(f_1, \dots, f_{n+1})$: Dixon matrix of $\mathrm{Sys}(f_1, \dots, f_{n+1})$.
Step 1. (Select variable $x_k$ to be eliminated from $\mathrm{Sys}(f_1, \dots, f_{n+1})$ by applying
heuristic scheme.)
Select the variable $x_k$ to be eliminated according to (5). Then, rearrange the
  polynomial $f_j$ in terms of the degrees of $f_j (j = 1, \dots, n+1)$ in $x_k$. Denote the
  rearranged polynomial system as $\mathrm{Sys}(p_1, \dots, p_{n+1})$.
Step 2. (Eliminate $x_k$ from $\mathrm{Sys}(p_1, \dots, p_{n+1})$.)
Assume the polynomials $p_1, \dots, p_l$ do not contain variable $x_k$. Eliminate $x_k$ from
  $\{p_{l+1}, \dots, p_{n+1}\}$ by Sylvester resultant:

        $p_{l+1,j} = \mathrm{res}(p_{l+1}, p_j, x_k), j = l+2, \dots, n+1$

According to the size and feature of given system (1), the process of selection and
  elimination can be continued until $n - t$ variables are removed from
  $\mathrm{Sys}(f_1, \dots, f_{n+1})$. Assume the derived system is denoted by $\mathrm{Sys}(q_1, \dots, q_{t+1})$.
Step 3. (Construct the matrix $P_{k,i}$.)
for $i = 0, \dots, \tilde{m}_1 - 1$ do
  for $k = 1, \dots, t+1$ do
    Construct the $P_{k,i}$ by (13) in Algorithm 1.
  end $k$ for;
end $i$ for ;
Step 4. (Construct the matrix $DD_{k,j+1}$.)
for $j = 0, \dots, t\tilde{m}_1 - 1$ do
  for $k = 1, \dots, t+1$ do
    Construct the $DD_{k,j+1}$ by (14) recursively in Algorithm 1.
  end $k$ for;
end $j$ for ;
Step 5. (Initialize the elements $D_{i,j}$ of $\mathrm{Dixon}(q_1, \dots, q_{t+1})$.)
From step 3 and step 4, compute the product of $S_i \cdot F_j$ by (8) and then initialize the
  elements $D_{i,j} = S_i \cdot F_j$ of $\mathrm{Dixon}(q_1, \dots, q_{t+1})$.
Step 6. (Construct the $\mathrm{Dixon}(q_1, \dots, q_{t+1})$.)
Observing (15), we find that the following relationship holds,

        $D_{i,j} = D_{i-1,j+1} + S_i \cdot F_j, \quad 1 \le i \le \tilde{m}_1 - 1, \quad 0 \le j \le t\tilde{m}_1 - 2,$

  then recursive operation is carried out on each anti-diagonal line. Hence,
  constructing the $\mathrm{Dixon}(q_1, \dots, q_{t+1})$ as follows
```

Algorithm 2: *Cont.*

$$\mathrm{Dixon}(q_1,\ldots,q_{t+1}) = \begin{pmatrix} S_0\cdot F_0 & \cdots & S_0\cdot F_{\tilde{m}_1-1} & S_0\cdot F_{\tilde{m}_1} & \cdots & S_0\cdot F_{t\tilde{m}_1-1} \\ \swarrow & \iddots & \swarrow & \swarrow & \iddots & S_1\cdot F_{t\tilde{m}_1-1} \\ \iddots & \iddots & \iddots & \iddots & \iddots & \vdots \\ \swarrow & \iddots & \swarrow & \swarrow & \iddots & S_{\tilde{m}_1-2}\cdot F_{t\tilde{m}_1-1} \\ \swarrow & \iddots & \swarrow & \swarrow & \iddots & S_{\tilde{m}_1-1}\cdot F_{t\tilde{m}_1-1} \end{pmatrix} \tag{16}$$

can avoid a mount of repeated computations.

