 CMC, vol.58, no.2, pp.567-583, 2019

Optimization Algorithm for Reduction the Size of Dixon Resultant Matrix: A Case Study on Mechanical Application

Shang Zhang[1, *], Seyedmehdi Karimi[2], Shahaboddin Shamshirband[3, 4, *] and Amir Mosavi[5, 6]

Abstract: In the process of eliminating variables in a symbolic polynomial system, the extraneous factors are referred to the unwanted parameters of resulting polynomial. This paper aims at reducing the number of these factors via optimizing the size of Dixon matrix. An optimal configuration of Dixon matrix would lead to the enhancement of the process of computing the resultant which uses for solving polynomial systems. To do so, an optimization algorithm along with a number of new polynomials is introduced to replace the polynomials and implement a complexity analysis. Moreover, the monomial multipliers are optimally positioned to multiply each of the polynomials. Furthermore, through practical implementation and considering standard and mechanical examples the efficiency of the method is evaluated.

Keywords: Dixon resultant matrix, symbolic polynomial system, elimination theory, optimization algorithm, computational complexity.

1 Introduction

Along with the advancement of computers, during the past few decades, the search for advanced solutions to polynomials has received renewed attention. This has been due to their importance in theoretical, as well as the practical interests including robotics [Sun (2012)], mechanics [Wang and Lian (2005)], kinematics [Zhao, Wang and Wang (2017)], computational number theory [Kovács and Paláncz (2012)], solid modeling [Tran (1998)], quantifier elimination and geometric reasoning problems [Qin, Yang, Feng et al. (2015)]. Without explicitly solving for the roots, the difficulties in solving a polynomial is to identify the coefficients conditions where the system meets a set of solutions [Li (2009)]. These conditions are called resultant. One possible theory, which is commonly used to

[1] College of Computer and Information Technology, China Three Gorges University, Yichang, China.

[2] Department of Mathematics, Jouybar branch, Islamic Azad University, Jouybar, Iran.

[3] Department for Management of Science and Technology Development, Ton Duc Thang University, Ho Chi Minh City, Vietnam.

[4] Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam.

[5] Department Institute of Structural Mechanics, Bauhaus University of Weimar, Germany.

[6] Institute of Automation, Kando Kalman Faculty of Electrical Engineering, Obuda University, 1431 Budapest, Hungary.

[*] Corresponding Authors: Shang Zhang. Email: wetoo@ctgu.edu.cn;
Shahaboddin Shamshirband. Email: shahaboddin.shamshirband@tdtu.edu.vn.

find the resultant and solve a polynomial system, is elimination of the variables [Yang, Zeng and Zhang (2012)]. There are two major class of formulation for eliminating the variables of a polynomial system in matrix based methods to compute the resultants. They are called Sylvester resultant [Zhao and Fu (2010)] and Bezout-Cayley resultant [Palancz (2013); Bézout (2010)]. Both of these methods aim at eliminating n variables from $n + 1$ polynomials via developing resultant matrices. The algorithm that adapted for this article is inspired by Dixon method which is of Bezout-Cayley type.

The Dixon method is considered as an efficient method for identifying a polynomial. This polynomial would also include the resultant of a polynomial system which in some literature is known as projection operator [Chtcherba (2003)]. Dixon method produces a dense resultant matrix which is considered as an arrangement of the non-existence of a great number of zeroes in rows and columns of the matrix. In addition, the Dixon method produces a small resultant matrix in a lower dimension. The Dixon method's uniformity which is being implied as computing the projection operator without considering a particular set of variables is considerable properties. Besides, the method is automatic, and, therefore it eliminates the entire variables at once [Chtcherba (2003); Kapur, Saxena and Yang (1994)].

The majority of multivariate resultant methods, perform some multiplications for resultant [Faug'ere, Gianni, Lazard et al. (1992) ; Feng, Qin, Zhang et al. (2011)]. these multiplications do not deliver any insight into the solutions of the polynomial system [Saxena (1997)]. In fact, they only perform the multiplicative product of the resultant which include a number of extraneous factors. Nonetheless, these extraneous factors are not desirable resulting problems in a number of critical cases [Chtcherba (2003); Saxena (1997)]. Worth mentioning that Dixon method highly suffers from this drawback [Chtcherba and Kapur (2003); Chtcherba and Kapur (2004)].

However, in the polynomial systems, a Dixon matrix well deals with the conversion of the exponents of the polynomials [Lewis (2010)]. With this property, the Dixon method is highly capable of controlling the size of the matrix. Via utilizing this property, this research aims at optimizing the size of Dixon matrix aiming at the simplification of the solving process and gaining accuracy. In this regards, there has been similar cases reported in the literature on optimizing the Dixon resultant formulation e.g., [Chtcherba and Kapur (2004); Saxena (1997)].

This paper presents a method to finding optimally designed Dixon matrix for identifying smaller degree of the projection operator using dependency of the size of Dixon matrix to the supports of polynomials in the polynomial system, and dependency of total degree of the projection operator to the size of Dixon matrix. Having investigated these relations, some virtual polynomials have been presented to replace with original polynomials in the system to suppress the effects of supports of polynomials on each other. Applying this replacement, the support hulls of polynomials can be moved solely to find the best position to make smallest Dixon matrix. These virtual polynomials are generated by considering the widest space needed for support hulls to be moved freely without going to negative coordinate. In order to find the best position of support hulls related to each other, monomial multipliers are created to multiply to the polynomials of the system while original polynomial system is considered in the condition that the support of

monomial multipliers are located in the origin. Starting from the origin and choosing all neighboring points as support of monomial multiplier for each polynomial to find smaller Dixon matrix, the steps of optimization algorithm have been created. This procedure should be done iteratively for finding a monomial multiplier set to multiply to polynomials in the system for optimizing of Dixon matrix. This will lead to less extraneous factors in the decomposed form. Further, a number of sample problems are solved using the proposed method and the results are compared with the conventional methods.

The paper is organized as follow. In Section 2, we describe the method of Dixon construction by providing the details and further evidences. In addition, an algorithm for optimizing the size of Dixon matrix has been implemented and tested in some examples, along with the complexity analysis of the optimization algorithm in this section the advantages of presented algorithm is deriving the conditions under which the support hulls of polynomials in a polynomial system do not effect on each other during the optimizing. The comparisons made with relevant optimizing heuristic of Chtcherba show the superiority of the new optimization algorithms to present the results with regards to accuracy. Finally, in the Section 3, a discussion and conclusion remarks are given

2 Methodology

In this section the optimization algorithm of the Dixon matrix and the related formulation procedure is illustrated via flowchart with some related information and theorems.

