A new algorithm for improved determinant computation with a view towards resultants

by

Khalil Shivji

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Honours Bachelor of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Fall 2019

Approval

Name:	**Khalil Shivji**
Degree:	**Honours Bachelor of Science (Mathematics)**
Title:	**A new algorithm for improved determinant computation with a view towards resultants**
Supervisory Committee:	**Dr. Michael Monagan** Supervisor Professor of Mathematics **Dr. Nathan Ilten** Course Instructor Associate Professor of Mathematics
Date Approved:	**December 13, 2019**

Abstract

Certain symbolic matrices can be highly structured, with the possibility of having a block triangular form. Correctly identifying the block triangular form of such a matrix can greatly increase the efficiency of computing its determinant. Current techniques to compute the block diagonal form of a matrix rely on existing algorithms from graph theory which do not take into consideration key properties of the matrix itself. We developed a simple algorithm that computes the block triangular form of a matrix using basic linear algebra and graph theory techniques. We designed and implemented an algorithm that uses the Matrix Determinant Lemma to exploit the nature of block triangular matrices. Using this new algorithm, the computation of any determinant that has a block triangular form can be expedited with negligible overhead.

Keywords: Linear Algebra; Determinants; Matrix Determinant Lemma; Schwartz-Zippel Lemma; Algorithms; Dixon Resultants

Dedication

For my brothers,
Kassim, Yaseen, and Hakeem.

Acknowledgements

I would like to acknowledge Dr. Michael Monagan for providing his support throughout the entire research project. This project would not have been possible without his commitment towards developing whatever was needed to move forwards. Additionally, I would like to thank Dr. Nathan Ilten for taking the time to create a rewarding research course, and for giving us some insight as to what it is like to be a mathematician. I would also like to thank David Sun, who began this research project with me and someone who positively influenced many ideas throughout the project. Lastly, I would like to acknowledge NSERC and Simon Fraser University for giving me the special opportunity to participate in research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 History

From robotics [CLO15, chapter 3] to geodesy [PZAG08], to control theory [Pal13], systems of polynomial equations have the power to model complex problems. A solution of a system of polynomial equations might identify the optimal placement of an antenna on a satellite. It could help robots navigate through space more effectively, giving rise to machines more capable then ever before. These polynomial systems, and the methods for solving them, have helped realize countless modern advances across a variety of domains.

Elimination theory is concerned with developing methods and techniques for solving systems of polynomial equations. These techniques revolve around the idea that we can generate new polynomial equations from a given collection of polynomials that involve fewer variables. Under certain circumstances which will be discussed later on, the solution sets of these new polynomials give us a hint as to what the solution to the original system looks like.

Definition 1.1.1. [GCL92, chapter 9] A *solution* (or root) of a system of j polynomial equations in r variables

$$p_i(x_1, x_2, \ldots, x_r) = 0,\ 1 \leq i \leq j. \tag{1.1}$$

over a field k is an r-tuple $(\alpha_1, \alpha_2, \ldots, \alpha_r)$ from a suitably chosen extension field of k such that

$$p_i(\alpha_1, \alpha_2, \ldots, \alpha_r) = 0,\ 1 \leq i \leq j. \tag{1.2}$$

Given a large system of equations in many variables, it will most likely be difficult to see what the solution could be just by inspection. The general idea of elimination theory is to eliminate variables from a system of polynomial equations to produce new equations in a subset of the original variables. The advantage is that these new equations have fewer variables, which provides us with the opportuniyt to find a partial solution. Given that certain properties hold, these partial solutions can then be extended to full solutions as we have described them in Definition 1.1.1.

When considering linear equations, a solution can be obtained using the combination of Gaussian elimination and back substitution. Systems of equations that are row equivalent share the same solution set [Lay06]. Reducing the system to row echelon form generates new polynomials from the lower rows that have fewer variables. From this point of view, it is easier to see a potential solution or lack thereof.

Gaussian Elimination provides us with a method of systematically eliminating variables from a system of *linear* equations. What about systems of polynomials that contain non-linear equations?

Throughout the 19^{th} and 20^{th} century mathematicians such as Sylvester [Has07, chapter 5] and Macaulay [Mac03] improved our understanding of solutions sets of polynomial systems, and produced general methods for obtaining solution sets to multivariate non-linear systems. They developed methods to produce what are known as eliminants. These are objects, often polynomials, that would give them an indication of whether such a solution exists. These eliminants include the familiar quadratic discriminant $b^2 - 4ac$, which tells us whether the polynomial $ax^2 + bx + c$ has any real solutions, and whether it has a double root.

In 1965, Bruno Buchburger introduced his algorithm for computing a Gröbner basis [Buc76]. Once a Gröbner basis of an ideal defined by a collection of polynomials has been computed, one can more easily answer many questions related to the existence of solutions, and how to generate them. It is one of the main tools used in elimination theory today, as computer algebra systems have become powerful enough to handle relatively large systems. Buchburger's algorithm can be seen as a non-linear analogue of Gaussian elimination, so it is not hard to believe that it can help us uncover the solution to a system of polynomial equations [CLO15].

1.2 Resultants

We would like to preface this discussion on resultants by saying that it is not the purpose of this thesis to give a complete and rigorous treatment of resultant theory. We wish to provide a basic introduction to the ideas and techniques that we found helpful in computing resultants. This will provide some motivation as to why computing resultants is important.

We begin by providing the most general definition of a resultant. While this definition may seem abstract, its power lies in its generality. While we will not discuss all types of resultants, this definition has allowed mathematicians such as Sylvester [CLO15] and Macaulay [Mac03] to formulate resultants in different contexts.

Definition 1.2.1. [GKZ94, Chapter 12] The resultant of $k+1$ polynomials $f_0, f_1, \ldots, f_k$ in k variables is defined as an irreducible polynomial in the coefficients of $f_0, f_1, \ldots, f_k$, which *vanishes* whenever these polynomials have a common root.

In 1908, Dixon defined a sequence of steps that allows us to eliminate one variable from a system of two polynomial equation [Dix08]. One year later, he published a similar paper describing a procedure to eliminate two variables from three equations [Dix09]. Almost a century later in 1994, Kapur, Saxena, and Yang generalized Dixon's method for systems $\mathcal{F}$ of $n+1$ polynomial equations with n variables [KSY94]. Their idea, known as the KSY-variant of Dixon's method, eliminates n variables, leaving us with a polynomial in the coefficients of $\mathcal{F}$. This polynomial is known as the Dixon resultant. Dixon's method uses a matrix called Dixon's matrix, a matrix whose determinant produces a Dixon resultant. It is similar in many respects to other matrix-based resultant formulations such as the Bèzout's matrix [Cay57] and Sylvester's matrix. Because the Dixon's method relies on determinant computations, its time complexity is more manageable than that of Buchburger's algorithm. This is not to say computing symbolic determinants is always fast enough for practical applications. We require a plethora of tools and tricks to compute symbolic determinants efficiently, some of which we will see in chapters 3 and 4.

1.3 Structured matrices

It is possible to formulate many geometric problems as systems of polynomial equations. Computing the Dixon resultant for such systems can provide us with information about the solution to these system. Viewed in the correct context, the solution of such a system can give us information about the volume of tetrahedron [KSY94], or prove geometric theorems [Kap97],[CLO15, Chapter 6]. Although it has not yet been detailed in the literature, the Dixon matrix of systems of polynomial equations that represent geometric problem are often highly structured. This is most likely due to some underlying structure of the polynomials. These Dixon matrices frequently present themselves as row and column permutations of block triangular matrices. These matrices have what is known as a block triangular form. However, these matrices can be quite large, so determining if a matrix has a block triangular form is no trivial task. A large portion of our research went into developing an algorithm that is able to detect this structure within matrices.

If a matrix has a block triangular form, then many matrix computations become easier to perform. Most importantly for us, the determinant of a block triangular matrix is the product of the determinants of each sub-matrix along the main diagonal [Dym13]. Since each one of these sub-matrices is smaller, we can more efficiently compute the determinant of large matrices by splitting the problem into many smaller determinant computations. This is especially true if the matrix is a symbolic one, that is to say, it has polynomials in its entries. Dixon matrices are symbolic matrices that can have a block triangular form. Determinant computations can become infeasible if this block structure is not exploited. The challenge is that it is not trivial to compute a block triangular form, even if we know

a priori the matrix has one. Therefore it is crucial that we have an efficient method for detecting and computing block triangular forms in matrices.

Our new algorithm consists of two phases, with the first phase containing two key lemmas that allow us to perform inexpensive operations on matrices. The first phase utilizes the Matrix Determinant Lemma (MDL), which gives us the ability to update the determinant of a matrix after changing some of its entries without recomputing the entire determinant. The second phase uses the Schwartz-Zippel Lemma (SZL), which assures us that when we evaluate a symbolic matrix at random values, the image we obtain is an accurate one with high probability. We will go into greater depth as to what an accurate image means, but for now we will keep things simple and just say that this image will satisfies some desirable properties. The key idea is that we can look at the structure of the inverse of an image of a matrix (using MDL) to see which entries affect the determinant. If the matrix has a block triangular form, then we may set equal to zero all entries not within the blocks of the original matrix to simplify it. From there it becomes trivial for the second phase to detect the hidden block structure of the original matrix.

This thesis begins with a general discussion about Dixon's method, and then it spends the majority of the time on determinant computation techniques. Chapter 2 begins with an introduction to the notation and vocabulary that is used throughout the thesis. We then present data that motivates the use of Dixon's method, as well as an outline of Dixon's method. This section concludes with a basic example to solidify our understanding before moving on to more specific topics. Chapter 3 is entirely devoted to determinant computation techniques. Given that the Dixon resultant is the determinant of a symbolic matrix, it is crucial that we are well equipped to compute determinants efficiently. We describe a variety of determinant methods, and provide data to compare and contrast them. Chapter 4 presents our main contribution to the research area. We present a new method for testing whether a matrix has a block diagonal form, along with pseudo-code and a proof. In Chapter 5 we display a table of data about Dixon matrices that was collected during our time experimenting. This highlights the value in using Dixon's method, especially for larger polynomial systems. We conclude with chapter 6, where we communicate open questions, summarize the thesis, and provide an assortment of references for further reading material.

