Elimination with the Dixon Resultant

A fast algorithm implemented in Mathematica provides one-step elimination of a block of unknowns from a system of polynomial equations.

by George Nakos and Robert M. Williams

We implement an algorithm of Kapur, Saxena and Yang that computes the Dixon resultant and its generalization by these authors. The Dixon resultant, can be used to eliminate a number of unknowns from a system of polynomial equations in one step. To the *Mathematica* user this means code that complements and greatly enhances the command `Eliminate`. Our program also improves the command `Resultant`, which implements the Sylvester resultant. This is up to constant factor(s) a special case of the Dixon resultant. The Dixon resultant has two advantages over the Sylvester resultant: 1) The end matrix has smaller size; hence, it is often easier to row reduce it or compute its determinant. 2) A whole block of variables can be eliminated in one calculation, instead of the successive eliminations. In the process we also offer a symbolic Gauss elimination (without scaling) that may be of independent interest.

1. Introduction

From algebraic geometry to computer graphics there is a substantial need for efficient solutions of systems of polynomial equations. The theory of resultants was developed and used during the nineteenth and early twentieth century [Cayley 1865], [Dixon 1908], [Macaulay 1916], [Macaulay 1994] to solve such systems. Recently, there has been renewed interest in the subject because fast computers and the availability of symbolic mathematical packages have made old and new problems much more tractable.

We are interested in the Dixon resultant [Dixon 1908] and a recent generalization of it by Kapur, Saxena and Yang [Kapur, Saxena, and Yang 1994]. References on resultants and related algebra include [van der Waerden 1950], [Salmon 1964], [Mostowski and Stark 1964], [Gantmacher 1959], [Chionh 1990], [Canny 1990], [Gelfand, Kapranov, and Zelevinsky 1994]. The Dixon resultant is discussed in [Dixon 1908], [Kapur and Lakshman 1992], [Kapur, Saxena, and Yang 1994], [Kapur and Saxena], [Chionh 1990]. An excellent treatment of computational aspects of polynomials can be found in [Knuth 1969].

The material of this manuscript is organized as follows: In Section 2 we introduce the classical Dixon resultant and discuss its merits and limitations. In Section 3 we outline the work of Kapur, Saxena and Yang. In Sections 4 and 5 we compare our main function, `DixonResultant`, with *Mathematica*'s `Eliminate` and `Resultant` and point out strengths and weaknesses. It *seems* that for small easily factorable polynomials `Eliminate` is at least as good as `DixonResultant`. However, if the polynomials are of higher degree, or cannot be factored, or have constant terms, `Eliminate` is very slow or fails, while `DixonResultant` may yield an answer in seconds. On the negative side, the computation of the Dixon resultant can introduce extraneous factors that are avoided by slower methods. (Gröbner bases, for example.) Section 6 briefly shows how to use the part of our code that performs symbolic Gauss elimination. Finally, Section 7 follows with a description of our *Mathematica* program that computes the Dixon resultant. The program is based on the main algorithm of [Kapur, Saxena, and Yang 1994] and is distributed with the electronic subscriptions as `Dixon.m`.

2. The Classical Dixon Resultant

Cayley's Formulation of Bezout's Method

First we recall Cayley's formulation [Cayley 1865] of Bezout's method for solving a system of two polynomial equations. It seems, however, (as professor Kapur points out) that the method is actually due to Euler!

Let $f(x)$ and $g(x)$ be polynomials in x, let $deg = \max(\text{degree}(f), \text{degree}(g))$, and let a be an auxiliary variable. The quantity

$$\delta(x, a) = \frac{1}{x - a} \begin{vmatrix} f(x) & g(x) \\ f(a) & g(a) \end{vmatrix}$$

is a symmetric polynomial in x and a of degree $deg - 1$ which we call the **Dixon polynomial** of f and g. Cayley (and Bezout in different notation) observed that: Every common zero of f and g is a zero of $\delta(x, a)$ for all values of a. Hence, at a common zero, each coefficient of a^i in $\delta(x, a)$ is $\equiv 0$. In matrix notation,

$$M \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{deg-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where, the columns of the $deg \times deg$ matrix M consist of the coefficients of the a^i's. This yields a homogeneous system in new variables v_1, v_2, … v_{deg} corresponding to x^0, x^1, …, x^{deg-1} and equations corresponding to the coefficients of a^i.

$$M \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{deg} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

System (1) has non–trivial solutions if and only if its determinant D is zero. D is called the **Dixon resultant** of f and g and M is the **Dixon matrix.** We conclude that:

The vanishing of the Dixon resultant D is a necessary condition for the existence of a common zero of f and g.

Cayley's formulation of Bezout's method is illustrated in Examples 1–4. We compute M and D for the given f and g and comment on the relation between the vanishing of D and the common zeros of the system $f = 0, g = 0$. In each case we factor f and g for easy checking.

Example 1: [Numeric Coefficients – Common Solution]

$$f(x) = x^3 - 2x^2 - 11x + 12 = (x - 1)(x + 3)(x - 4)$$

$$g(x) = x^2 + 3x - 4 = (x+4)(x-1)$$

Solution: The Dixon polynomial is

$$\delta = \frac{1}{x-a} \begin{vmatrix} x^3 - 2x^2 - 11x + 12 & x^2 + 3x - 4 \\ a^3 - 2a^2 - 11a + 12 & a^2 + 3a - 4 \end{vmatrix} =$$

$$x^2 a^2 + 3x^2 a + x a + 3(x a)^2 - 4a - 4a^2 - 4x^2 - 4x + 8$$

Equating the coefficients of the powers of a to zero yields,

$$8 - 4x - 4x^2 = 0$$

$$-4 + x + 3x^2 = 0$$

$$-4 + 3x + x^2 = 0$$

Therefore,

$$M = \begin{bmatrix} 8 & -4 & -4 \\ -4 & 1 & 3 \\ -4 & 3 & 1 \end{bmatrix} \text{ and } D = \begin{vmatrix} 8 & -4 & -4 \\ -4 & 1 & 3 \\ -4 & 3 & 1 \end{vmatrix} = 0$$

Since the system has a common root ($x = 1$) the resultant should be 0. (It is.) However, since D vanishes identically, we get no information on how to compute this common root.