2.1 Degree of the projection operator and the size of the Dixon matrix

Consider a multivariate polynomial $\mathrm{f} \in \mathbb{Z}[\mathrm{c}, \mathrm{x}]$. The set $\mathcal{A} \subset \mathbb{N}^{\mathrm{d}}$, is a finite set of exponents referred as the support of the f. Further, the polynomial system $\mathcal{F} = \{\mathrm{f}_0, \mathrm{f}_1, \dots, \mathrm{f}_{\mathrm{d}}\}$, with support $\mathcal{A} = \langle \mathcal{A}_0, \mathcal{A}_1, \cdots, \mathcal{A}_{\mathrm{d}} \rangle$ is named unmixed if $\mathcal{A}_0 = \mathcal{A}_1 = \cdots = \mathcal{A}_{\mathrm{d}}$, and is called mixed otherwise. In Dixon formulation, the computed projection operator will not be able to efficiently adapt to mixed systems, and therefore, for mixed systems, the Dixon resultant formulation is almost guaranteed to produce extraneous factors. This is a direct consequence of the exact Dixon conditions theorem, presented as follow from Saxena [Saxena (1997)]:

Theorem 1. In generic d-degree cases, the determinant of the Dixon matrix, is exactly its resultant (i.e., does not have any extraneous factors).

While, generally, most of the polynomial systems are non-generic or not d-degree.

Considering simplex form of the Dixon polynomial [Chtcherba and Kapur (2004)], every entry in the Dixon matrix Θ is obviously a polynomial of the coefficients of system $\mathcal{F}$ which its degree in the coefficients of any single polynomial is at most 1. Then the projection operator which is computed, is utmost of total degree $|\Delta_{\mathcal{A}}|$ in the coefficients of any single polynomial. Note that $|\Delta_{\mathcal{A}}|$ is the number of columns of Dixon matrix. A similar illustration can be presented for $|\Delta_{\bar{\mathcal{A}}}|$ (the number of rows of Dixon matrix) when the transpose of Dixon matrix be considered. Then,

$$\max\{|\Delta_{\mathcal{A}}|, |\Delta_{\bar{\mathcal{A}}}|\}, \tag{1}$$

is an upper bound for the degree of projection operator created using Dixon formulation in the coefficients of any single polynomial, where $\mathcal{A}$ and $\bar{\mathcal{A}}$ are the supports of xs and $\bar{\mathrm{x}}$s

(new variables presented in the Dixon method instruction) in the Dixon polynomial, respectively. Then minimizing the size of Dixon matrix leads to minimizing the degree of projection operator and decreasing the number of extraneous factors which exist next to the resultant.

2.2 Supports converting and its effects on Dixon matrix

The Dixon matrix size is not invariant with respect to the relative position of the support hulls (the smallest convex set of points in a support in an affine space) of polynomials [Lewis (2010)]. In other word, the Dixon resultant matrix for a polynomial $\mathcal{F}$ with the support set of $\mathcal{A}+\mathrm{c}=\langle\mathcal{A}_0+\mathrm{c}_0,\ldots,\mathcal{A}_\mathrm{d}+\mathrm{c}_\mathrm{d}\rangle$ is not the same as a system with support set $\mathcal{A}=\langle\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_\mathrm{d}\rangle$, where $\mathrm{c}=\langle\mathrm{c}_0,\mathrm{c}_1,\ldots,\mathrm{c}_\mathrm{d}\rangle$ and $\mathrm{c}_\mathrm{i}=\left(\mathrm{c}_{\mathrm{i},1},\ldots,\mathrm{c}_{\mathrm{i},\mathrm{d}}\right)\in\mathbb{N}^\mathrm{d}$ for $\mathrm{i}=0,1,\ldots,\mathrm{d}$. We call $\mathcal{A}_\mathrm{i}+\mathrm{c}_\mathrm{i}$ the converted support of f_i and $\mathcal{A}+\mathrm{c}=\langle\mathcal{A}_0+\mathrm{c}_0,\ldots,\mathcal{A}_\mathrm{d}+\mathrm{c}_\mathrm{d}\rangle$ is converted support set of polynomial system $\mathcal{F}$. However, as theorem 2 [Chtcherba (2003)], if a conversion is performed uniformly on all supports of polynomials in a system, the size of the Dixon matrix will not change.

Theorem 2. As the support for a generic polynomial system, consider $\mathcal{A}+\mathrm{c}$, where $\mathrm{c}_\mathrm{i}=\mathrm{c}_\mathrm{j}=\mathrm{t}=(\mathrm{t}_1,\mathrm{t}_2,\ldots,\mathrm{t}_\mathrm{d})$ for all $\mathrm{i},\mathrm{j}=0,1,\ldots,\mathrm{d}$, then

$\Delta_{\mathcal{A}+\mathrm{c}}=(\mathrm{t}_1,2\mathrm{t}_2,\ldots,\mathrm{dt}_\mathrm{d})+\Delta_{\mathcal{A}}$,

$\bar{\Delta}_{\mathcal{A}+\mathrm{c}}=(\mathrm{dt}_1,(\mathrm{d}-1)\mathrm{t}_2,\ldots,\mathrm{t}_\mathrm{d})+\bar{\Delta}_{\mathcal{A}}$,

where "+" is the Minkowski sum [Chtcherba (2003)], and Δ , $\bar{\Delta}$ are the support of Dixon polynomial for the original variables and new variables respectivly.

Consider a polynomial system $\mathcal{F}$. Assuming $\mathrm{f}_\mathrm{i}=\mathrm{hf'}_\mathrm{i}$ for some polynomials h and $\mathrm{f'}_\mathrm{i}$, clearly;

$$\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{f}_\mathrm{i},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d})=\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{h},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d})\times\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{f'}_\mathrm{i},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d}). \quad (2)$$

where Res_V is resultant of $\mathcal{F}$ over verity V. In particular, if h be considered as a monomial, the points where satisfy the part $\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{h},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d})$ are on the axis and the degree of this resultant is not more than $\max_{\mathrm{i}=1,\ldots,\mathrm{d}}(\mathrm{d}_{\max_\mathrm{i}}-1)$ where $\mathrm{d}_{\max_\mathrm{i}}$ is maximum total degree of f_i. If we do not consider the axis, we have;

$$\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{f}_\mathrm{i},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d})=\mathrm{Res}_\mathrm{V}(\mathrm{f}_0,\ldots,\mathrm{f}_{\mathrm{i}-1},\mathrm{f'}_\mathrm{i},\mathrm{f}_{\mathrm{i}+1},\ldots,\mathrm{f}_\mathrm{d}). \quad (3)$$

Since, we consider 1 as a variable for $\mathrm{d}+1$, polynomials in d variables and the resultant computed using Dixon formulation has property (3), a polynomial in a system could be multiplied by a monomial, without changing the resultant [Chtcherba and Kapur (2004)]. A direct consequence from above illustration and considering simplex form of Dixon formulation [Chtcherba and Kapur (2004)], is the sensitivity of the size of matrix to exponent of multipliers to original the polynomial system.