Chapter 2

Tools, Methods, & Vocabulary

For definitions related to matrices, we follow the conventions seen in *Linear Algebra in Action* [Dym13]. For notation related to polynomials we refer the reader to *Introduction to algebraic geometry* [Has07].

2.1 Notation

We use k and $\mathbb{F}$ to denote fields, and R will denote a ring. When we are referring to matrices we will always use $\mathbb{F}$, while we will use k when we speak about polynomials and systems of polynomials.

We use $\mathcal{F}$ and $\mathcal{C}$ to refer to a system of polynomial equations and a collection of polynomials respectively. Depending on the system, it might have indeterminates which we do not wish to consider as variables. We call these indeterminates parameters of the system, and they belong to the coefficient ring. We denote the parameters of a system with using $\mathbf{c} = \{a, b, \ldots\}$, where $\mathbf{c}$ is a finite list of symbols. Hence the parameters belong to the coefficient ring $k[\mathbf{c}]$. For the entire thesis, we only consider systems $\mathcal{F}$ which have a finite number of equations, variables, and parameters.

We can view multivariate polynomials as univariate polynomials whose coefficients are polynomials in the other variables. Given a ring R, a polynomial $f \in R[x_1, \ldots, x_n]$ can be viewed recursively as a polynomial in $R[x_2, \ldots, x_n][x_1]$. An example illustrates this idea nicely.

Example 2.1.1. [GCL92, chapter 2] The polynomial $a(x, y) = 5x^3y^2 - x^2y^4 - 3x^2y^2 + 7xy^2 + 2xy - 2x + 4y^4 + 5 \in \mathbb{Z}[x, y]$ may be viewed as a polynomial in the ring $\mathbb{Z}[y][x]$:

$$a(x, y) = (5y^2)x^3 - (y^4 + 3y^2)x^2 + (7y^2 + 2y - 2)x + (4y^4 + 5). \tag{2.1}$$

We can also consider $a(x, y)$ as a polynomial in the ring $\mathbb{Z}[x][y]$:

$$a(x, y) = (-x^2 + 4)y^4 + (5x^3 - 3x^2 + 7x)y^2 + (2x)y(-2x + 5). \tag{2.2}$$

This recursive view of polynomials is important when solving problems with Dixon's method, as it gives us the freedom to move variables and parameters in and out of the coefficient ring depending on how we want to view the problem.

The set of integers modulo p will be represented as $\mathbb{F}_p$. The list $X = [x_1, x_2, \ldots, x_n]$ is an ordered list of variables. The list $\bar{X} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n]$ is also an ordered list of variables where $X \cap \bar{X} = \emptyset$.

An n-tuple $< a_1, a_2, \ldots, a_n >$ represents a column vector, and $< a_1, a_2, \ldots, a_n >^T$ is its corresponding row vector. The narrow angle brackets are reserved for ideals; $\langle \mathcal{C} \rangle$ is the ideal generated by the collection of polynomials $\mathcal{C}$.

If A is any matrix, then a_{ij} will represent the entry in row i, column j of A. If instead we use A_{ij}, we are referring to some sub-matrix in A whose dimensions will be made clear in each instance. The order of a square matrix is the number of rows it has. A matrix is sparse if at least 50% of its entries are equal to 0.

An arbitrary Dixon matrix will be denoted by Θ, while a sub-matrix of maximal rank in Θ is denoted by Θ'. Note that for certain Dixon matrices Θ is full rank, hence $\Theta = \Theta'$. The determinant of Θ' is known as Dixon's resultant, and we represent it with $\mathcal{DR}$.

We make a distinction between the concept of a particular algorithm, and the implementation of said algorithm. If we write 'Gaussian elimination', we are referring to the abstract notion of the algorithm. However if we write `GaussianElimination`, we are referring to a specific implementation of Gaussian elimination, usually writte in Maple.

We will present various tables of data throughout the thesis. In order to save space, we will abbreviate many words and/or sentences. The number of terms in a polynomial f is denoted nops(f) or $\#f$. We shorten Gaussian elimination with G.E. where convenient.

2.2 Motivation for Dixon's Method

Before we explain the intricacies of Dixon's method, we would like to provide some motivation as to why we prefer this method over other techniques. Buchburger's algorithm for computing a Gröbner basis is used to compute elimination ideals. Since resultants lie in certain elimination ideals, this algorithm can also be used to produce a resultant for a systems of multivariate polynomial equations.

Wu's method of characteristic set is another method that produces a resultant of a system of equations. It takes a collection of polynomials on input, and returns a collection of polynomials which help to produce solutions to the original system of polynomial equations.

We ran these two algorithms, and Dixon's method, on systems of randomly generated quadratic and cubic polynomials. We timed each one, and present a table for direct comparison.

The timings for Gröbner Basis were done using Maple's `Groebner[Basis]` command, while the timings for triangular sets were done using Maple's `Triangularize` command.

As we will see shortly, Dixon's method requires the computation of determinants, in particular Dixon's matrix. For the timings in Table 2.1 and 2.2, we used a technique known as Dixon-EDF, which is essentially a version of fraction-free Gaussian elimination [Lew08]. It was was designed specifically for taking determinants of Dixon matrices. A detailed explanation of the technique with examples and pseudo-code can be found in [Lew17].

Table 2.1: Comparison of elimination techniques: part 1

Number of equations	2	3	4	5	6	7	8	9	10
Gröbner Basis	0.016	0.000	0.015	0.016	0.032	0.016	15.36	99.98	147.6
Triangular sets	0.000	0.015	0.032	0.062	0.047	0.078	897.672	$> 10^3$	$> 10^3$
Dixon's method	0.015	0.032	3.967	44.227	0.015	0.015	1.610	3.016	65.640

All polynomials are quadratic. Time recorded in seconds

Table 2.2: Comparison of elimination techniques: part 2

Number of equations	2	3	4	5	6	7	8	9	10
Gröbner Basis	0.016	0.047	0.063	0.125	0.102	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$
Triangular sets	0.000	0.032	0.062	906.6	0.437	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$
Dixon's method	0.000	0.000	0.391	2.343	16.079	158.640	$> 10^3$	$> 10^3$	$> 10^3$

All polynomials are cubic. Time recorded in seconds.

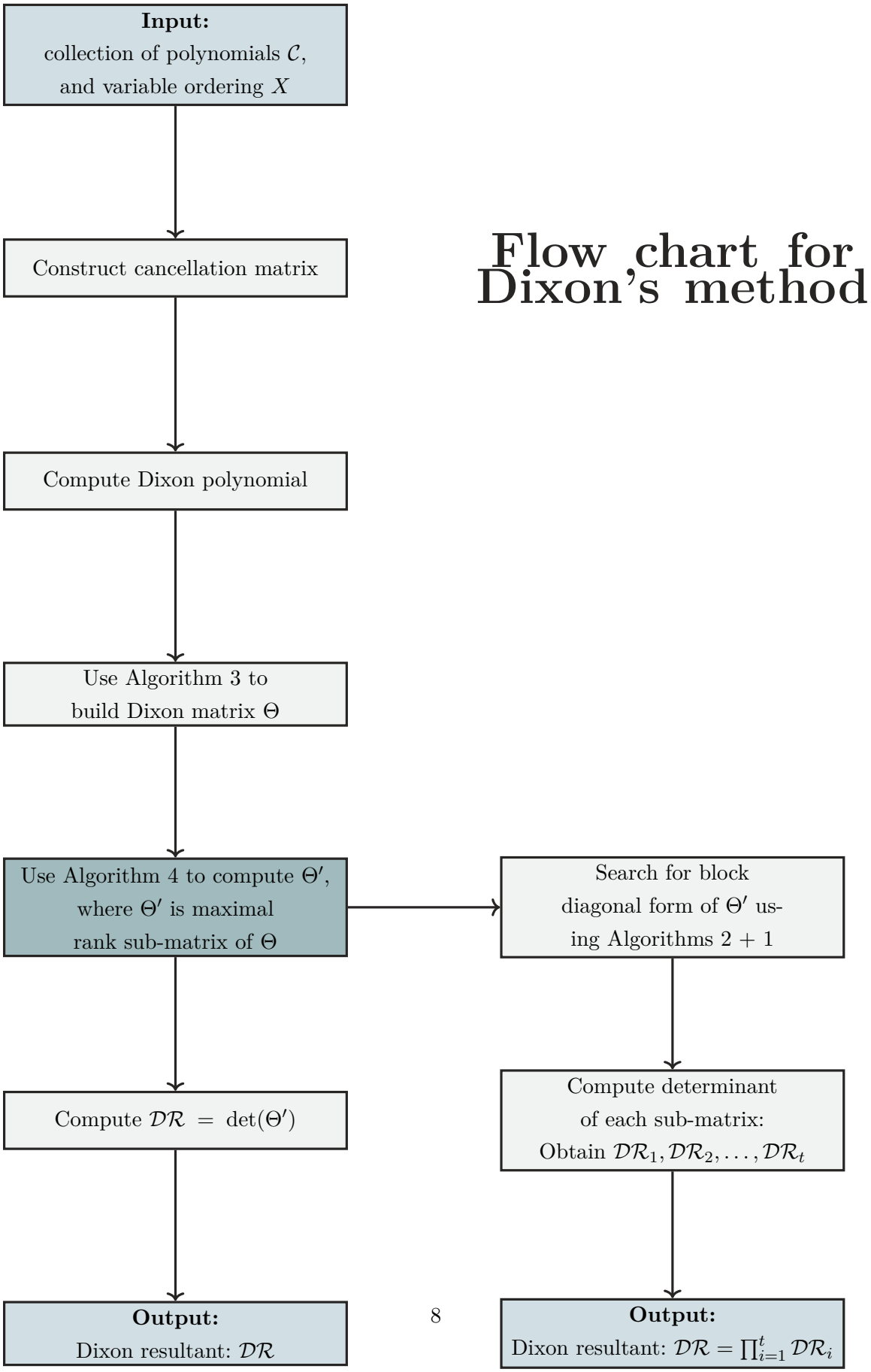
The main difference between Table 2.1 and Table 2.2 is that Table 2.1 tests quadratic polynomials, while Table 2.2 tests cubic polynomials.