Our method exploits the sparsity in variables of the system and introduces heuristics strategy. Parts of the variables are chosen to be eliminated from the original system via successive Sylvester resultant computations. Next, the exported system with fewer variables is operated by improved FRDixon. Meanwhile, these two elimination processes can be paralleled.

4. Analysis and Evaluation

We first analyze the time complexity of the hybrid algorithm in Section 4.1, and then evaluate the performance of our algorithm. In Section 4.2, we discuss implementations in random instances. Section 4.3 reports on our approach in a real problem. The effectiveness and practicality of our method are illustrated in these examples.

4.1. Time Complexity Analysis

We now give the sequential complexity of the hybrid algorithm in terms of the number of arithmetic operations, and use big-O notation to simplify expressions and asymptotically estimate the number of operations algorithm used as the input grows.

Theorem 1. *The time complexity of the hybrid algorithm (Algorithm 2) for construction of Dixon matrix of* $\mathrm{Sys}(f_1,\ldots,f_{n+1})$ *defined by (1) is* $O(\bar{m}^3(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3\prod_{i=2}^{t}\tilde{m}_i^3)$ *(numerical type determinant) or* $O(\bar{m}!(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3\prod_{i=2}^{t}\tilde{m}_i^3)$ *(symbolic type determinant).*

Proof. Similar to the framework of a hybrid algorithm, we partition two phases to analyze the sequential complexity of our new method.

In the first phase, we consider successive Sylvester resultant computations. The most expensive component of this part is to compute a set of Sylvester resultants. It can be shown that evaluation of a $m\times m$ numerical determinant using row operations requires about $2m^3/3$ arithmetic operations ([35]). If symbolic determinant needs to be computed, a cofactor expansion requires over $m!$ multiplications in general. Suppose that $n-t$ variables are eliminated in the whole successive Sylvester resultant computations. When the ith elimination process is performed, we let $l_i(\leq n+1-i)$ for $1\leq i\leq n-t$ denote the number of Sylvester resultant computations and $\bar{m}$ denote the degree bound of polynomials.

In terms of the complexity of determinant computation, the first phase requires $\frac{2}{3}\bar{m}^3\sum_{i=1}^{n-t}l_i$ arithmetic operations for numerical determinant or $\frac{\bar{m}!}{2}\sum_{i=1}^{n-t}l_i$ arithmetic operations for the symbolic determinant. Hence, the complexity of this part is $O(\bar{m}^3(n^2-t^2+n+t))$ or $O(\bar{m}!(n^2-t^2+n+t))$ using big-O notation.

In the second phase, we consider improved FRDixon for system (7). The main cost is calculation of (8). Each of these $P_{k,i}\cdot DD_{k,j+1}$'s requires

$$[((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^3+(t-1)((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^2]$$

multiplications and

$$[((t-1)!\prod_{l=2}^{t}\tilde{m}_i)^3-(t-2)((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^2]+(t-2)(t-1)!\prod_{l=2}^{t}\tilde{m}_l)$$

additions. The $t\tilde{m}_1^2$ calls to compute $S_i \cdot F_j$ cost

$$(t^2-1)t\tilde{m}_1^2[((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^3+(t-1)((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^2]$$

multiplications and

$$(t^2-1)t\tilde{m}_1^2[\quad ((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^3-(t-2)((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^2+(t-2)(t-1)!\prod_{l=2}^{t}\tilde{m}_l+ (t!\prod_{l=2}^{t}\tilde{m}_l)^2]$$

additions. The Dixon$(q_1,\ldots,q_{t+1})$ can be constructed in $(t\tilde{m}_1-1)^2$ additions using the method given in (16). Hence, the complexity of improved FRDixon is

$$(t^2-1)t\tilde{m}_1^2[((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^3+(t-1)((t-1)!\prod_{l=2}^{t}\tilde{m}_l)^2]$$

multiplications and

$$(t^2-1)t\tilde{m}_1^2[\tilde{T}^3-(t-2)\tilde{T}^2+(t-2)\tilde{T}+(t!\prod_{l=2}^{t}\tilde{m}_l)^2]+(t\tilde{m}_1-1)^2$$

additions, where $\tilde{T}=(t-1)!\prod_{l=2}^{t}\tilde{m}_l$. If we use big-$O$ notation, the complexity of improved FRDixon is $O(\tilde{m}_1^2 t!^3\prod_{l=2}^{t}\tilde{m}_l^3)$.