2.3 Optimizing method for Dixon matrix

Considering the point that, $|\Delta_{\mathcal{A}}|\neq|\Delta_{\mathcal{A}+\mathrm{t}}|$ and/or $|\bar{\Delta}_{\mathcal{A}}|\neq|\bar{\Delta}_{\mathcal{A}+\mathrm{t}}|$ unless $\mathrm{c}_\mathrm{i}=\mathrm{c}_\mathrm{j}$ for all $\mathrm{i},\mathrm{j}=0,1,\ldots,\mathrm{d}$ from Sections 2.1, 2.2 and [Saxena (1997)], it can be said: the size of

Dixon matrix depends on the position of support hulls of polynomials in relation with each other. Besides, there is an direct dependency between the area overlapped by convex hulls of polynomials and the size of Dixon matrix [Saxena (1997)].

Taking advantage of above properties, this paper intend present an optimization algorithm with conversion set $c = \langle c_0, c_1, \dots, c_d \rangle$ where $c_i = (c_{i,1}, \dots, c_{i,d}) \in \mathbb{N}^d$ for $i = 0,1, \dots, d$ which is considered for converting the support of a polynomial system $\mathcal{A} = \langle \mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_d \rangle$, to make $|\Delta_{\mathcal{A}}|$ and $|\bar{\Delta}_{\mathcal{A}}|$ smaller. The converted support set for polynomial system appears as $\mathcal{A} + c = \langle \mathcal{A}_0 + c_0, \dots, \mathcal{A}_d + c_d \rangle$. In the other words, if the polynomial system is considered in $\mathbb{Z}[c][x_1 \dots x_d]$, the algorithm multiplies f_i to the monomial $x_1^{c_{i,1}} \dots x_d^{c_{i,d}}$ to shift the polynomials support hull to find smaller Dixon matrix. The minimizing method is sequential and has initial guess for c_0 at the beginning. The choice of c_0 should be so that other c_i could be chosen without getting in to negative coordinates. Then, search for c_1 is beginning from origin and will be continued by a trial and error method. The turn of c_2 is after c_1 when the support of multiplier for f_1 is fixed then, c_3 and so on. The process continues for finding all c_is.

Not considering the effect of support hulls on each other is a disadvantage of the previous minimizing method [Chtcherba (2003)]. In fact, sometimes the effects of the support hulls on each other lead the algorithm to the wrong direction of optimization. In the other word, giving high relative distance between convex hulls of supports, the algorithm for optimizing fails and ends up with incorrect results as is noted in the end of solved examples in this section.

To suppress the effect of support hulls on each other, during of running the new algorithm presented in this paper, some polynomials should be replaced by some new polynomials, which are called virtual polynomials. The rules of selecting and using of virtual polynomials are presented in details in this section.

In the presented optimization approach, moving of support hulls of polynomials in system of coordinates is divided in four phases as;

Phase 1: Choosing c_0,

Phase 2: Shifting the support hull of f_0 by multiplying x^{c_0} to f_0,

Phase 3: Presenting the virtual polynomials,

Phase 4: Converting of other supports.

Phase 1, 2: The space needed for executing the optimization algorithm is presented in dimension of S, where

$S = (s_1, s_2, \dots, s_d)$,

$$s_i = c_{0,i} + \max_{\gamma \in \mathcal{A}_0} \gamma_i + \max_{\gamma \in \mathcal{A}_1 \cup \dots \cup \mathcal{A}_{d-1}} \gamma_i \quad \text{for } i = 1,2, \dots, d. \tag{4}$$

Here, $\gamma = (\gamma_1, \dots, \gamma_d)$ is a d-dimensional integer vector. Considering the following polynomial system for optimization,

$$\begin{cases} f_0(x_1, \cdots, x_d) = 0, \\ f_1(x_1, \cdots, x_d) = 0, \\ \vdots \\ f_d(x_1, \cdots, x_d) = 0, \end{cases}$$

The choice of S is highly dependent to choice of c_0 that choosing bigger $c_{0,i}$ make the s_i bigger and vice versa. Whereas the complexity of the algorithm is also depending on the choice of c_0, following definition for c_0 is presented for controlling the time complexity, however one can choose bigger elements for c_0. Then $c_0 = \langle c_{0,1}, c_{0,2}, \dots, c_{0,d} \rangle$ is introduced as;

$$c_{0,i} = \max_{\gamma \in \cup_{j=1,2,\dots,d} \mathcal{A}_j} (\gamma_i) \text{ for } i = 1,2,\dots,d. \quad (5)$$

The above choice for c_0 can be explained by the fact that it can guarantee to move other support hulls to stay in positive coordinate when they want to approach $\mathcal{A}_0 + c_0$. So the $c_{0,i}$ should be chosen as above equation or bigger.

Phase 3: By searching for c_p as the best monomial multiplier for $f_p, p = 1, \dots, d-1$ the virtual polynomials are supposed as $v_{p+1}, v_{p+2}, \dots, v_d$ in total degrees of $s, s+1, \dots, s+(d-p-1)$ respectively, where $s = \sum_{i=1}^{d} s_i$. These virtual polynomials are considered without any symbolic coefficient and each one should have multifaceted vertical corner support hull. For example, in two-dimensional form ($x = (x_1, x_2)$), if $S = (s_1, s_2)$, the following polynomial is considered as virtual polynomial to replace by f_2 when the algorithm intended to find optimal c_1.

$v_2 = 1 + {x_1}^{s_1} + {x_2}^{s_2} + {x_1}^{s_1}{x_2}^{s_2}$,

Having investigated the above relations, some virtual polynomials have been found to replace the original polynomials to suppress the effects of support of polynomials. Applying this replacement, the support hulls of polynomials can be moved solely to find the best position to make smallest Dixon matrix. These virtual polynomials are generated by considering the widest space needed for support hulls to be moved freely without going to negative coordinate and having effects on each other.

Phase 4: For the purpose of finding the optimal positions of support hulls related to each other, monomial multipliers are created to multiply to the polynomials of the system while original polynomial system is considered in the condition that the support of monomial multipliers are located in the origin. Starting from the origin and choosing all neighboring points as support of monomial multiplier for each polynomial to find smaller Dixon matrix, the steps of optimization algorithm have been created. This procedure should be done iteratively for finding a monomial multiplier set to multiply to polynomials in the system for optimizing of Dixon formulation.