2.3 Dixon's method

Dixon's method is a multi-stage process that requires the execution of a few different procedures. We outline the steps in Dixon's method and the KSY-variant of Dixon's method in a visual manner to prepare the reader for what is ahead. The KSY-variant begins at the fork (darkest blue) in the diagram on the next page, and extends rightwards and then downwards. On the other hand, Dixon's original method continues directly downwards, but can only do so under certain circumstances which will be discussed later in the section.

It also acts as reference if the reader should choose to implement this method. Pseudo-code for most of the steps involved can be found the appendix, and we encourage the reader to read it at least once as it illustrates some aspects of this method nicely. With that begin said, the content of this section can be understood without reading any code.

Flow chart for Dixon's method



Arriving at the Dixon resultant involves three major steps. Throughout our research, we used the Kapur-Saxena-Yang variant of the Dixon resultant [Lew96]. The first step is to construct what is known as the cancellation matrix [CK02]. The determinant of the cancellation matrix is the Dixon polynomial. The Dixon polynomial acts as an intermediary stage between the cancellation matrix and the Dixon matrix. To produce the Dixon matrix from the Dixon polynomial, one needs to rewrite the Dixon polynomial as a special vector-matrix-vector product. The matrix produced in this product is called the Dixon matrix.

Beginning with a system of $n+1$ equations $\mathcal{F} = \{f_0, \ldots, f_n\}$, we must select n variables $X = [x_1, \ldots, x_n]$ to eliminate. We call the list variables in the list X the set of original variables. The order of these variables can affect the size and degree of of the Dixon polynomial. For this reason we denote the Dixon polynomial of a given system $\mathcal{F}$ as $\Delta(\mathcal{F}, X)$, to emphasize that the Dixon polynomial depends on both $\mathcal{F}$ and X. Literature on selecting the optimal order to eliminate the variables does exist but will not be the focus of this thesis. See [CK02] for more information. Once a variable ordering is chosen, we can compute the Dixon polynomial using the following formula:

Definition 2.3.1. [CK02] Let $\pi(\mathbf{x}^\alpha) = \bar{x}_1^{\alpha_1} \cdots \bar{x}_i^{\alpha_i} x_{i+1}^{\alpha+1} \cdots x_d^{\alpha_d}$, for $i \in \{0, \ldots, d\}$, and $\bar{x}_i$'s are new variables; $\pi_0(x^\alpha) = x^\alpha$. We extend π to polynomials as: $\pi_i(f(x_1, x_2, \ldots, x_d)) = f(\bar{x}_1, \ldots, \bar{x}_i, x_{i+1}, \ldots, x_d)$. We define the Dixon polynomial as:

$$\Delta(\mathcal{F}, X) = \prod_{i=1}^{n} \frac{1}{x_i - \bar{x_i}} \begin{vmatrix} \pi_0(f_0) & \pi_0(f_1) & \pi_0(f_2) & \ldots & \pi_0(f_n) \\ \pi_1(f_0) & \pi_1(f_1) & \pi_1(f_2) & \ldots & \pi_1(f_n) \\ \vdots & \vdots & \vdots & & \vdots \\ \pi_n(f_0) & \pi_n(f_1) & \pi_n(f_2) & \ldots & \pi_n(f_n) \end{vmatrix} \tag{2.3}$$

The matrix in definition 2.3.1 is the cancellation matrix of $\mathcal{F}$ with respect to a particular variable elimination order X. Hence we denote this matrix with $C_{\mathcal{F},X}$. Since for all $1 \leq i \leq n$, $(x_i - \bar{x}_i)$ is a zero of $C_{\mathcal{F},X}$, $\prod_{i=1}^{n}(x_i - \bar{x}_i)$ divides $C_{\mathcal{F},X}$ [Kap97]. Hence $\Delta(\mathcal{F}, X)$ is a polynomial, which we call the Dixon polynomial of $\mathcal{F}$ with respect to a particular elimination variable ordering of X. The Dixon polynomial can also tell us the dimension of Dixon's matrix, which will be produced in the next stage.

Definition 2.3.2. [Kap97] Let $\mathbf{V}$ be a row vector of all monomials in X which appear in $\Delta(\mathcal{F}, X)$, when $\Delta(\mathcal{F}, X)$ is viewed as a polynomial in X. Similarly, Let $\mathbf{W}$ be a column vector of all monomials in $\bar{X}$ which appear in $\Delta(\mathcal{F}, X)$, when $\Delta(\mathcal{F}, X)$ is viewed as a polynomial in $\bar{X}$. The **Dixon matrix**, Θ, of $\mathcal{F}$ is defined to be the matrix for which $\Delta(\mathcal{F}, X) = \mathbf{V}\Theta\mathbf{W}$. The entries of the Dixon matrix for $\mathcal{F}$ are polynomials in the coefficients of $\mathcal{F}$. Dixon's matrix is an $\prod_{i=1}^{n}(e_i+1) \times \prod_{i=1}^{n}(d_i+1)$ matrix, where $e_i = \deg(\Delta(\mathcal{F}, X), x_i)$, and $d_i = \deg(\Delta(\mathcal{F}, X), \bar{x}_i)$.

The **Dixon resultant** is the determinant of Dixon's matrix. Technically speaking, the Dixon resultant is not a resultant as we defined it in Definition 1.2.1. It is a non-zero multiple of the resultant of a given system.

Let $\mathcal{C}$ be a collection of polynomials in the variables X, with parameters $\mathbf{c}$. Let I be the ideal generated by $\mathcal{C}$ in the ring $k[X, \mathbf{c}]$, and let $J = I \cap k[\mathbf{c}]$. Then J is the elimination ideal of I over $\mathbf{c}$, and all polynomials $h \in J$ are known as the projection operators of $\mathcal{C}$ with respect to X [Kap97]. The Dixon resultant is a member of J, or a projection operator of $\mathcal{C}$. It can be shown that the resultant of $\mathcal{C}$ also belongs to J, and divides all projection operators. Hence every Dixon resultant contains as one of its factors the resultant of $\mathcal{C}$ with respect to a particular set of variables X [Kap97]. All factors of a Dixon resultant that are not the resultant are called extraneous factors [Kap97]. A Dixon resultant that does not contain any extraneous factors is called exact [Kap97]. Identifying extraneous factors in Dixon resultants is not trivial, especially when the Dixon resultants are large. Further information about extraneous factors can be found in [KS97].

As was discussed by Dixon himself, his method is only defined if the system of polynomials equations is generic and n-degree [Dix08], [Dix09].

Definition 2.3.3. [Kap97] A collection of polynomials $\mathcal{C} = \{f_1, \ldots, f_{n+1}\}$ is called generic n-degree if there exists non-negative integers $d_1, \ldots, d_n$ such that,

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

where each coefficient $a_{j,i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$ is a distinct indeterminate. The n-tuple $(d_1, \ldots, d_n)$ is known as their n-degree.

This limits us to a small class of systems. One reason is because if this condition does not hold, then using the formula in Definition 2.3.2 it is not hard to show that Dixon's matrix will not be square. Dixon's original method only allows us to take determinants of square matrices, so if Θ is not square we are stuck. Additionally, Θ could be square but rank deficient, which means its determinant is identical to zero [Dym13]. This gives us no information about the system in question.

The Kapur-Saxena-Yang variant [KS95] of Dixon's method gives us the necessary tools to sidestep both of these problems. Once the Dixon matrix Θ of $\mathcal{C}$ is constructed, we search for a sub-matrix of maximal rank Θ' within Θ. Not only is Θ' guaranteed to be square [Lay06], but given that certain conditions are satisfied, $\det(\Theta')$ is a non-zero projection operator of $\mathcal{C}$. This is stated nicely in the theorem below. An algorithm to search for a sub-matrix of maximal rank is provided in the appendix (Algorithm 4).

Theorem 2.3.4 (Kapur-Saxena-Yang)**.** *[Lew96] Let $\mathcal{DR}$ be the determinant of any maximal rank sub-matrix of Θ. Then, if a certain condition holds, $\mathcal{DR} = 0$ is necessary for the existence of a common zero.*

In short, this theorem is saying that if a system $\mathcal{F}$ generates a rank deficient Dixon matrix Θ, including the possibility of Θ begin non-square, then the determinant of any maximal rank sub-matrix of Θ will be a projection operator of $\mathcal{F}$. Theorem 2.3.4 provides us with the means to compute the Dixon resultant of arbitrary systems of polynomial equations.

So once we produce a Dixon resultant, what can we do with it? We mentioned early in the introduction that under certain circumstances the resultant of a polynomial system gives us information about what the solution to the entire system looks like. We formalize that notion with a central theorem.

Theorem 2.3.5 (Fundamental Theorem of Resultants)**.** *[GCL92, chapter 9] Let $\bar{k}$ be an algebraically closed field, and let*

$$f = \sum_{i=0}^{m} a_i(x_2, \ldots, x_r)x_1^i, \quad g = \sum_{i=0}^{n} b_i(x_2, \ldots, x_r)x_1^i \tag{2.4}$$

be elements of $\bar{k}[x_1, \ldots, x_r]$ of positive degrees in x_1. Then if $(\alpha_1, \ldots, \alpha_r)$ is a common zero of f and g, their resultant with respect to x_1 satisfies

$$\mathrm{Res}(f, g, x_1)(\alpha_2, \ldots, \alpha_r) = 0. \tag{2.5}$$

Conversely, if the above resultant vanishes at $(\alpha_2, \ldots, \alpha_r)$, then at least one of the following holds:

1. $a_m(\alpha_2, \ldots, \alpha_r) = \cdots = a_0(\alpha_2, \ldots, \alpha_r)$
2. $b_n(\alpha_2, \ldots, \alpha_r) = \cdots = b_0(\alpha_2, \ldots, \alpha_r)$
3. $a_m(\alpha_2, \ldots, \alpha_r) = b_n(\alpha_2, \ldots, \alpha_r) = 0$
4. $\exists \alpha_1 \in \bar{x}$ *such that* $(\alpha_1, \alpha_2, \ldots, \alpha_r)$ *is a common zero of* f *and* g.