Hence, the complexity of hybrid algorithm is $O(\bar{m}^3(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3\prod_{i=2}^{t}\tilde{m}_i^3)$ for numerical type or $O(\bar{m}!(n^2-t^2+n+t)+\tilde{m}_1^2 t!^3\prod_{i=2}^{t}\tilde{m}_i^3)$ for symbolic type. □

4.2. *Random Systems*

To compare the performance of our hybrid algorithm, successive Sylvester resultant elimination method and FRDixon ([11,16]), we have implemented these algorithms on three benchmark sets with different sizes. All timings reported are in CPU seconds and were obtained using Maple 18 on Intel Core i5 3470 @ 3.20 GHz running Windows 10, involving basic operations such as matrix multiplication and determinant calculations.

4.2.1. Timings

Each polynomial of systems S1–S30 in Tables 1–3 is generated at random using the Maple command 'randpoly'. To guarantee the sparsity of the system, remove one variable randomly from each polynomial in every system of S1–S30. One hundred instances are contained in each system and the average running time is reported. The timings for columns 4, 5 and 6 of Tables 1–3 are for the successive Sylvester resultant elimination method (SylRes for short), FRDixon and hybrid algorithm (Hybrid for short), respectively. To better assess the parallel implementation of our algorithm, we report timings and speed-ups for four cores listed in column 7 in Tables 1–3. '—' indicates that the program went on for more than 2000s, or ran out of space.

***Benchmark* #1**

This set of benchmarks consists of 10 groups of systems. Every system of S1–S10 contains five polynomial equations in four variables.

The data in Table 1 show that for systems S1–S10, our new algorithm has a better performance compared to FRDixon. For systems S1–S4, SylRes has fewer timings than the hybrid algorithm. As terms and degrees increase, systems become more complex. Our hybrid algorithm is superior to SylRes. Considering test instances S9 and S10, we tried successive resultant computations using the existing implementations of Sylvester's resultant in Maple for various variable orderings, but most of the time, the computations run out of memory. We also tried to compute Dixon matrix by FRDixon. The program ran for up to 2000s in these examples.

Table 1. Benchmark #1: variables = 4.

System	Term	Degree	Average Time (s)			
			SylRes	**FRDixon**	**Hybrid**	**Hybrid (in Parallel)**
S1	4	2	0.43	11.575	2.283	0.771
S2	5	2	0.93	19.274	2.673	0.879
S3	6	3	11.40	216.115	40.398	11.481
S4	7	3	36.02	234.975	49.641	13.264
S5	8	4	234.51	558.974	94.813	24.813
S6	9	4	265.42	568.757	105.221	28.952
S7	10	5	1469.08	906.224	287.381	73.475
S8	11	5	1564.15	921.901	297.517	76.837
S9	12	6	—	—	764.242	195.073
S10	13	6	—	—	773.361	197.019

Benchmark #2

This set of systems differs from the first benchmark in that the degree of each polynomial is set to be four in the second set. The number of terms and variables varies from small to large.

In the experimentation of benchmark #2 in Table 2, we found that our new hybrid algorithm is faster than FRDixon for computing the Dixon matrix. Notice that when the number of variables increases up to 7 and the terms of polynomials reach 12 or more, the FRDixon was not successfully terminated after 2000s. There were similar results for benchmark #1. Successive Sylvester resultant computations cost less than our method in simple systems S11 and S12. However, for complicated systems, the successive technique shows its inefficiency.

Table 2. Benchmark #2: degrees = 4.