The algorithm of the method for optimizing the size of matrix is presented by flowchart shown in following Figure.

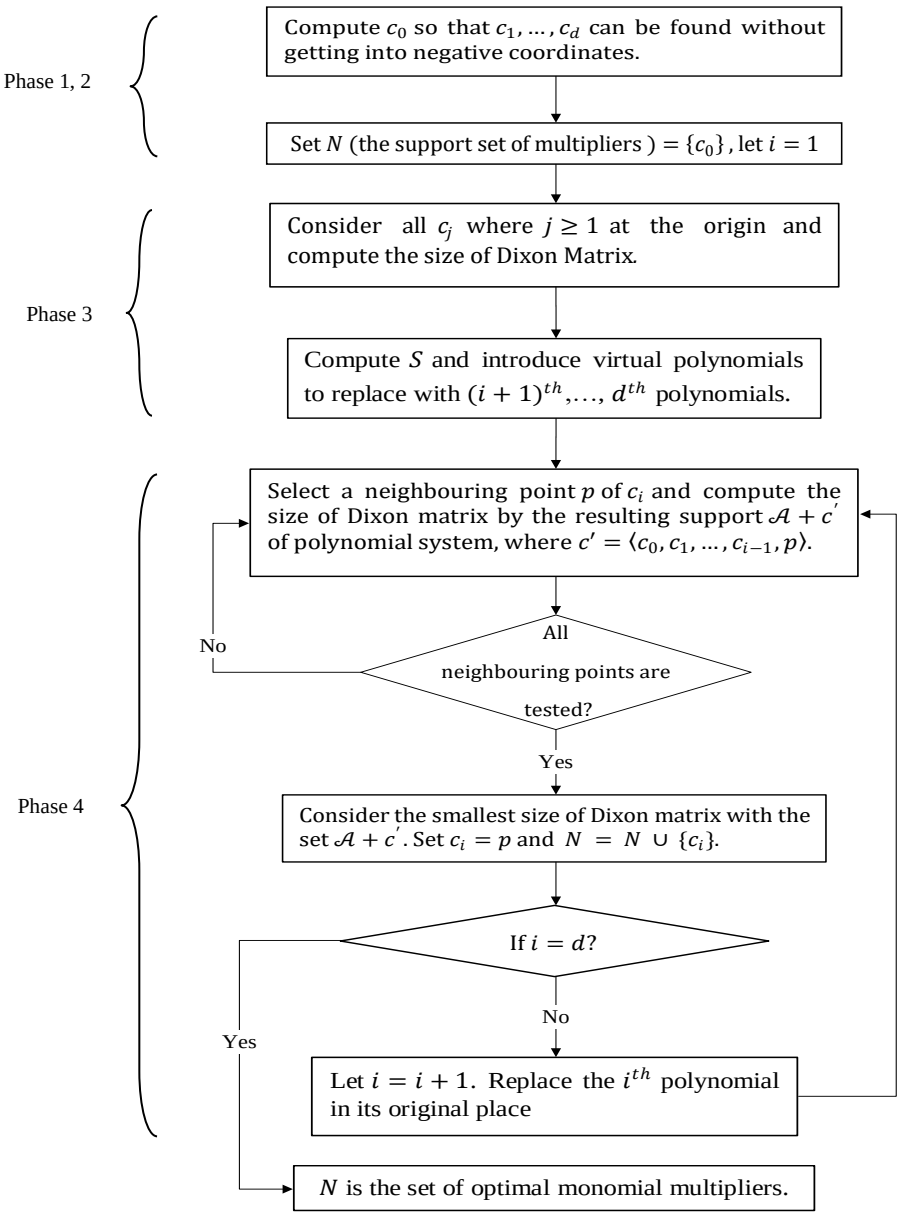

Figure 1: Flowchart for optimizing of Dixon matrix

Here, the complexity of optimization direction for the system of polynomials $\mathcal{F} = \{f_0, f_1, \dots, f_d\}$ is obtained via recognition of search area in addition to the cost of receiving better Dixon matrix in the reiterative phases. The reiterative method phases are illustrated in Fig. 1 in form of flowchart. If we consider k as $k = \max_{i=1,\dots,d} (s_i + (d - i - 1))$ with regard to the point that for each c_i we can have maximum kd shifting, we should account Dixon matrix size in maximum kd times to arrive the best size, where we are selecting

the neighboring point p (See Fig. 1, phase 4). Complexity of finding each Dixon matrix has been bounded by $O(d^2n^d)$ where $n = \left|\cup_{i=1}^{d} \mathcal{A}_i\right|$ [Qin, Wu, Tang et al. (2017); Grenet, Koiran and Portier (2013)]. Therefore, in each phase of selecting all neighboring points and considering the smallest size of the Dixon matrix, we have a complexity as $O(kd^3n^d)$. For getting general answer, the optimizing direction is repeated d times as each complexity are considered for each polynomial (see the step of checking "if $i = d$?" in Fig. 1). Then, we have total complexity of presented algorithm as $O(kd^4n^d)$.

Example 1 Considering the bellow mixed polynomial system from Chtcherba [Chtcherba (2003)]:

$$\mathcal{F} = \begin{cases} f_0 &= a_{01} + a_{02}x^2 + a_{03}x^3y^6 + a_{04}x^7y^6 \ , \\ f_1 &= a_{11}x + a_{12}y^7 + a_{13}x^2y^9 + a_{14}x^3y^9, \\ f_2 &= a_{21} + a_{22}x^2y^5 + a_{23}x^8y^4, \end{cases}$$

once x, y are considered as variables and a_{ij} are parameters. The size of Dixon matrix is 99×90 which 99 and 90 regard to the number of rows and the number of columns of the matrix respectively. The polynomials support is

$\mathcal{A}_0 = \{(0,0), (2,0), (3,6), (7,6)\}$, $\mathcal{A}_1 = \{(1,0), (0,7), (2,9), (3,9)\}$