This theorem provides us with a way to use resultants to construct partial solutions to systems of polynomial equations, which under certain circumstances lead to the full solution. If the first three conditions of Theorem 2.3.5 do not hold, then Condition 4 guarantees us that we are able to extend a partial solution one more coordinate position. Hence producing a resultant can be thought of as simplifying the problem of finding the solution to many polynomial equations at once, to finding the solution of a single polynomial in fewer variables.

Before we end this chapter, we want to do a small example with Dixon's method so that we become comfortable with the sequence of procedures. The flow chart was presented on page 8 if it is needed for reference.

2.4 The area of a triangle

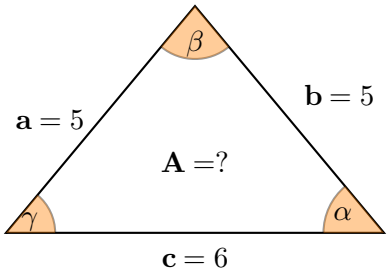


Figure 2.1: Heron's problem [not drawn to scale]

Our goal is to determine the area of a triangle without computing any of its angles. We will assume that the side lengths we are given are in fact valid side lengths of a triangle. One can easily check this by verifying that the follow three inequalities hold:

$$a + b > c, \qquad a + c > b, \qquad b + c > a$$

It is important to check this since this problem does not make sense if there does not exist a triangle with the given side lengths to begin with. These inequalities tell us for which values we are allowed to specialize Dixon's resultant; essentially only values which produce a valid triangle.

We begin by setting up a system of polynomial equations which correspond to the geometric nature of this problem. We will not detail the process of constructing a system of polynomial equations from a geometric problem, however [CLO15, Chapter 6] has an introduction to geometric theorem proving which provides a concrete example.

Example 2.4.1. Let $\mathcal{F} = \{x^2 + y^2 - c^2, (x-a)^2 + y^2 - b^2, 2A - ay\}$. Let $X = [x, y]$ be the variables which we wish to eliminate, our new variables $\bar{X} = [\bar{x}_1, \bar{x}_2]$ and the parameters $\mathbf{c} = \{A, a, b, c\}$. Note that both X and $\bar{X}$ are ordered lists, and correspond to the order we are eliminating the variables. Simply put, the polynomials in $\mathcal{F}$ provide an algebraic description of the problem we are trying to solve.
Using the formula in 2.3.1 we get:

$$\Delta = \Delta(\mathcal{F}, X) = \prod_{i=1}^{2} \frac{1}{x_i - \bar{x}_i} \begin{vmatrix} -c^2 + x^2 + y^2 & (x-a)^2 + y^2 - b^2 & -ay + 2A \\ -y - \bar{x}_1 & -y - \alpha_1 & a \\ -x - \bar{x}_2 & -x + 2a - \bar{x}_2 & 0 \end{vmatrix} \tag{2.6}$$

Computing the determinant in equation 2.6, and dividing by $\prod_{i=1}^{2}(x_i - \bar{x}_i)$ of gives us:

$$\Delta = \left(-a^3 + 2\,a^2\alpha_2 + ab^2 - ac^2\right)x + \left(2\,a^2\alpha_1 - 4\,Aa\right)y - a^3\alpha_2 + 2\,a^2c^2 + ab^2\alpha_2 - ac^2\alpha_2 - 4\,Aa\alpha_1$$

Now we can use Algorithm 3, or any equivalent method, to produce the Dixon matrix for Δ. In this particular example, $\mathbf{V} = [x, y, 1]$, $\mathbf{W} = [\bar{x}_1, \bar{x}_2, 1]^T$, and

$$\Theta = \begin{bmatrix} 0 & 2\,a^2 & -a^3 + ab^2 - ac^2 \\ 2\,a^2 & 0 & -4\,Aa \\ -4\,Aa & -a^3 + ab^2 - ac^2 & 2\,a^2c^2 \end{bmatrix} \tag{2.7}$$

In this example Θ is already a matrix of full rank so we skip the step involving finding a sub-matrix of maximal rank. We then try and compute a block diagonal form of Θ, however it does not have one so we continue with Θ. Now we use any efficient method to compute the determinant of Θ

$$\det(\Theta) = \mathcal{DR} = 2\,a^4\left(a^4 - 2\,a^2b^2 - 2\,a^2c^2 + b^4 - 2\,b^2c^2 + c^4 + 16\,A^2\right)$$

Recalling that the system of polynomials has a solution if and only if the resultant vanishes, we want to set this Dixon resultant equal to zero. Additionally, we may now specialize the parameters found in $\mathcal{DR}$ with the side lengths we are given; this amounts to evaluating $\mathcal{DR}$ at $E = \{a = 5, b = 5, c = 6\}$.

$$\mathcal{DR}|_E = 20000A^2 - 2880000 = 0$$

We could use numerical methods to solve this quadratic, however it is small enough that we can just factor it. Using Maple's `factor` command we obtain:

$$20000A^2 - 2880000 = 20000(A - 12)(A + 12) = 0$$

We discard the negative solution, as the area of a valid non-degenerate triangle will always be positive. Hence the area of the triangle in Figure 2.1 solved using a Dixon resultant is:

$$A = 12$$

While this example is small and there are many other ways to calculate the area of a triangle, this procedure extends to higher dimensional objects in a very natural way [KSY94]. For this reason computing a resultant can be used for a wide array of geometric problems [Kap97], [LC06]. In general, the more complicated the problem, the more polynomial equations we will have in a system. As seen in the previous example, we need to take deter-

minants of matrices in order to obtain Dixon's resultant. Furthermore, the matrices we are working are over a polynomial ring. We need efficient methods to compute the determinant of symbolic matrices if we hope to produce Dixon resultants for large systems. This leads us into Chapter 3, as we explore determinant computation.

Chapter 3

Determinants

3.1 An overview of methods

Let us start with Gaussian elimination which is taught in a first linear algebra course as a method for calculating the determinant of a matrix. Gaussian elimination needs divisions, and thus requires a field. For an $n \times n$ matrix it does $\mathcal{O}(n^3)$ field operations. When applied to a symbolic matrix, Gaussian elimination produces rational functions which grow in size/degree with each elimination step. These expressions require the computation of polynomial gcds to simplify. These operations can be expensive even for small matrices.

In this chapter, we consider two other methods that compute the determinant of a matrix using only the ring operations addition, subtraction, and multiplication. The cost of the methods are shown below for convenient reference.

Algorithm	Number of ring/field operations
Gaussian Elimination	$\mathcal{O}(n^3)$
Berkowitz	$\mathcal{O}(n^4)$
Minor Expansion	$\mathcal{O}(n2^n)$

The number of arithmetic operations in the table above do not account for the cost of individual arithmetic operations, which may vary greatly. We will see in the following sections that on symbolic matrices, the method of Minor expansion by Gentlemen & Johnson is often the best choice despite the exponential number of operations it requires.

3.2 Minor expansion by Gentleman & Johnson

In this section we discuss the method of Minor expansion for computing determinants, which also goes by the names cofactor expansion or Laplace expansion. In 1973 paper [GJ73], two researchers Gentleman and Johnson noticed something peculiar about minor expansion. They noticed that for matrices of order 4 and larger, minor expansion computes certain expressions multiple times. We use a short example with a matrix over the integers to

illustrate this concept in a simple manner, however the same idea applies to matrices with polynomial entries.

Example 3.2.1.

$$A = \begin{bmatrix} 9 & 3 & 4 & 2 \\ 2 & 4 & 6 & 8 \\ 2 & 3 & 5 & 7 \\ 9 & 8 & 2 & 1 \end{bmatrix} \tag{3.1}$$

We will expand across the first row of A.

$$|A| = 9\begin{vmatrix} 4 & 6 & 8 \\ 3 & \mathbf{5} & \mathbf{7} \\ 8 & \mathbf{2} & \mathbf{1} \end{vmatrix} - 3\begin{vmatrix} 2 & 6 & 8 \\ 2 & \mathbf{5} & \mathbf{7} \\ 9 & \mathbf{2} & \mathbf{1} \end{vmatrix} + 4\begin{vmatrix} 2 & 4 & 8 \\ 2 & 3 & 7 \\ 9 & 8 & 1 \end{vmatrix} - 2\begin{vmatrix} 2 & 4 & 6 \\ 2 & 3 & 5 \\ 9 & 8 & 2 \end{vmatrix} \tag{3.2}$$

It is enough to expanded the first two terms from Equation 3.2 to see that the same sub-determinants, highlighted in boldface, get computed more than once.

In order to see where they could avoid performing redundant computations, they took a bottom-up approach. We refer to rows, although the same analysis can be done with columns. For an $n \times n$ matrix, they reasoned that any generic determinant calculation must compute all 1×1 determinants in the bottom row. It then must compute all 2×2 sub-determinants in the bottom two rows. However, once it reaches the 3×3 determinants in the bottom three rows, it already has all 2×2 determinants computed from the previous step. So we only have to multiply the non-zero entries of the third rows with the appropriate sub-determinants in the bottom two rows. This process climbs all the up the matrix until it reach the first row of the matrix, where there is only one $n \times n$ determinant to compute.

In short, the method of Minor expansion by Gentleman & Johnson computes all $k \times k$ sub-determinants from the bottom k rows of a matrix and stores them. When it is time to compute all $k+1 \times k+1$ sub-determinants, it already has all $k \times k$ sub-determinants stored from the previous step. So instead of recomputing these, we just use them to compute the $k+1 \times k+1$ sub-determinants.

In Example 3.2.1 above, the standard minor expansion algorithm requires a total of 72 multiplications; with the modification from Gentlemen & Johnson that number drops to 28. When considering the number of multiplications done, the improved algorithm still requires $\mathcal{O}(n(2^{n-1}-1))$ ring operations [GJ73]. However, at the end of this chapter we will show that this method turns out to be one of the most efficient methods of computing symbolic determinants.