System	Term	Variable	Average Time (s)			
			SylRes	**FRDixon**	**Hybrid**	**Hybrid (in Parallel)**
S11	4	3	0.23	6.391	0.507	0.187
S12	5	3	0.21	5.330	0.847	0.223
S13	6	4	203.74	524.672	89.325	23.492
S14	7	4	219.53	533.013	98.321	26.398
S15	8	5	564.12	760.180	279.945	70.447
S16	9	5	596.09	781.112	299.864	76.106
S17	10	6	1759.25	1265.803	594.381	151.093
S18	11	6	—	1301.021	617.829	155.479
S19	12	7	—	—	1359.986	346.983
S20	13	7	—	—	1505.042	382.271

Benchmark #3

This set of benchmarks consists of 10 groups of systems of terms = 6 polynomial equations with varying numbers of variables and degrees. See Table 3. As analysis of the complexity of the hybrid algorithm, the number of arithmetic operations is mainly affected by variables and degrees. With the increase in these two parameters, three algorithms need more computation time for completion. Facing intractable systems S27–S30, SylRes and FRDixon are both powerless.

In all experiments listed in Tables 1–3, our hybrid algorithm always takes advantage of the sparsity of a given system and combines successive Sylvester resultant computations with improved FRDixon. The average running time is never more than the original FRDixon. Except for some "simple" instances, the new algorithm has a better performance compared to successive Sylvester resultant computations.

Table 3. Benchmark #3: terms = 6.

System	Degree	Variable	Average Time (s)			
			SylRes	**FRDixon**	**Hybrid**	**Hybrid (in Parallel)**
S21	2	3	0.02	2.640	0.207	0.072
S22	3	3	0.12	5.639	0.440	0.134
S23	3	4	11.01	223.527	34.568	9.007
S24	4	4	125.03	516.969	87.380	22.043
S25	4	5	532.52	727.617	271.239	69.971
S26	5	5	844.64	1106.289	478.947	122.307
S27	5	6	—	—	851.086	215.152
S28	6	6	—	—	987.602	249.005
S29	6	7	—	—	1823.056	460.764
S30	7	7	—	—	1909.443	483.125

4.2.2. Matrix Dimension

According to the definition of the resultant (determinant of resultant matrix), one can notice that the dimension of resultant matrix is the key factor affecting the computational complexity on the target resultant. In order to compare the matrix dimension generated by FRDixon and the hybrid algorithm, we record the matrix sizes of systems S1–S30, as shown in Tables 4–6.

Table 4. Matrix dimensions corresponding to systems in Table 1.

Algorithm	Matrix Dimension				
	S1	**S2**	**S3**	**S4**	**S5**
Hybrid	48×48	48×48	480×480	720×720	1008×1008
FRDixon	192×192	192×192	864×864	864×864	1944×1944
	S6	**S7**	**S8**	**S9**	**S10**
Hybrid	1440×1440	1260×1260	1296×1296	2592×2592	3402×3402
FRDixon	2304×2304	2800×2800	2800×2800	—	—

Table 5. Matrix dimensions corresponding to systems in Table 2.

Algorithm	Matrix Dimension				
	S11	**S12**	**S13**	**S14**	**S15**
Hybrid	72×72	96×96	864×864	1008×1008	1296×1296
FRDixon	144×144	144×144	1944×1944	1944×1944	5760×5760
	S16	**S17**	**S18**	**S19**	**S20**
Hybrid	1944×1944	2970×2970	3168×3168	6336×6336	7128×7128
FRDixon	5760×5760	8640×8640	8640×8640	—	—

Table 6. Matrix dimensions corresponding to systems in Table 3.

Algorithm	Matrix Dimension				
	S21	**S22**	**S23**	**S24**	**S25**
Hybrid	12×12	18×18	480×480	840×840	1440×1440
FRDixon	24×24	48×48	864×864	1944×1944	5760×5760
	S26	**S27**	**S28**	**S29**	**S30**
Hybrid	2160×2160	5760×5760	7128×7128	8640×8640	9072×9072
FRDixon	6480×6480	—	—	—	—

From Tables 4–6, we can see that the size of the Dixon matrix produced by hybrid algorithm is much smaller than FRDixon for any system.

4.3. Real Problems

In order to measure the comprehensive performance of the hybrid algorithm in the real problems, we implement our new algorithm on an open optimization problem in combinatorial geometry.