$\mathcal{A}_2 = \{(0,0), (2,5), (8,4)\}$,

with support hulls

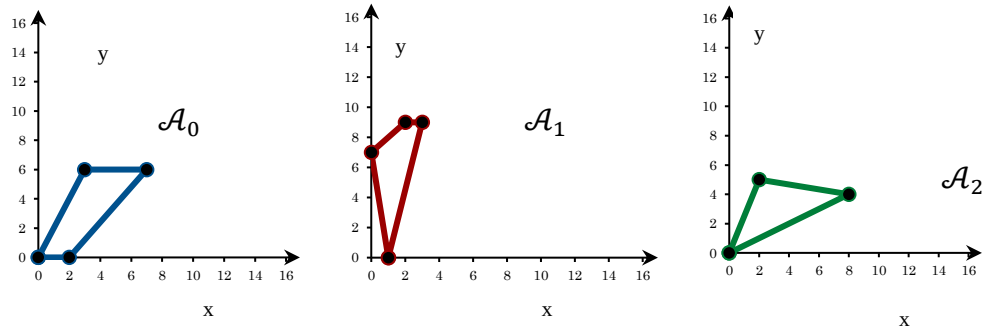

Figure 2: Support hulls of system, Ex. 1

To avoid negative exponents, we need to shift initial support $\mathcal{A}_0$. Using the formulation (5) we have $c_0 = (8,9)$. According to the presented algorithm, we need to replace f_2 with virtual polynomial v_2 for finding a monomial multiplier for f_1 with support c_1. Then using relation (4),

$S = (s_1, s_2) = (8 + 7 + 3, 9 + 6 + 9) = (18,24)$,

and the following polynomial system is achieved.

$$\begin{cases} f_0 &= a_{01}x^8y^9 + a_{02}x^{10}y^9 + a_{03}x^{11}y^{15} + a_{04}x^{15}y^{15} \ , \\ f_1 &= a_{11}x + a_{12}y^7 + a_{13}x^2y^9 + a_{14}x^3y^9, \\ v_2 &= 1 + x^{18} + y^{24} + x^{18}y^{24}. \end{cases}$$

The size of Dixon matrix is 765×675. The above polynomials system can be

considered in the case of $c_1 = (0,0)$. The best monomial for multiplying to f_1 will be found by trial and error method as x^9y^6. Now when c_0 and c_1 are fixed, the f_2 can return to its original place and we have following system which is ready to start process for finding c_2;

$$\begin{cases} f_0 &= a_{01}x^8y^9 + a_{02}x^{10}y^9 + a_{03}x^{11}y^{15} + a_{04}x^{15}y^{15}\ , \\ f_1 &= a_{11}x^{10}y^6 + a_{12}x^9y^{13} + a_{13}x^{11}y^{15} + a_{14}x^{12}y^{15}, \\ f_2 &= a_{21} + a_{22}x^2y^5 + a_{23}x^8y^4. \end{cases}$$

The size of Dixon matrix is 201×216. Using same method which is done for c_1, the best monomial multiplier for f_2 will be found as $x^8y^{10}$and Dixon matrix of size 82×82. The optimal resulted polynomial system is

$$\begin{cases} f_0 &= a_{01}x^8y^9 + a_{02}x^{10}y^9 + a_{03}x^{11}y^{15} + a_{04}x^{15}y^{15}\ , \\ f_1 &= a_{11}x^{10}y^6 + a_{12}x^9y^{13} + a_{13}x^{11}y^{15} + a_{14}x^{12}y^{15}, \\ f_2 &= a_{21}x^8y^{10} + a_{22}x^{10}y^{15} + a_{23}x^{16}y^{14}, \end{cases}$$

with optimized supports hulls which are shown in Fig. 3.

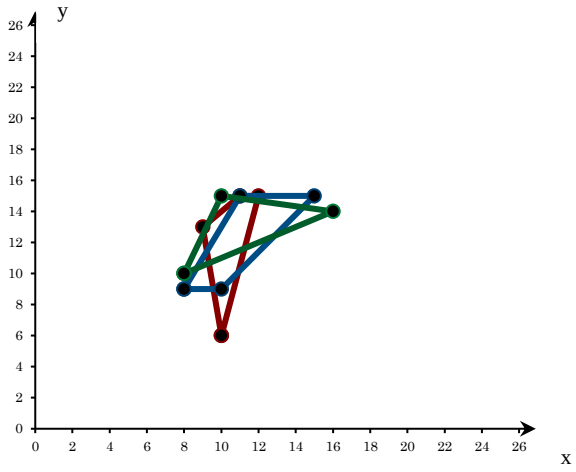

Figure 3: Support hulls of optimized system, Ex. 1

The steps of trial and error method for finding optimal c_1 and c_2 are summarized in following tables.

Table 1: Steps for finding c_1 and c_2 using presented method with $c_0 = (8,9)$, Ex

No	c_1	$\|\Theta\|$	No	c_1	$\|\Theta\|$
1.	(1,0)	733×666	9.	(8,1)	491×575
2.	(2,0)	700×657	10.	(8,2)	480×546
3.	(3,0)	667×648	11.	(8,3)	469×517
4.	(4,0)	634×639	12.	(8,4)	458×488
5.	(5,0)	601×630	13.	(8,5)	447×459
6.	(6,0)	658×621	14.	(8,6)	436×430
7.	(7,0)	535×612	15.	(9,6)	436×430
8.	(8,0)	502×603	16.	(10,6)	436×430

No	c_2	$\|\Theta\|$	No	c_2	$\|\Theta\|$
1.	(1,0)	192×202	11.	5,6)	114×121
2.	(2,0)	183×188	12.	(6,6)	110×112
3.	(2,1)	168×181	13.	(6,7)	104×107
4.	(2,2)	154×174	14.	(7,7)	99×99
5.	(2,3)	147×167	15.	(7,8)	92×92
6.	(2,4)	140×160	16.	(7,9)	86×87
7.	(2,5)	133×154	17.	(7,10)	83×85
8.	(2,6)	126×148	18.	(8,10)	82×82
9.	(3,6)	122×139	19.	(9,10)	89×89
10.	(4,6)	118×130	20.	(8,11)	86×85

Comparing the size of Dixon matrix after executing the algorithm and before executing in beginning of Example 1, the advantage of new presented optimizing method is evident. Optimizing the size of Dixon matrix using Chtcherba's presented heuristic [Chtcherba (2003)] shows a big failure where the size of Dixon matrix never becomes smaller than 369×306.

Example 2 Here the strophoid is considered. The strophoid is a curve widely studied by mathematicians in the past two century. It can be written in a parametric form described as follow.

$\mathrm{x} = \mathrm{a}\sin(\mathrm{t})\,,\ \mathrm{y} = \mathrm{a}\tan(\mathrm{t})(1+\sin(\mathrm{t}))$,

To find an implicit equation for the strophoid using resultant, we have to restate the equations in terms of polynomials instead of trigonometric functions, as follows.