3.3 Characteristic polynomials by Berkowitz

The Samuelson-Berkowitz algorithm is an efficient method of computing the characteristic polynomial of an $n \times n$ matrix. It is particularly useful in our case because the Samuelson-Berkowitz algorithm well behaved when the entries of the matrix belong to any unital commutative ring without zero divisors. Given an $n \times n$ matrix, it recursively partitions the matrix into principal sub-matrices until it reaches the 1×1 sub-matrix in the upper left hand corner. It then assembles the coefficients of the characteristic polynomial by taking successively larger vector-matrix products. The Samuelson-Berkowitz algorithm stands as one of the most efficient ways to compute the characteristic polynomial of an $n \times n$ matrix over a ring. As was mentioned early in this chapter, the algorithm requires $\mathcal{O}(n^4)$ ring operations.

So how does computing the characteristic polynomial relate to determinants? To answer this question, we need to look at the definition of the characteristic polynomial.

Definition 3.3.1. Consider an $n \times n$ matrix $A \in \mathbb{F}^{n \times n}$. The characteristic polynomial of A, denoted by $p_A(x)$, is the polynomial defined by

$$\sum_{i=0}^{n} a_i x^i = p_A(x) = \det(xI - A).$$

where I denotes the $n \times n$ identity matrix.

The characteristic polynomial $p_A(x)$ is a polynomial in x, with coefficients from the domain which the matrix is defined over. In addition, definition 3.3.1 will produce a monic polynomial. The eigenvalues of A are the roots of $p_A(x)$, and so we may write:

$$p_A(x) = (x - \lambda_1)(x - \lambda_2) \cdots (x - \lambda_n) = \prod_{i=1}^{n} (x - \lambda_i). \tag{3.3}$$

We can easily show that the constant term in the expansion of the characteristic polynomial of A is in fact the determinant of A. Consider what would happen it we set $x = 0$ in Equation 3.3.

$$a_0 = \prod_{i=1}^{n} \lambda_i = p_A(0) = \det(0I - A) = \det(-A) = -\det(A).$$

It follows that the constant term of p_A is, up to a unit, the determinant of the A. To summarize, the Samuelson-Berkowitz algorithm for computing the characteristic polynomial of a matrix acts as a vehicle for producing determinants.

3.4 A comparison of methods

We now directly compare the determinant methods discussed throughout this section. All testing was done using Maple 2018, on a desktop equipped with an Intel Core i5-6699 CPU @ 3.30 GHz, and 16 GB of RAM. Our method was to run a variety of determinant computation algorithms on random symbolic matrices of different orders. More precisely, for matrices of order $2, 3, \ldots, 10$, we generated 5 random matrices and ran each algorithm on them, and took the average time in seconds. We also recorded the average number (approximated to 3 decimal places) of terms for the determinants of each order, placing these numbers in the column nops(det(A)). The random numerical matrices were produced with Maple's `RandomMatrix` command, where a_{ij} were chosen at random from the integer range $[-99, 99]$.

Table 3.1: Numerical determinant computation methods

Order	Minor Expansion	Berkowitz	Gaussian Elimination
12	0.078	0	0
14	0.313	0.015	0
16	1.422	0	0.016
18	6.328	0.016	0.015
20	40.110	0.016	0
30	> 1000	0.109	0.031
50	> 1000	0.672	0.156
70	> 1000	2.188	0.516
90	> 1000	6.437	1.203
110	> 1000	14.547	2.484
130	> 1000	29.063	5.000
150	> 1000	53.203	8.266

Data is from a single random matrix over the integers.
nops = number of terms.

In the next experiment, we wanted to know how effective the same determinant computation algorithms are when used on symbolic matrices. So that no algorithm has an unfair advantage due to some undetected structure of a matrix, we opted for random symbolic matrices of varying orders. For the sake of clarity and reproducibility, we present the Maple code in Figure 3.1 that was used to generate these random symbolic matrices.

The program in Figure 3.1 takes as its inputs a number, and a finite list of variables. First it generates an empty matrix of appropriate size. It then fills every entry with a random polynomials in the input variables. These polynomials are of degree 5 with exactly five terms, and all coefficients are 1.

Random Symbolic Matrix

```
RanSymbolicMtx := proc(order::integer, vars::{list,set})
local A, i, j;
A := Matrix(order);
for i from 1 to order do
    for j from 1 to order do
        A[i,j] := randpoly(vars,terms=5,degree=5,coeffs=proc() 1 end);
    od;
od;
return A;
end:
```

Figure 3.1: Maple code for generating random symbolic matrices

Table 3.2: Symbolic determinant computation methods

Order	Minor Expansion	Berkowitz	Gaussian Elimination	nops(det(A))
5	0.003	0.019	0.262	2376.4
6	0.021	0.116	1.400	4540.4
7	0.078	0.468	5.250	7609.8
8	0.303	1.153	15.053	11609
9	0.922	3.343	34.887	16399
10	2.768	21.593	83.021	22780

Data is the average of 5 tests on random symbolic matrices.
nops = number of terms.

The key message to take from this experiment is that determinant methods which are effective for numerical computations can perform poorly when the matrix is a symbolic one. A runtime analysis for polynomial matrices is presented in [GJ73].

In the next section we explore a method that improves determinant computation of symbolic matrices.

Chapter 4

Algorithms for revealing block structure

This chapter contains information pertaining to the algorithm we developed, and constitutes our main contribution.

4.1 Divide and Conquer

Divide and Conquer is an algorithm design paradigm that revolves around breaking down a given problem into at least two smaller sub-problems. In problems well suited to this paradigm, the smaller sub-problems are easier to solve. We will not have a comprehensive discussion about which algorithms are well suited for Divide and Conquer, however there is one application in which using the paradigm of Divide and Conquer pays dividends.

We will define block matrices in general, however throughout this section we will only make reference to either block diagonal matrices or block diagonal triangular matrices. The determinant properties we are interested in apply to both upper and lower block triangular matrices.

Definition 4.1.1. [Dym13, Chapter 1] A matrix $A \in \mathbb{F}^{n\times n}$ with block decomposition

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & & \vdots \\ A_{k1} & \cdots & A_{kk} \end{bmatrix}$$

where $A_{ij} \in \mathbb{F}^{p_i \times q_j}$ for $i, j = 1, \ldots, k$ and $p_1 + \cdots + p_k = q_1 + \cdots + q_k = n$ is said to be:

- **upper block triangular** if $p_i = q_i$ for $i = 1, \ldots, k$ and $A_{ij} = O$ for $i > j$.
- **lower block triangular** if $p_i = q_i$ for $i = 1, \ldots, k$ and $A_{ij} = O$ for $i < j$.
- **block triangular** if it is either upper block triangular or lower block triangular.

- **block diagonal** if $p_i = q_i$ for $i = 1, \ldots, k$ and $A_{ij} = O$ for $i \neq j$.

Note that the blocks A_{ii} in a block triangular decomposition need not be triangular.

Certain Dixon matrices Θ' have a block triangular form. By block triangular form, we mean that there exists some permutation of the rows and columns of the matrix such that the matrix now satisfies one of the statements in Definition 4.1.1. We consider block diagonal matrices a special subset of block triangular matrices. Knowing if a matrix has a block triangular form is important because we know that the determinant of such a matrix is the product of the determinants of the square sub-matrices along its main diagonal [Dym13, Chapter 5]. Using the notation for Dixon matrices, rather then computing $\det(\Theta')$, we employ a Divide and Conquer strategy, and compute the determinant of each sub-matrix along the main diagonal. Even better, the computation of the determinant of each sub-matrix can be done in parallel, since the computations are now independent of one another. This can provide significant savings in computational resources as the matrices become large. We do not have necessary and sufficient conditions as to which systems of polynomial equations produce Dixon matrices that have a block triangular form.

Before we give an explanation of the algorithm, we would like to justify the claim that computing a block diagonal form using our algorithm does actually result in faster determinant computation. Figure 4.1 shows the expected speed-up for computing determinants when computing the determinants of the blocks as opposed to the entire matrix at once. We have plotted in number of blocks on the x-axis, the order of the matrix on the y-axis, and the speed-up achieved on the vertical axis.

In the following two sections we will outline our method. The algorithms unfolds in two phases, with the possibility of running the second phase without the first. For this reason we introduction the second phase of the algorithm before the first. This is also consistent with how we actually developed the sub-routines during the course of our research. Phase 1 is carried out by Algorithm 2, and we will refer to this algorithm as `Clarify`. Phase 2 is carried out by Algorithm 1, and we refer to this algorithm as `Link`.

4.2 Phase 2: Link

The structure of a graph derived from a matrix can tell us many things about the matrix in question. With this in mind, we took inspiration for phase 2 of our new algorithm from [PF90]. In [PF90], they show a technique for computing the block diagonal form of a rectangular matrix. Their method first creates a bipartite graph. Then it seeks to construct a maximal matching. Since our ultimate goal is to take the determinant of all the matrices we work with, we only needed an algorithm for square matrices.