Given a spherical triangle $\triangle ABC$ whose length of sides are a, b and c, respectively, the problem is to find a point P on the sphere such that the sum of distances L between the point P and the vertexes of $\triangle ABC$ reaches the minimum.

As illustrated in Figure 1, let $AB = c$, $BC = a$, $AC = b$, $PA = u$, $PB = v$, $PC = w$ and $L = u + v + w$. All distances are measured by an Euclidean metric. If we can find the relationship between a, b, c and L, the original minimum problem is solved.

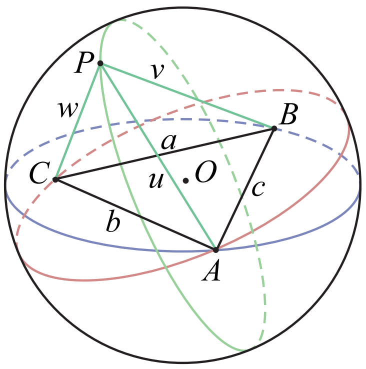

Figure 1. A triangle ABC on the sphere and their Fermat–Torricelli point P.

By applying the Lemma 42.1 in [36] and the compactness of the sphere, we can prove that a point P on sphere is such that the Cayley–Menger determinant equals to zero. That is, the distances between the center point O of the sphere and A, B, C, P should satisfy

$$V(a,b,c,u,v,w) = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & u^2 & v^2 & w^2 & 1 \\ 1 & u^2 & 0 & c^2 & b^2 & 1 \\ 1 & v^2 & c^2 & 0 & a^2 & 1 \\ 1 & w^2 & b^2 & a^2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{vmatrix} = 0. \tag{17}$$

Now, we transform the relationship between a, b, c and L into an optimization problem of the following form

$$\begin{array}{l} \min \ L = u + v + w \\ s.t. \ \ V(a,b,c,u,v,w) = 0. \end{array} \tag{18}$$

Consequently, the Fermat–Torricelli point P on the sphere is such that the polynomial system

$$\begin{cases} L - u - v - w = 0 \\ \frac{\partial G}{\partial \lambda} = 0, \quad \frac{\partial G}{\partial u} = 0, \quad \frac{\partial G}{\partial v} = 0, \quad \frac{\partial G}{\partial w} = 0. \end{cases} \tag{19}$$

where

$$G{=}u + v + w + \lambda \cdot V(a,b,c,u,v,w). \tag{20}$$

From (17)–(20), we obtain the polynomial system

$$\begin{cases} f_1(u,v,w,\lambda) = L - u - v - w \\ f_2(u,v,w,\lambda) = -a^4u^4 + 2a^2b^2u^2v^2 + \cdots - 4c^2v^2w^2 + 4c^2w^4 \quad \text{(28 terms)} \\ f_3(u,v,w,\lambda) = \left(-4a^4u^3 + 4a^2b^2uv^2 + \cdots + 8c^2uv^2 - 8c^2uw^2\right)\lambda + 1 \quad \text{(14 terms)} \\ f_4(u,v,w,\lambda) = \left(4a^2b^2u^2v - 4b^4v^3 + \cdots + 8c^2u^2v - 8c^2vw^2\right)\lambda + 1 \quad \text{(14 terms)} \\ f_5(u,v,w,\lambda) = \left(4a^2c^2u^2w + 4b^2c^2v^2w + \cdots - 8c^2v^2w + 16c^2w^3\right)\lambda + 1 \quad \text{(14 terms)}. \end{cases} \tag{21}$$

It is easy to know that this geometry problem is equivalent to solving the polynomial system (21) with five equations and four variables.