Letting $\mathrm{S} = \sin\mathrm{t}\,, \mathrm{C} = \cos\mathrm{t}\,, \mathrm{T} = \tan\mathrm{t}$, the trigonometric equations of the strophoid can be written as

$$\begin{cases} \mathrm{f}_0 = \mathrm{C}^2 + \mathrm{S}^2 - 1 = 0 \\ \mathrm{f}_1 = \mathrm{CT} - \mathrm{S} = 0 \\ \mathrm{f}_2 = \mathrm{x} - \mathrm{aS} = 0 \\ \mathrm{f}_3 = \mathrm{y} - \mathrm{aT}(1+\mathrm{S}) = 0 \end{cases}$$

Using the variable ordered set $\langle \mathrm{C}, \mathrm{S}, \mathrm{T} \rangle$ the support sets are,

$\mathcal{A}_0 = \{(2,0,0), (0,2,0), (0,0,0)\}$, $\mathcal{A}_1 = \{(1,0,1), (0,1,0)\}$,

$\mathcal{A}_2 = \{(0,0,0),(0,1,0)\}, \quad \mathcal{A}_3 = \{(0,0,0),(0,0,1),(0,1,1)\},$
and the support hulls of polynomials is shown in Fig. 4 in system of 3-dimensions coordinate.

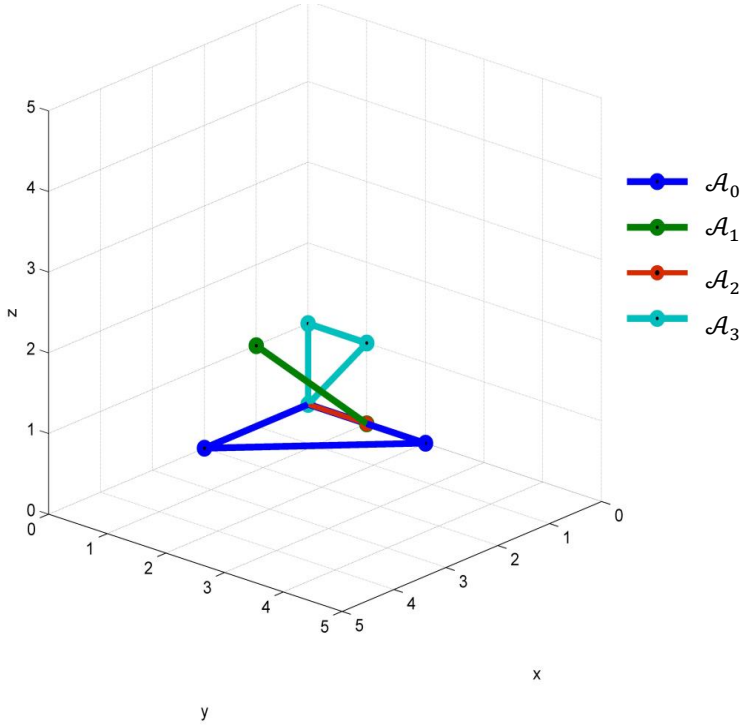

Figure 4: Support hulls of polynomials, Ex 1.2.2

For starting to search for finding best c_1 we introduce c_0 as $(1,1,1)$ using formula (5) and replace f_2, f_3 with virtual polynomial v_2, v_3 respectively. Then finding the vector S is required.

$S = (s_1, s_2, s_3) = (1+2+1, 1+2+1, 1+0+1) = (4,4,2),$

Therefore, we can continue the optimizing method which is dedicated to find c_1 using following polynomial system.

$$\begin{cases} f_0 = C^3ST + CS^3T - CST, \\ f_1 = CT - S, \\ v_2 = 1 + C^4 + S^4 + T^2 + C^4T^2 + S^4T^2 + C^4S^4 + C^4S^4T^2, \\ v_3 = 1 + C^5 + S^5 + T^3 + C^5T^3 + S^5T^3 + C^5S^5 + C^5S^5T^3. \end{cases}$$

The process is summarized in the Tab. 2.

Table 2: Steps for finding c_1 using presented method with assumption $c_0 = (1,1,1)$, Ex 2

No	c_1	$\lvert\Theta\rvert$
1.	(1,0,0)	267×248
2.	(1,1,0)	247×232
3.	(2,1,0)	247×232
4.	(1,2,0)	247×232
5.	(1,1,1)	247×232

The resulting polynomial system which is used for finding best c_2 is

$$\begin{cases} f_0 = C^3ST + CS^3T - CST, \\ f_1 = C^2ST - CS^2, \\ f_2 = x - aS, \\ v_3 = 1 + C^5 + S^5 + T^3 + C^5T^3 + S^5T^3 + C^5S^5 + C^5S^5T^3, \end{cases}$$

Results of algorithm with optimization are presented in the Tab. 3.

Table 3: Steps for finding c_2 by new presented method, Ex 2

No	c_2	$\lvert\Theta\rvert$
1.	(0,1,0)	58×70
2.	(1,1,0)	41×60
3.	(1,1,1)	39×44
4.	(2,1,1)	47×45
5.	(1,2,1)	45×46
6.	(1,1,2)	56×83

The process of finding best c_3, which can be seen in the Tab. 4, is derived from the original polynomial system with polynomials f_0, f_1 and f_2 multiplied by CST, CS and CST respectively.

$$\begin{cases} f_0 = C^3ST + CS^3T - CST, \\ f_1 = C^2ST - CS^2, \\ f_2 = xCST - aCS^2T, \\ f_3 = y - aT(1 + S). \end{cases}$$

Table 4: Steps for finding c_3 by new presented method, Ex 2

No	c_3	$\lvert\Theta\rvert$
1.	(1,0,0)	7×10
2.	(1,1,0)	5×6
3.	(1,2,0)	5×5
4.	(2,2,0)	7×9
5.	(1,3,0)	7×9
5.	(1,2,1)	10×7
6.	(0,2,0)	6×10

The set of support hulls of optimized polynomial system is shown in Fig. 5.

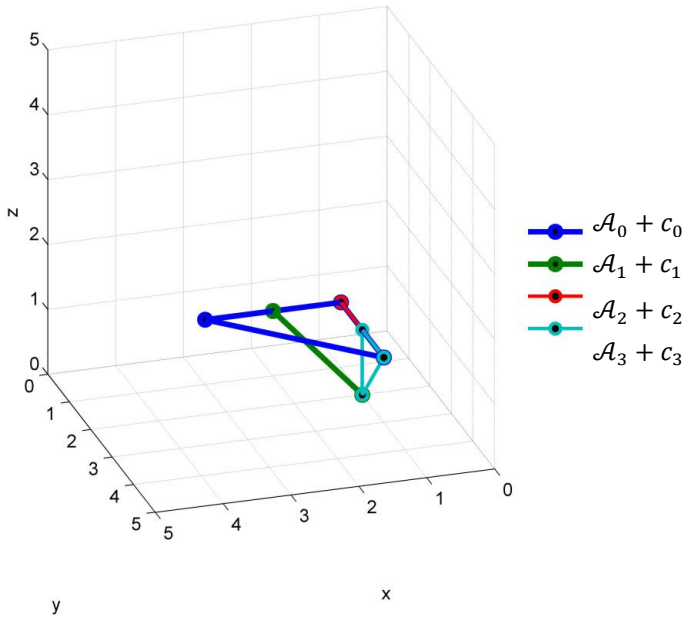

Figure 5: Support hulls of optimized polynomial system, Ex. 2

Comparing the presented result presented in Tab. 4 to the result of finding Dixon matrix [Chtcherba (2003)], which tells the Size of Dixon matrix is 6×5, we could minimize the size of Dixon matrix.