We create a bipartite graph that is based on the locations of the non-zero entries in each row and column. To begin, let A be a $k \times k$ matrix of full rank where a_{ij} represents

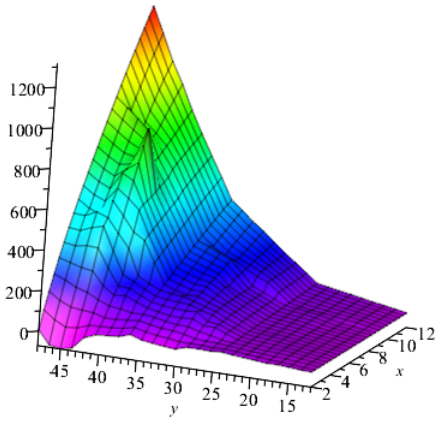


Figure 4.1: Plot of speedup from computing block diagonal form of A'

the entry in row i column j of A. Generate $2k$ vertices and partition them into two equal sets such that $V = R \cup C$. The vertices of R and C represent the rows and columns of A respectively. Let r_i and c_j represent the vertices in R and C respectively. Now scan through the matrix and whenever $a_{ij} \neq 0$ draw an edge e_{ij} from $r_i \in R$ to $c_j \in C$. This creates a companion bipartite graph for A, denoted H_A. After finishing this step we now run any algorithm that identifies the connected components of H_A. Depth-First Search is a good option here, as it runs in linear time ($\mathcal{O}(|V| + |E|)$) and easy to implement. As we will see shortly, if the matrix A has a block diagonal form then each connected components of the companion graph defines a partition of the row and column vertices. Since we labelled these vertices, they correspond to the rows and columns of each sub-matrix along the main diagonal.

$$A = \begin{bmatrix} 3 & 0 & 1 & 0 & 9 \\ 0 & 2 & 0 & 3 & 0 \\ 5 & 0 & 0 & 0 & 4 \\ 0 & 5 & 0 & 7 & 0 \\ 2 & 0 & 7 & 0 & 6 \end{bmatrix} \qquad A' = \begin{bmatrix} \mathbf{3} & \mathbf{1} & \mathbf{9} & 0 & 0 \\ \mathbf{5} & \mathbf{0} & \mathbf{4} & 0 & 0 \\ \mathbf{2} & \mathbf{7} & \mathbf{6} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{2} & \mathbf{3} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{7} \end{bmatrix}$$

(a) Random matrix A (b) Block diagonal form of A

Figure 4.2: Transforming a matrix into block diagonal form

To provide some additional intuition, we present a short visual example. We use a matrix over the integers for simplicity, and because the relevant part of a matrix for Algorithm 1 is where the non-zero entries are not what they are. While this example is small and the reader can probably detect the blocks by inspection, this is not practical when the matrices

Algorithm 1: Link

input : M

M *is a matrix with full rank, and has a block diagonal form*

output: Blocks along the main diagonal of M

begin

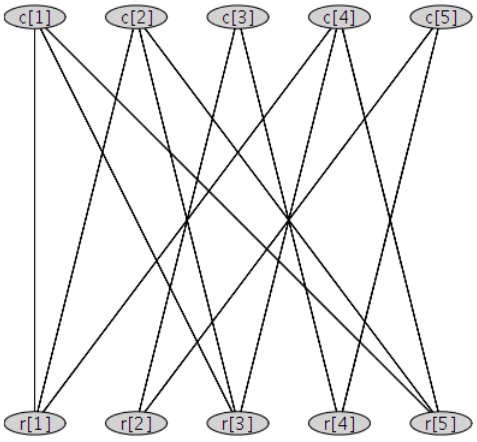
- $E \longleftarrow \varnothing$
- $MatxList \longleftarrow$ empty list
- $Blist \longleftarrow$ empty list
- $n \longleftarrow \text{order}(M)$
- `// Initialize the set of vertices`
- $V \longleftarrow \{r_1, r_2, \ldots, r_n, c_1, c_2, \ldots, c_n\}$
- `// Build the set of edges`
- $E \longleftarrow \{(r_i, c_j) : M_{i,j} \neq 0\}$
- `// Construct graph with on vertices` V `and edges` E
- $G \longleftarrow \text{Graph}(V, E)$
- `// Run Depth-First Search on` G `to identify connected components`
- $CC \longleftarrow \text{ConnectedComponents}(G)$
- `// Identify rows and columns of each block`
- **for** $\ell \in CC$ **do**
  - $Rlist \longleftarrow \{i : (r_i, c_j) \in \ell\}$
  - $Clist \longleftarrow \{j : (r_i, c_j) \in \ell\}$
  - $MatxList \longleftarrow MatxList \cup \text{SubMatrix}(M, Rlist, Clist)$
- **return** $MatxList$

get large.

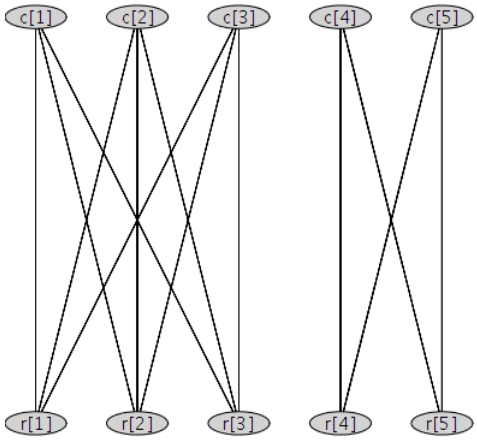
Example 4.2.1. Consider the following matrix:

$$A = \begin{bmatrix} 3 & 16 & 0 & 15 & 0 \\ 0 & 0 & 32 & 0 & 9 \\ 2 & 14 & 0 & 41 & 0 \\ 0 & 0 & 17 & 0 & 16 \\ 27 & 21 & 0 & 33 & 0 \end{bmatrix} \tag{4.1}$$

Our goal here is to determine if the rows and columns of A can be permuted in such a way that it is either in block diagonal form, or upper block diagonal form. If this is in fact possible, we would also like to know which rows and column belong to each block. `Link` begins by producing the bipartite graph in figure 4.3a.



(a) Bipartite construction of matrix A



(b) Bipartite construction of matrix A'

Now we can run any algorithm that identifies the connected components of this graph. Depth-First Search is a good choice as it runs in $\mathcal{O}(|V| + |E|)$. If we let n be the order of the matrix A, then $|V| = 2n$ and $|E| = hn^2$ for some h constant $h \in \mathbb{Q}$. Since H_A is a bipartite graph, it can have at most n^2 edges. The matrices we usually work with are sparse, so h could be as small as $\frac{1}{20}$. Hence $\mathcal{O}(|V| + |E|)$ simplifies to $\mathcal{O}(2n + hn^2) = \mathcal{O}(n + hn^2)$. Algorithm 1 runs Maple's `ConnectedComponents` command which gives us:

$$L = [[c_1, c_2, c_4, r_1, r_3, r_5], [c_3, c_5, r_2, r_4]] \,.$$

This tells us that the first block is of order 3, defined by the rows $1, 3, 5$ and columns $1, 2, 4$. The second block is identified in the same manner. With this information we can now permute the rows and columns of A to reveal the block structure. The rows and columns need to be permuted in the order that we see the indices in L. So the row and column permutations are:
rows:

$$\text{rows:}[1,2,3,4,5] \to [1,3,5,2,4] \quad \text{columns:}[1,2,3,4,5] \to [1,2,4,3,5]$$

.
This results in:

$$A' = \begin{bmatrix} \mathbf{3} & \mathbf{16} & \mathbf{15} & 0 & 0 \\ \mathbf{2} & \mathbf{14} & \mathbf{41} & 0 & 0 \\ \mathbf{27} & \mathbf{21} & \mathbf{33} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{32} & \mathbf{9} \\ 0 & 0 & 0 & \mathbf{17} & \mathbf{16} \end{bmatrix} \tag{4.2}$$

If we run `Link` on A', the companion graph $H_{A'}$ is shown in figure 4.3b.

Now we can clearly see the two connected components of the graph. This is the intuitive reason why this algorithm works without having to permute rows and columns. This is also the reason why `Link` fails when the matrix has a block triangular but not strictly block diagonal. The entries that do not belong to the blocks on the diagonal create "bridges" from one block to another. This effectively puts an edge from one connected component to another, and prevents any connected components algorithm from differentiating between multiple blocks. Fortunately we have a solution in that case, and it is the main topic of discussion in the next section.

4.3 Phase 1: Clarify

4.3.1 Upper block triangular matrices

Now suppose we are given a matrix that has a block triangular form, but not a strictly block diagonal form. The difference here begin that there exists some non-zero entries in blocks above the main diagonal. It can be shown that the determinant of these matrices is also the product of the determinant of the matrices along the main diagonal. In other words the determinant of such a matrix does not depend on the entries in blocks off the main diagonal. If we use Algorithm 1 as seen in the previous section, it will fail to reliably

identify any block structure. An example illustrates what must occur in order for us to retrieve the blocks in this case.

Observation 4.3.1. Let A be an upper block triangular matrix.

$$A = \begin{bmatrix} 1 & 4 & 2 & 9 \\ 6 & 5 & 4 & 0 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 4 & 8 \end{bmatrix}, \qquad \det(A) = -836 \tag{4.3}$$

We can obtain a new matrix A' from A by adding 1 to a_{13}. This corresponds to adding the outer product of the two column vectors

$$u = < 1, 0, 0, 0 >, v = < 0, 0, 1, 0 > .$$

to the matrix A.

$$A + uv^T = A' = \begin{bmatrix} 1 & 4 & \mathbf{3} & 9 \\ 6 & 5 & 4 & 0 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 4 & 8 \end{bmatrix}, \qquad \det(A') = -836 \tag{4.4}$$

It follows that $\det(A) = -836 = \det(A')$.

In addition, row and column swaps only change the sign of the determinant. Hence the determinant of a block triangular matrix whose rows and columns have been permuted is still the product of the determinants of the blocks along the main diagonal. However, if the rows and columns have been permuted, we cannot see where the blocks are and which entries belong to which block. So we need some way of learning whether an arbitrary entry belongs to some block along the main diagonal; this amounts to determining if it contributes to the determinant.

With that, we introduce the key lemma which acts as the defining identity of our new algorithm. In addition, we reproduce the proof found in [DZ07] because it is short and may help in understanding why the algorithm on following page is correct.

Lemma 4.3.2 (Matrix Determinant Lemma)**.** *[DZ07, Lemma 1.1] If A is an invertible $n \times n$ matrix, and u and v are two n-dimensional column vectors, then*

$$\det(A + uv^T) = (1 + v^T A^{-1} u) \det(A) \tag{4.5}$$

Proof. We may assume $A = I$, the $n \times n$ identify matrix, since then equation 4.5 follows from $A + uv^T = A(I + A^{-1}uv^T)$ in the general case. In this special case, the result comes

from the equality

$$\begin{bmatrix} I & 0 \\ v^T & 1 \end{bmatrix} \begin{bmatrix} I + uv^T & u \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ -v^T & 1 \end{bmatrix} = \begin{bmatrix} I & u \\ 0 & 1 + v^T u \end{bmatrix} \tag{4.6}$$

□

The main idea is as follows: given a matrix A over the integers we look at the location of zeros in A^{-1} in order to set equal to zero entries in A without altering the determinant of A. If A is a symbolic matrix, then we induce an evaluation homomorphism on A, and do the same thing on an image of A. The next sub-section will explain this idea in greater detail.