We first report the result that (21) solved by successive Sylvester resultant computation techniques. The order of elimination is λ, u, v and w. Starting with computing $\mathrm{res}(f_3, f_5, \lambda)$ and $\mathrm{res}(f_4, f_5, \lambda)$, the variable λ is eliminated from (21),

$$\begin{cases} f_1(u,v,w) = L - u - v - w \\ f_2(u,v,w) = -a^4u^4 + 2a^2b^2u^2v^2 + \cdots - 4c^2v^2w^2 + 4c^2w^4 \quad (28 \text{ terms }) \\ f_{3,5}(u,v,w) = -4a^4u^3 - 4a^2b^2u^2v + \cdots - 8c^2uw^2 + 8c^2vw^2 \quad (26 \text{ terms }) \\ f_{4,5}(u,v,w) = -4a^4u^3 + 4a^2b^2uv^2 + \cdots + 8c^2v^2w - 16c^2w^3 \quad (26 \text{ terms }). \end{cases} \tag{22}$$

This is followed by computing

$$p_1 = \mathrm{res}(f_1, f_2, u), \quad p_2 = \mathrm{res}(f_1, f_{3,5}, u), \quad p_3 = \mathrm{res}(f_1, f_{4,5}, u); \tag{23}$$

and

$$p_4 = \mathrm{res}(p_1, p_2, v), \quad p_5 = \mathrm{res}(p_1, p_3, v);$$

to eliminate u and v, respectively. Finally, w is eliminated by computing

$$p_6 = \mathrm{res}(p_4, p_5, w).$$

It was found that the elimination process of $\mathrm{res}(p_4, p_5, w)$ could not be completed due to the memory overflows after 7961.3 s.

Now, we discuss the trace of our algorithm on this system. First, λ was eliminated by computing $\mathrm{res}(f_3, f_5, \lambda)$ and $\mathrm{res}(f_4, f_5, \lambda)$. This took 0.85 s. This was followed by the construction of a Dixon matrix of system (22), which turned out to be 384×384, and this took 40.127 s. The total computational cost took 40.977 s. Another scheme was to eliminate two variables λ and u by proceeding with (22) and (23). This took 1.48 s. Then, FRDixon was used to compute the target Dixon matrix. After 1.626 s, a 32×32 Dixon matrix was obtained. The total computations took 3.106 s. The second scheme, in contrast, works better than the first.

Lastly, let us look at the FRDixon. It took 895.013 s to compute the Dixon matrix of 1536×1536. The results of these methods are summarized in Table 7.

Table 7. Comparing the timings and the dimensions of Fermat–Torricelli problem on sphere with Euclidean metric.

	SylRes	FRDixon	Hybrid	
			Scheme 1	Scheme 2
Timings	>7961.3	895.013	40.977	3.106
Dimension of matrix	—	1536×1536	384×384	32×32

5. Conclusions

In this paper, we proposed a hybrid algorithm which combines the Sylvester resultant elimination based on heuristic strategy and the improved fast recursive Dixon matrix construction algorithm. Our hybrid algorithm can construct a Dixon matrix efficiently, and the dimension is smaller than other existing algorithms. Moreover, we can achieve reasonable robustness through randomization cases over different sizes of systems and a real problem.

To conclude, we point out that the polynomial systems are sparse in many applications. Our hybrid method seems to be very suitable for such a scenario. We have demonstrated some prospects of our approach through a great example. These preliminary findings are very encouraging and suggest that further studies are needed to examine methods based on hybrid algorithms. We are therefore planning to explore different and specific applications.

For instance, we are planning to apply a Dixon resultant as an algebraic attack method to solve multivariate polynomial quadratic systems over finite fields.

Author Contributions: G.D. contributed to conceptualization, methodology and writing—original draft preparation. N.Q. contributed to formal analysis, resources, software and writing—review and editing. M.T. contributed to validation and writing—review and editing. X.D. contributed to project administration, supervision and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Guangxi Science and Technology project under Grant (No. Guike AD18281024), the Guangxi Key Laboratory of Cryptography and Information Security under Grant (No. GCIS201821), the Guilin University of Electronic Technology Graduate Student Excellent Degree Thesis Cultivation Project under Grant (No. 2020YJSPYB02), and Innovation Project of GUET Graduate Education (No. 2022YCXS144).