Example 3 The Stewart platform problem is a standard benchmark elimination problem of mechanical motion of certain types of robots. The quaternion formulation we present here is by Emiris [Emiris (1994)]. It contains 7 polynomials in 7 variables. Let $x = [x_0, x_1, x_2, x_3]$ and $q = [1, q_1, q_2, q_3]$ be two unknown quaternions, to be determined. Let $q^* = [1, -q_1, -q_2, -q_3]$. Let a_i and b_i for $i = 2, \ldots, 6$ be known quaternions and let α_i for $i = 1, \ldots, 6$ be six predetermined scalars. The 7 polynomials are:

$$f_0 = x^T x - \alpha_1 q^T q$$

$$f_i = {b_i}^T (xq) - {a_i}^T (qx) - (q b_i q^*)^T a_i - \alpha_i q^T q \qquad i = 1, \ldots, 5,$$

$$f_6 = x^T q^*$$

Out of the 7 variables $x_0, x_1, x_2, x_3, q_1, q_2, q_3$ any six are to be eliminated to compute the resultant as a polynomial in the seventh. Saxena [Saxena (1997)] successfully computed the Dixon matrix of the Stewart problem by eliminating 6 variables $x_0, x_1, x_2, x_3, q_2, q_3$. The size of his Dixon matrix is 56×56. To optimize the Saxena's resulted matrix, using our presented method, we should compute the c_0 according vector variable $(x_0, x_1, x_2, x_3, q_2, q_3)$, to avoid negative coordinate. Using formula (5), the c_0 is $(2,2,2,2,2,2)$. Then using formula (4), $S = (6,6,6,6,6,6)$ which helps us to find appropriate virtual polynomials v_2 , …, v_6 as stated by details in algorithm formulation. Now we can start to fine optimal c_1 to present the best monomial multiplier for f_1. Due to long process of finding best direction of optimizing for 6 considered variables, the

process which is presented in Tab. 5 is summarized.

Table 5: Steps for finding c_1 using presented method with assumption $c_0 = (2,2,2,2,2,2)$, Ex 3

No	c_1	$\|\Theta\|$
1,..., 5.	(1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,1,0,0,0), (0,0,0,0,1,0), (0,0,0,0,0,1)	Bigger than 1176 × 1176
6.	(0,0,0,1,0,0)	1176 × 1176
7,...,11	(0,1,0,1,0,0), (0,0,11,0,0), (0,0,0,2,0,0), (0,0,0,1,1,0), (0,0,0,1,0,1)	Bigger than 1065 × 1065
12.	(1,0,0,1,0,0)	1065 × 1065
13,...,17	(2,0,0,1,0,0), (1,0,11,0,0), (1,0,0,2,0,0), (1,0,0,1,1,0), (1,0,0,1,0,1)	Bigger than 1020 × 1020
18.	(1,1,0,1,0,0)	1020 × 1020
19,...,23	(1,2,0,1,0,0), (1,1,11,0,0), (1,1,0,2,0,0), (1,1,0,1,1,0), (1,1,0,1,0,1)	Bigger than 1015 × 1015
24.	(2,1,0,1,0,0)	1015 × 1015
25,...,29	(3,1,0,1,0,0), (2,1,11,0,0), (2,1,0,2,0,0), (2,1,0,1,1,0), (2,1,0,1,0,1)	Bigger than 997 × 997
30.	(2,2,0,1,0,0)	997 × 997
31,...,35	(3,2,0,1,0,0), (2,3,01,0,0), (2,2,1,1,0,0), (2,2,0,2,0,0), (2,2,0,1,1,0)	Bigger than 986 × 986
36.	(2,2,0,1,0,1)	986 × 986
37,...,41	(3,2,0,1,0,1), (2,3,01,0,1), (2,2,1,1,0,1), (2,2,0,2,0,1), (2,2,0,1,1,1)	Bigger than 980 × 980
42.	(2,2,0,1,0,2)	980 × 980
42,...,48	(3,2,0,1,0,2), (2,3,01,0,2), (2,2,1,1,0,2), (2,2,0,2,0,2), (2,2,0,1,1,2), (2,2,0,1,0,3)	Bigger than 980 × 980

Then the best monomial multiplier for f_1 is $x_0^2x_1^2x_3q_3^2$ which stand on best c_1 presented step number 42 in Tab. 5. Then the polynomial system can be prepared for doing the process of finding best c_2 by replacing v_2 with f_2. The same trial and error method is used for finding best c_2. It is found after 36 steps as (2,2,1,1,0,1) and the Dixon matrix size, at the beginning of the process, was 335×335 while at the end of optimizing process it was 286×286. Likewise, the other supports of monomial multipliers which are known as c_3, c_4, c_5 and c_6, are (2,2, 1,1,0,2), (2,1,0,2,1,1), (2,1,1,2,1,2) and (2,2,0,1,0,1) respectively, as it explained by details in [karimi (2012)]. Then, in optimized form, the polynomials f_i, $i = 0, \dots, 6$, should be multiplied by monomials $x_0^2x_1^2x_2^2x_3^2q_2^2q_3^2$, $x_0^2x_1^2x_3q_3^2$, $x_0^2x_1^2x_2x_3q_3$, $x_0^2x_1^2x_2x_3q_3^2$, $x_0^2x_1x_3^2q_2q_3$, $x_0^2x_1x_2x_3^2q_2q_3^2$ and $x_0^2x_1^2x_3q_3$ respectively and the Dixon matrix size of optimized polynomial system is 48×48. Comparing the size of Dixon matrix according to Saxena's achieved results (as mentioned at beginning of this example), the advantage of new presented optimizing method is evident.