4.3.2 Explanation and proof for Clarify algorithm

Algorithm 2: Clarify

input : A, n

1 *A is an $n \times n$ matrix with full rank, n is order of A, with $a_{ij} \in \mathbb{Z}[x_1, \ldots, x_t]$*

output: A matrix M with $\det(M) = \det(A)$, or the input matrix A

2 **begin**

3 $M \longleftarrow A$

4 Pick a suitably large prime, e.g $2^{62} < p < 2^{63}$

5 Pick $\gamma = (\gamma_1, \ldots, \gamma_t)$ at random from $\mathbb{F}_p^t$

6 $B \longleftarrow$ Evaluate M at $(x_j = \gamma_j \ : 1 \leq j \leq t) \bmod p$

7 **if** $\text{rank}(B) < n$ **then**

8 Go back to line 4 `//` γ `is unlucky`

9 $V \longleftarrow B^{-1} \bmod p$

10 **for** $i \leftarrow 1$ **to** n **do**

11 **for** $j \leftarrow 1$ **to** n **do**

12 **if** $v_{ij} = 0$ **then**

13 $m_{ji} \longleftarrow 0$

14 **return** M

Our new algorithm is comprised of two separate sub-routines. Usually `Clarify` (Algorithm 2) is run first, followed by `Link` (Algorithm 1). As we have seen it is possible that `Link` can identify block structure without the use of `Clarify`. If the matrix has a block triangular form but not a strictly block diagonal form, then using `Clarify` is required for `Link` to return the blocks.

By Lemma 4.5, we have seen that we can update the determinant of a matrix without having to compute another determinant. `Clarify` exploits this by checking if the following

equality holds:

$$\det(A + uv^T) = (1 + v^T A^{-1} u)\det(A) \stackrel{?}{=} \det(A). \tag{4.7}$$

In our implementation of `Clarify`, the column vectors u and v are always unit vectors with exactly one non-zero entry; this is implicit as we do not perform any vector-matrix multiplications. `Clarify` is a modular algorithm, as we perform all computations over a finite field $\mathbb{F}_p$. Hence the inverse of B is computed modulo some prime p. This allows us to compute block diagonal forms of large matrices without creating massive numbers in B. Lastly, `Clarify` is a probabilistic algorithm, as it could happen that $m_{ij}^{-1} \neq 0$ but $v_{ij} = m_{ij}^{-1}(\gamma) = 0$. With this in mind, we introduce the Schwartz-Zippel Lemma, which will be used in the approaching proof.

Lemma 4.3.3 (Schwartz-Zippel Lemma)**.** *Let $f \in k[x_1, x_2, \ldots, x_n]$ be a* ***non-zero*** *polynomial of total degree $d \geq 0$ over a field k. Let S be a finite subset of k and let $r_1, r_2, \ldots, r_n$ be selected at random independently and uniformly from S. Then*

$$\text{Prob}[f(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

In order to explain Algorithm `Clarify` in greater detail, we will provide a proof of correctness. This proof will not only show why the algorithm is correct, but also why it is implemented in the way it has been.

Theorem 4.3.4. *Let A be an $n \times n$ matrix of rank n, where $a_{ij} \in \mathbb{F}_p[x_1, \ldots, x_t]$. Let $C = \text{adj}(A)$ and let $B \geq \max \deg(c_{ij})$. Let p be the prime chosen in Algorithm* `Clarify`*. Then the output matrix M of Algorithm* `Clarify` *satisfies $\det(M) \neq \det(A)$ with probability at most $\frac{n^2 B}{p}$.*

Proof. Let $C = \text{adj}(A)$. To use the Schwartz-Zippel Lemma, we need a degree bound $B \geq \deg(c_{ij})$. We can use

$$B = \min(\sum_{i=1}^{n} \max_{j=1}^{n} \deg(c_{ij}), \sum_{j=1}^{n} \max_{i=1}^{n} \deg(c_{ij})).$$

Then $\deg(\det A) \leq B$ and $\deg(c_{ij}) \leq B$. Recall that $A^{-1} = \frac{C}{\det(A)}$. In algorithm `Clarify`, $V = A(\gamma)^{-1} = \frac{C(\gamma)}{\det(A)(\gamma)}$ and $\det(A)(\gamma) \neq 0$. If $c_{ij} \neq 0$ but $v_{ij}(\gamma) = 0$, we say algorithm `Clarify` is unlucky. We wish to determine the probability that `Clarify` outputs a matrix M with $\det(M) \neq \det(A)$. This can happen in one of two ways. First, if a $c_{ij} \neq 0$ but

$v_{ij} = 0$, or in other words $c_{ij} \neq 0$ but $c_{ij}(\gamma) \equiv 0 \mod p$. Then we have the following bound:

$$\begin{aligned}\text{Prob}[c_{ij} \neq 0 \wedge c_{ij}(\gamma) \equiv 0 \mod p \text{ for some } ij] &\leq \prod_{1 \leq i,j \leq n} \text{Prob}[c_{ij} \neq 0 \wedge c_{ij}(\gamma) \equiv 0 \mod p] \\ &\leq \frac{n^2 \cdot \max\limits_{1 \leq i,j \leq n} \deg(c_{ij})}{p} \\ &\leq \frac{n^2 B}{p}\end{aligned}$$

The other way we could get unlucky is if the prime p divides every coefficient of $\det(A)$. Since we are working modulo p, this would cause the determinant to vanish. In practice this has never occurred because usually the system of polynomial equations has small coefficients to begin with. This in turn means the coefficients of the Dixon resultant will be relatively small. Given that p is of suitable size, it would be very unlikely for a non-zero coefficient to belong to the residue class [0]. Overall, the bigger p is, the less likely Algorithm `Clarify` will return a bad matrix M.

□

To summarize, if A has a block triangular form, then `Clarify` will delete all entries that lie above/below the block diagonal; it transforms the matrix into one which has a block diagonal form without knowing first what that block diagonal form is. After `Clarify` is run of a matrix that has a block triangular form, the resulting matrix has a block diagonal form. We are now in the case where `Link` will correctly return all blocks along the main diagonal.

We finish this chapter with a small demonstration of `Clarify`.

Example 4.3.5. Let

$$A = M = \begin{bmatrix} 36y^2 + 69 & 76 + 84y \\ 0 & 62y + 1 \end{bmatrix}, \qquad p = 997.$$

`Clarify` produces the following objects:

$$S = \{x = 771, y = 218\}, \quad B = \begin{bmatrix} 81 & \mathbf{442} \\ 0 & 556 \end{bmatrix}, \quad V = \begin{bmatrix} 837 & 493 \\ \mathbf{0} & 945 \end{bmatrix}.$$

Looking at V, $v_{21} = 0$ indicates that we can set b_{12} equal to zero, which implies we can also set m_{12} equal to zero. Since v_{21} was the only entry that was identical to zero, Algorithm `Clarify` terminates and outputs:

$$M = \begin{bmatrix} 36y^2 + 69 & 0 \\ 0 & 62y + 1 \end{bmatrix}, \quad \det(M) = \det(A).$$

Chapter 5

Empirical Results

We tested 15 polynomial systems using the KSY-variant of Dixon's method, all coming from real-world geometric and scientific problems. For each system, we present an assortment of information.

In Table 5.1, we show the number of polynomial equations in the system (**Eqns**), the number of variables which are to be eliminated (**Vars**), and the number of indeterminates in **c** (**Pars**). We also mention the size of the associated Dixon matrix Θ *before* we searched for a sub-matrix of maximal rank. After finding a sub-matrix Θ' of Θ which has maximal rank, we recorded the **rank** of Θ'. Then we attempted to compute the upper block triangular or block diagonal form of Θ'. We define the **Sparsity** of a matrix as the number of entries of a matrix that are zero, divided by the total number of entries in the matrix Θ'. The columns **Time (M.E)** and **Time (EDF)** show the time it took to compute the determinant of the smallest block along the main diagonal of Θ'. Finally **nops(**$\mathcal{DR}$) shows the number of terms in a Dixon resultant of the corresponding system.

Table 5.2, **Block Structure** shows the sizes of the blocks along the main diagonal of each Θ'. Next to this column, **Number of Multiplications** shows how many polynomial multiplications the method of Minor Expansion by Gentleman & Johnson [GJ73] requires for computing each sub-matrix along the main diagonal of Θ'. These two lists are ordered from left to right, and are in a one-to-one correspondence. For example, the first row of Table 5.2 says that the Minor Expansion algorithm by Gentleman & Johnson requires 229568 and 7767 multiplications to compute the determinants of the order 17 and order 12 blocks along the main diagonal respectively. Most importantly, the number of multiplications for each system is within the computable range. We hope this provides some insight as to what Dixon matrices look like in practice.

Table 5.1: Systems Information for Dixon matrices + Minor Expansion versus Dixon-EDF

System	**Eqns**	**Vars \Pars**	**dim Θ**	**Rank**	**Sparsity**	**Time (M.E)**	**Time (EDF)**	**nops**($\mathcal{DR}$)
bricard	6	5/12	41×44	29	0.804	166.200	69.80	1111775
cathedral†	6	5/2	46×41	34	0.748	0.896	0.026	7
heron2d†	3	2/4	3×3	3	0.375	0.000	0.027	7
heron3d†	6	5/7	16×14	13	0.769	0.001	0.008	23
heron4d†	10	9/11	103×75	63	0.941	0.026	0.043	1471
heron5d	15	14/16	707×514	399	0.989	!	36.39	?
image2d†	6	5/6	32×29	29	0.907	0.000	0.005	22
image3d†	10	9/10	178×152	130	0.975	0.009	0.036	1456
image4d†	10	9/10	136×126	120	0.976	0.008	0.023	704
imageFlex†	10	9/10	136×126	120	0.976	0.006	0.023	704
robotarms	4	3/4	32×32	16	0.531	!	3.38	?
sys22	4	3/9	26×26	22	0.736	556.2	519.6	3990252
tot	3	2/3	40×40	33	0.567	!	46.78	?
vanaubel†	9	8/6	28×28	28	0.893	3.220	0.218	32166

Minor Expansion timings done with Maple's implementation of Gentleman & Johnson

$\dagger$ = Projection operator computable using Maple's `Groebner[Basis]` command in < 1000 seconds

! = Ran out of memory attempting calculation

Table 5.2: Block structure of Dixon matrices + Minor Expansion multiplication count

System	**Block Structure**	**Number of Multiplications**
bricard	[17,12]	[229568,7767]
cathedral†	[16,18]	[170905,955315]
heron2d†	[4]	[10]
heron3d†	[6,7]	[41,73]
heron4d†	[14,14,17,18]	[648,1592,19078,46586]
heron5d	[49,50,52,49,48,48,50,53]	[12350078,?,?,20206571,24905078,?,?,?]
image2d†	[10,7,7,5]	[111,99,39,39]
image3d†	[13,14,14,15,18,19,18,19]	[575,579,3462,914,4527,7118,14777,4737]
image4d†	[13,14,14,15,17,16,16,15]	[945,719,1505,664,6872,4215,5043,3575]
imageFlex†	[13,14,14,15,17,16,16,15]	[945,719,1505,644,6872,4215,5043,3575]
robotarms	[8,8]	[728,544]
sys22	[11,11]	[4090,4276]
tot	[17,16]	[890695,442778]
vanaubel†	[7,7,7,7]	[93,103,68,39]

The block structure of a system is sensitive to the variable elimination order

† = Projection operator computable using Maple's `Groebner[Basis]` command in < 1000 seconds

Chapter 6

Conclusion

We begin the concluding chapter with two open research questions. While solving these is not going to change the basic procedure we have outline in the paper, it could potentially increase the speed at which we produce Dixon resultants.