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors wish to thank the referee for his or her very helpful comments and useful suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mohammadi, A.; Horn, J.; Gregg, R.D. Removing phase variables from biped robot parametric gaits. In Proceedings of the IEEE Conference on Control Technology and Applications—CCTA 2017, Waimea, HI, USA, 27–30 August 2017; pp. 834–840.
2. Jaubert, O.; Cruz, G.; Bustin, A.; Schneider, T.; Lavin, B.; Koken, P.; Hajhosseiny, R.; Doneva, M.; Rueckert, D.; René, M.B. Water-fat Dixon cardiac magnetic resonance fingerprinting. *Magn. Reson. Med.* **2020**, *83*, 2107–2123. [CrossRef] [PubMed]
3. Winkler, J.R.; Halawani, H. The Sylvester and Bézout Resultant Matrices for Blind Image Deconvolution. *J. Math. Imaging Vis.* **2018**, *60*, 1284–1305. [CrossRef]
4. Lewis, R.H.; Paláncz, B.; Awange, J.L. Solving geoinformatics parametric polynomial systems using the improved Dixon resultant. *Earth Sci. Inform.* **2019**, *12*, 229–239. [CrossRef]
5. Paláncz, B. Application of Dixon resultant to satellite trajectory control by pole placement. *J. Symb. Comput.* **2013**, *50*, 79–99. [CrossRef]
6. Tang, X.J.; Feng, Y. Applying Dixon Resultants in Cryptography. *J. Softw.* **2007**, *18*, 1738–1745. [CrossRef]
7. Gao, Q.; Olgac, N. Dixon Resultant for Cluster Treatment of LTI Systems with Multiple Delays. *IFAC-PapersOnLine* **2015**, *48*, 21–26. [CrossRef]
8. Han, P.K.; Horng, D.E.; Gong, K.; Petibon, Y.; Kim, K.; Li, Q.; Johnson, K.A.; Georges, E.F.; Ouyang, J.; Ma, C. MR-Based PET Attenuation Correction using a Combined Ultrashort Echo Time/Multi-Echo Dixon Acquisition. *Math. Phys.* **2020**, *47*, 3064–3077. [CrossRef] [PubMed]
9. Yang, L.; Zhang, J.; Hou, X. *Nonlinear Algebric Equation System and Automated Theorem Proving*; Shanghai Scientific and Technological Education Publishing House: Shanghai, China, 1996; ISBN 7-5428-1379-X.
10. Tang, M.; Yang, Z.; Zeng, Z. Resultant elimination via implicit equation interpolation. *J. Syst. Sci. Complex.* **2016**, *29*, 1411–1435. [CrossRef]
11. Fu, H.; Zhao, S. Fast algorithm for constructing general Dixon resultant matrix. *Sci. China Math.* **2005**, *35*, 1–14. (In Chinese) [CrossRef]
12. Zhao, S. Dixon Resultant Research and New Algorithms. Ph.D. Thesis, Graduate School of Chinese Academy of Sciences, Chengdu Institute of Computer Applications, Chengdu, China, 2006.
13. Zhao, S.; Fu, H. An extended fast algorithm for constructing the Dixon resultant matrix. *Sci. China Math.* **2005**, *48*, 131–143. [CrossRef]
14. Zhao, S.; Fu, H. Three kinds of extraneous factors in Dixon resultants. *Sci. China Math.* **2009**, *52*, 160–172. [CrossRef]
15. Fu, H.; Wang, Y.; Zhao, S.; Wang, Q. A recursive algorithm for constructing complicated Dixon matrices. *Appl. Math. Comput.* **2010**, *217*, 2595–2601. [CrossRef]
16. Qin, X.; Wu, D.; Tang, L.; Ji, Z. Complexity of constructing Dixon resultant matrix. *Int. J. Comput. Math.* **2017**, *94*, 2074–2088. [CrossRef]
17. Lewis, R.H. Heuristics to accelerate the Dixon resultant. *Math. Comput. Simul.* **2008**, *77*, 400–407. [CrossRef]
18. Guo, X.; Leng, T.; Zeng, Z. The Fermat-Torricelli problem on sphere with euclidean metric. *J. Syst. Sci. Math. Sci.* **2018**, *38*, 1376–1392. [CrossRef]

19. Kotsireas, I.S.; Karamanos, K. Exact Computation of the bifurcation Point B_4 of the logistic Map and the Bailey-broadhurst Conjectures. *Int. J. Bifurc. Chaos* **2004**, *14*, 2417–2423. [CrossRef]
20. Lewis, R.H. Comparing acceleration techniques for the Dixon and Macaulay resultants. *Math. Comput. Simul.* **2010**, *80*, 1146–1152. [CrossRef]
21. Candes, E.J. Mathematics of sparsity (and a few other things). In Proceedings of the International Congress of Mathematicians 2017, Seoul, Korea, 13–21 August 2014; pp. 1–27.