3 Discussion and conclusion

Though considering the simplex form of the Dixon polynomial, the maximum number of rows and columns of the Dixon matrix was presented as an upper bound of the projection operator in the coefficients of any single polynomial. In addition, since we were working on the affine space, each polynomial in a system could be multiplied by a monomial, without changing the resultant. Moreover, another useful property of the Dixon matrix construction which has been revealed was the sensitivity of the size of Dixon matrix to support hull set of the given polynomial system. Therefore, multiplying some monomials to the polynomials, which were in the original system, changed the size of the Dixon

matrix yet had no effects on the resultant. It only changed the total degree of the projection operator which the resultant was a part. As long as the Dixon matrix had this property, the size of the Dixon matrix was able to be optimized by properly selecting the multipliers for polynomials in the system. Using this property, this paper sought to optimize the size of the Dixon matrix for the purpose of enhancing the efficiency of the solving process and identifying better results. Via considering the properties of Dixon formulation, it was concluded that, the size of Dixon matrix depends on the position of support hulls of the polynomials in relation with each other.

Furthermore, in order to suppress the effects of supports hulls of polynomials on each other, some virtual polynomials had been introduced to replace the original polynomials in the system. Applying this replacement, the support hulls of polynomials could be moved solely to find the best position to make smallest Dixon matrix. These virtual polynomials were generated by considering the biggest space needed for support hulls to be moved freely without going to negative coordinate. For the purpose to identify the optimal position of support hulls related to each other, monomial multipliers were created to multiply to the polynomials of the system while original polynomial system was considered on provided that the support of monomial multipliers were located in the origin.

The complexity analyses was performed for the corresponding algorithm namely the minimization method of the resultant matrix for the system of polynomials $\mathcal{F} = \{\mathrm{f}_0, \mathrm{f}_1, \dots, \mathrm{f}_\mathrm{d}\}$ considered by recognition of search area along with the time cost of receiving better Dixon matrix in every single phase.

For verifying the results by implementing the presented algorithm for optimizing the Dixon matrix for general polynomial systems, the algorithm was implemented and its applicability was demonstrated in example 1 (2 dimensions), example 2 (3 dimensions) and example 3 (6 dimensions). The results of the method for minimizing the size of Dixon resultant matrix were presented in tables that reveal the advantages and the practicality of the new method. Even if we had the optimal position of support hulls at the outset, the algorithm worked properly.

References

Bezout, E. (2010): Theorie generale des equations algebriques.

http://books. google. com. tw/books.

Chtcherba, A. (2003): *A New Sylvester-Type Resultant Method Based on the Dixon-Bezout Formulation (Ph.D. Thesis).* State University of New Mexico.

Chtcherba, A.; Kapur, D. (2003): Exact resultants for corner-cut unmixed multivariate polynomial systems using the Dixon formulation. *Journal of Symbolic Computation*, vol. 36**,** pp. 289-315.

Chtcherba, A. D.; Kapur, D. (2004): Support hull: relating the cayley-dixon resultant constructions to the support of a polynomial system. *ISSAC'04 Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pp. 95-102.

Dixon, A. L. (1908): The eliminant of three quantics in two independent variables. *London Mathematical Society*, vol. 6, pp. 468-478.

Dixon, A. L. (1909): The eliminate of three quantics in two independent variables. *Proceedings of The London Mathematical Society*, vol. s2-7, no. 1, pp. 49-69.

Emiris, I. (1994): *Sparse Elimination and Applications in Kinematics(Ph.D. Thesis)*. University of Calif.

Faug'**ere, J. C.; Gianni, P.; Lazard, D.; Mora, T.** (1992): Efficient computation of zero-dimensional grobner bases by change of ordering. *Journal of Symbolic Computation*, vol. 16, pp. 329-344.

Feng, Y.; Qin, X.; Zhang, J.; Yuan, X. (2011): Obtaining exact interpolation multivariate polynomial by approximation. *Journal of Systems Science and Complexity*, vol. 24, pp. 803-815.

Grenet, B.; Koiran, P.; Portier, N. (2013): On the complexity of the multivariate resultant. *Journal of Complexity*, vol. 29, pp. 142-157.

Kapur, D.; Saxena, T.; Yang, L. (1994): Algebraic and geometric reasoning using the Dixon resultants. *ACM ISSAC*, vol. 94**,** pp. 99-107.

Karimi, S. M. (2012): *New Algorithms for Optimizing the Sizes of Dixon and Dixon Dialytic Matrices(Ph.D. Thesis)*. University Technology Malaysia.

Kovács, L.; Paláncz, B. (2012): Solving robust glucose-insulin control by dixon resultant computations. *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 53-61.

Lewis, R. H. (2010): Comparing acceleration techniques for the Dixon and Macaulay resultants. *Mathematics and Computers in Simulation*, vol. 80, pp. 1146-1152.

Li, Y. (2009): An effective hybrid algorithm for computing symbolic determinants. *Applied Mathematics and Computation*, vol. 215, pp. 2495-2501.

Palancz, B. (2013): Application of Dixon resultant to satellite trajectory control by pole placement. *Journal of Symbolic Computation*, vol. 50, pp. 79-99.

Palancz, B.; Zaletnyik, P.; Awange, J. L.; Grafarend, E. W. (2008): Dixon resultant's solution of systems of geodetic polynomial equations. *Journal of Geodesy*, vol. 82, pp. 505-511.

Qin, X.; Wu, D.; Tang, L.; Ji, Z. (2017): Complexity of constructing Dixon resultant matrix. *International Journal of Computer Mathematics*, vol. 94, pp. 2074-2088.

Qin, X.; Yang, L.; Feng, Y.; Bachmann, B.; Fritzson, P. (2015): Index reduction of differential algebraic equations by differential algebraic elimination.

https://arxiv.org/abs/1504.04977.

Saxena, T. (1997): *Efficient Variable Elimination Using Resultants(Ph.D. Thesis)*. State University of New York.

Sun, W. K. (2012): Solving 3-6 parallel robots by dixon resultant. *Applied Mechanics and Materials*, vol. 235, pp. 158-163.

Tran, Q. N. (1998): Extended Dixon's resultant and its applications. In: Wang, D., ed. *Automated Deduction in Geometry*. Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 37-57.

Wang, W.; Lian, X. (2005): Computations of multi-resultant with mechanization. *Applied Mathematics and Computation*, vol. 170, pp. 237-257.

Yang, L.; Zeng, Z.; Zhang, W. (2012): Differential elimination with Dixon resultants. *Applied Mathematics and Computation*, vol. 218, pp. 10679-10690.

Zhao, S.; Fu, H. (2010): Multivariate Sylvester resultant and extraneous factors. *Scientia Sinica Mathematica*, vol. 40, pp. 649-660.

Zhao, Z.; Wang, T.; Wang, D. (2017): Inverse kinematic analysis of the general 6R serial manipulators based on unit dual quaternion and Dixon resultant. *Chinese Automation Congress*, pp. 2646-2650.