Open Question 6.0.1. Does there exists sufficient conditions on a polynomial system $\mathcal{F}$ for its Dixon matrix to have a block diagonal form?

It is possible that these matrices have even more structure then what we currently know about. If we knew more about the polynomials that generated these matrices, we might be able to develop even better tools for taking the determinants of the Dixon matrices they give rise to.

Open Question 6.0.2. For which systems are iterative resultant computation methods more efficient that Dixon's method?

Dixon's method simultaneously eliminates n variables from a system of $n+1$ polynomial equations. Another technique is to systematically eliminate one variables from the systems one at a time using simpler resultant formulations such as Sylvester's resultant. For large systems of polynomial equations Dixon's method seems to be the better choice, however it has not been well documented which techniques perform better for which systems in general.

Resultant matrices have much to be discovered about them, but our hope is that we have illuminated a fruitful research path for others. The algorithms and procedures from Chapter 2 should be implemented by those who are interested in learning more, as we found a hands-on learning approach to this kind of work more effective. The key idea from Chapter 3 was that despite the seemingly prohibitive cost of the method of Minor expansion, it was the most effective algorithm with respect to symbolic determinant computation. Gentleman & Johnson demonstrated this beautifully when they showed that their version of Minor expansion was actually the most effective when computing symbolic determinants

compared to methods which are superior in numerical computation. Chapter 4 contained our main results of the research project. Techniques and algorithms like `Clarify` and `Link` were crucial for computing many of the symbolic determinants that otherwise would not be possible. The implementation of `Clarify` was designed to be fast yet simple, and we have successfully used it to locate blocks within matrices of up to order 1000.

We hope it is clear that resultants can aid in solving polynomial systems. They can easily outperform competing methods, such as Buchburger's algorithm, if careful implementation is made. Since resultants computed from resultant matrices rely heavily on the speed at which we can compute symbolic determinants, high performance algorithms for computing symbolic determinants are also of great interest. For those interested in learning more about computer algebra, see [GCL92] for an nice introduction. For those who are interested in a more rigorous treatment of the Dixon matrix, and Dixon resultants in general, consult [CK02], [Chi01], and [CK04]. For those interested in learning more about resultants, good places to start would be [CLO15] and [Has07]. For more examples and applications of Dixon resultants see [Kap97], [Lew08], [LC06], [PZAG08], and [Lew17].

Bibliography

[Buc76] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *ACM SIGSAM Bull.*, 10(3):19–29, 1976.

[Cay57] A. Cayley. Note sur la méthode d'élimination de Bezout. *J. Reine Angew. Math.*, 53:366–367, 1857.

[Chi01] Eng-Wee Chionh. Rectangular corner cutting and Dixon A-resultants. *J. Symbolic Comput.*, 31(6):651–669, 2001.

[CK02] Arthur D. Chtcherba and Deepak Kapur. On the efficiency and optimality of Dixon-based resultant methods. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 29–36. ACM, New York, 2002.

[CK04] Arthur D. Chtcherba and Deepak Kapur. Constructing Sylvester-type resultant matrices using the Dixon formulation. *J. Symbolic Comput.*, 38(1):777–814, 2004.

[CLO15] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms.* Undergraduate Texts in Mathematics. Springer, Cham, fourth edition, 2015. An introduction to computational algebraic geometry and commutative algebra.

[Dix08] A. L. Dixon. On a Form of the Eliminant of Two Quantics. *Proc. London Math. Soc. (2)*, 6:468–478, 1908.

[Dix09] A. L. Dixon. The Eliminant of Three Quantics in two Independent Variables. *Proc. London Math. Soc. (2)*, 7:49–69, 1909.

[Dym13] Harry Dym. *Linear algebra in action*, volume 78 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, second edition, 2013.

[DZ07] Jiu Ding and Aihui Zhou. Eigenvalues of rank-one updated matrices with some applications. *Appl. Math. Lett.*, 20(12):1223–1226, 2007.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for computer algebra.* Kluwer Academic Publishers, Boston, MA, 1992.

[GJ73] W. M. Gentleman and S. C. Johnson. Analysis of algorithms, a case study: determinants of polynomials. In *Fifth Annual ACM Symposium on Theory of Computing (Austin, Tex., 1973)*, pages 135–141. 1973.

[GKZ94] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky. *Discriminants, resultants, and multidimensional determinants.* Mathematics: Theory & Applications. Birkhäuser Boston, Inc., Boston, MA, 1994.

[Has07] Brendan Hassett. *Introduction to algebraic geometry.* Cambridge University Press, Cambridge, 2007.

[Kap97] Deepak Kapur. Automated geometric reasoning: Dixon resultants, gröbner bases, and characteristic sets. In Dongming Wang, editor, *Automated Deduction in Geometry*, pages 1–36, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[KS95] Deepak Kapur and Tushar Saxena. Comparison of various multivariate resultant formulations. In *ISSAC*, volume 95, pages 187–194. Citeseer, 1995.

[KS97] Deepak Kapur and Tushar Saxena. Extraneous factors in the Dixon resultant formulation. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (Kihei, HI)*, pages 141–148. ACM, New York, 1997.

[KSY94] Deepak Kapur, Tushar Saxena, and Lu Yang. Algebraic and geometric reasoning using Dixon resultants. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 99–107. ACM, 1994.

[Lay06] David C Lay. *Linear algebra and its applications.* Third edition, 2006.

[LC06] Robert H Lewis and Evangelos A Coutsias. Algorithmic search for flexibility using resultants of polynomial systems. In *International Workshop on Automated Deduction in Geometry*, pages 68–79. Springer, 2006.

[Lew96] Robert H Lewis. The Kapur-Saxena-Yang variant of the Dixon resultant. 1996.

[Lew08] Robert H. Lewis. Heuristics to accelerate the Dixon resultant. *Math. Comput. Simulation*, 77(4):400–407, 2008.

[Lew17] Robert H. Lewis. Dixon-EDF: the premier method for solution of parametric polynomial systems. In *Applications of computer algebra*, volume 198 of *Springer Proc. Math. Stat.*, pages 237–256. Springer, Cham, 2017.

[Mac03] F. S. MacAulay. Some Formulae in Elimination. *Proc. Lond. Math. Soc.*, 35:3–27, 1903.

[Pal13] B. Paláncz. Application of Dixon resultant to satellite trajectory control by pole placement. *J. Symbolic Comput.*, 50:79–99, 2013.

[PF90] Alex Pothen and Chin-Ju Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Software*, 16(4):303–324, 1990.

[PZAG08] Béla Paláncz, Piroska Zaletnyik, Joseph L Awange, and Erik W Grafarend. Dixon resultant's solution of systems of geodetic polynomial equations. *Journal of Geodesy*, 82(8):505–511, 2008.

Appendix A

Code

Algorithm 3: BuildDixon

input : $P = \Delta(\mathcal{F}, X)$, X, $\bar{X}$

1 *P is a Dixon polynomial, $X = \{x_1, \ldots, x_n\}$ is the set of original variables, $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ is the set of new variables*

output: Dixon matrix

```
2  begin
3      BL ⟵ list of monomials of P in the variables X
4      BR ⟵ list of monomials of P in the variables X̄
5      r ⟵ |BL|
6      c ⟵ |BR|
7      D ⟵ r × c zero matrix
8      i ⟵ 1
9      for m1 ∈ BL do
10         c1 ⟵ coeff(P, m1)
11         j ⟵ 1
12         for m2 ∈ BR do
13             D_{i,j} ⟵ coeff(c1, m2)
14             j ⟵ j + 1
15         i ⟵ i + 1
16     return BL, D, BR
```

Algorithm 3 constructs the dixon matrix for a given Dixon polynomial by separating the variables from the parameters. Hence this matrix will be over $k[\mathbf{c}]$.

Algorithm 4 finds a sub-matrix of maximal rank. The SubMatrix command takes as its inputs a matrix A, and two sets of integers $L1$ and $L2$ that define the rows and columns of the sub-matrix being extracted.

Algorithm 4: Chop

input : A, N, m, n

A is a Dixon matrix, $N = \{x_n, a, b, \ldots\}$ is a list of t variables in A to be evaluated, m is the number of rows, n is the number of columns

output: Dixon matrix minor of full rank

begin

- Pick a suitably large prime $p > 2^{63}$
- Pick $\gamma = (\gamma_1, \ldots, \gamma_t)$ at random from $\mathbb{F}_p$
- Evaluate the entries of A at γ
- $A1 \longleftarrow A$ in row echelon form
- $L1 \longleftarrow$ PivotColumns($A1$)
- $A2 \longleftarrow A$ in column echelon form
- $L2 \longleftarrow$ PivotColumns($A2$)
- $M \longleftarrow$ Submatrix($A, L1, L2$)
- **return** M