22. Hu, J.; Monagan, M.B. A Fast Parallel Sparse Polynomial GCD Algorithm. In Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation—ISSAC 2016, Waterloo, ON, Canada, 19–22 July 2016; Abramov, S.A., Zima, E.V., Gao, X., Eds.; ACM: New York, NY, USA, 2016; pp. 271–278.
23. Qiu, W.; Skafidas, E. Robust estimation of GCD with sparse coefficients. *Signal Process.* **2010**, *90*, 972–976. [CrossRef]
24. Cuyt, A.A.M.; Lee, W. Sparse interpolation of multivariate rational functions. *Theor. Comput. Sci.* **2011**, *412*, 1445–1456. [CrossRef]
25. Dixon, A. The eliminant of three quantics in two independent variables. *Proc. Lond. Math. Soc.* **1909**, *s2-7*, 49–69. [CrossRef]
26. Li, B.; Liu, Z.; Zhi, L. A structured rank-revealing method for Sylvester matrix. *J. Comput. Appl. Math.* **2008**, *213*, 212–223. [CrossRef]
27. Zhao, S.; Fu, H. Multivariate Sylvester resultant and extraneous factors. *Sci. China Math.* **2010**, *40*, 649–660. [CrossRef]
28. Minimair, M. Computing the Dixon Resultant with the Maple Package DR. In Proceedings of the Applications of Computer Algebra (ACA), Kalamata, Greece, 20–23 July 2015; Kotsireas, I., MartinezMoro, E., Eds.; ACA: Kalamata, Greece, 2017; pp. 273–287.
29. Kapur, D.; Saxena, T.; Yang, L. Algebraic and Geometric Reasoning Using Dixon Resultants. In Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC '94, Oxford, UK, 20–22 July 1994; MacCallum, M.A.H., Ed.; ACM: New York, NY, USA, 1994; pp. 99–107.
30. Lu, Z. The Software of Gather2and2sift Based on Dixon Resultant. Ph.D. Thesis, Graduate School of Chinese Academy of Sciences, Beijing, China, 2003.
31. Chionh, E.; Zhang, M.; Goldman, R.N. Fast Computation of the Bézout and Dixon Resultant Matrices. *J. Symb. Comput.* **2002**, *33*, 13–29. [CrossRef]
32. Foo, M.; Chionh, E. Corner edge cutting and Dixon A-resultant quotients. *J. Symb. Comput.* **2004**, *37*, 101–119. [CrossRef]
33. Qin, X.; Feng, Y.; Chen, J.; Zhang, J. Parallel computation of real solving bivariate polynomial systems by zero-matching method. *Appl. Math. Comput.* **2013**, *219*, 7533–7541. [CrossRef]
34. Qin, X.; Yang, L.; Feng, Y.; Bachmann, B.; Fritzson, P. Index reduction of differential algebraic equations by differential Dixon resultant. *Appl. Math. Comput.* **2018**, *328*, 189–202. [CrossRef]
35. Lay, D.C. *Linear Algebric and Its Applications*; Addison-Wesley: Boston, MA, USA, 2013; ISBN 0321385178.
36. Blumenthal, L.M. *Theory and Applications of Distance Geometry*, 2nd ed.; Chelsea House Pub: New York, NY, USA, 1970; ISBN 978-0828402422.