Example 2: [Numeric Coefficients – No Common Solution]

$$f(x) = x^3 - 2x^2 - 11x + 12 = (x-1)(x+3)(x-4)$$

$$g(x) = x^2 + 5x + 4 = (x+4)(x+1)$$

Solution: A similar calculation yields

$$D = \begin{vmatrix} -104 & -20 & 4 \\ -20 & 5 & 5 \\ 4 & 5 & 1 \end{vmatrix} = 800$$

Since D is nonzero, there are no common zeros.

Let us examine now two cases with parametric coefficients.

Example 3: [Parametric Coefficients – Full Root Recovery]

$$f(x) = x^2 A + 3x^2 - x A - A^3 - 2A^2 - 3x + 3A$$

$$= (x + A - 1)(A + 3)(x - A)$$

$$g(x) = x^2 + 4xA + 3A^2 = (x + 3A)(x + A)$$

Solution: The Dixon matrix is

$$M = \begin{bmatrix} -21A^2 + 5A^3 + 4A^4 & -3A + 11A^2 + 4A^3 \\ -3A + 11A^2 + 4A^3 & 3 + 13A + 4A^2 \end{bmatrix}$$

Its determinant is the resultant D

$$D = -8A^2(2A+1)(A+3)^2$$

Setting $D = 0$ and solving for A yields (with multiplicities)

$$A = -3, -3, \frac{-1}{2}, 0, 0$$

On the other hand, the true solution of $f(x, A) = 0, g(x, A) = 0$ over Q is

$$(x, A) = (9, -3), (3, -3), (0, 0), \left(\frac{3}{2}, \frac{-1}{2}\right)$$

Note that all the values of A of the solutions yielded the zeros of D. However, $A = 0$ has multiplicity 2 in $D = 0$, while there is only one solution of the system with $A = 0$.

In this example *all the zeros of the system were detected by the zeros of the Dixon resultant.*

Example 4: [Parametric Coefficients – No Information]

$$\begin{aligned} f(x) &= x^2 A - (x A)^2 + 3 x^2 - 4 x A + A^2 - 3 x + 3 A \\ &= (A + 3)(x - 1)(x - A) \\ g(x) &= x^2 + x A - x - A = (x - 1)(x + A) \end{aligned}$$

Solution: The Dixon matrix is

$$M = \begin{bmatrix} 6 A + 2 A^2 & -6 A - 2 A^2 \\ -6 A - 2 A^2 & 6 A + 2 A^2 \end{bmatrix}$$

and the determinant vanishes identically.

$$D \equiv 0$$

The true solution set of $f(x, A) = 0, g(x, A) = 0$ over Q is

$$(x, A) = (3, -3), (1, A), (0, 0)$$

This time, *the vanishing of D yielded no information on the common roots.*

Dixon's Generalization of the Cayley–Bezout Method

Dixon generalized Cayley's approach to Bezout's method to systems of three polynomial equations in two unknowns.

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \\ h(x, y) &= 0 \end{aligned}$$

δ is now defined by

$$\delta(x, y, a, b) = \frac{1}{(x - a)(y - b)} \begin{vmatrix} f(x, y) & g(x, y) & h(x, y) \\ f(a, y) & g(a, y) & h(a, y) \\ f(a, b) & g(a, b) & h(a, b) \end{vmatrix}$$

for auxiliary variables a and b. One gets a homogeneous linear system as before by setting the power products $a^i b^j$ equal to zero. The corresponding determinant of the coefficient matrix is the Dixon resultant D.

Dixon proved that *for three generic 2 degree polynomials, the vanishing of D is a necessary condition for the existence of a common zero. Furthermore, D is not identically zero.*

Dixon's method and proofs easily generalize to a system of $n+1$ generic $n$degree polynomials in n unknowns.

Recall, that a polynomial is generic if all its coefficients are independent parameters, unrelated to each other. A polynomial in n variables is $n$degree if all powers to the maximum of each variable appear in it. (See [Kapur, Saxena, and Young 1994] for more details.)

$n+1$ polynomials in n variables, say, $p_j(x_1, \ldots, x_n)$, $j = 1, \ldots, n+1$, are generic $n$degree if there are n integers $k_1, \ldots, k_n$ such that

$$p_j = \sum_{i_1=0_1}^{k} \cdots \sum_{i_n=0_n}^{k} a_{j,i_1,\ldots,i_n} x_{11}^{i} \ldots x_{nn}^{i}, \qquad 1 \le j \le n+1$$

where the a's are distinct indeterminants.

For example, the three polynomials below are generic of n degree$(2, 1)$,

$$p_1 = a_0 + a_1 x + a_2 y + a_3 x y + a_4 x^2 + a_5 x^2 y$$
$$p_2 = b_0 + b_1 x + b_2 y + b_3 x y + b_4 x^2 + b_5 x^2 y$$
$$p_3 = c_0 + c_1 x + c_2 y + c_3 x y + c_4 x^2 + c_5 x^2 y$$

while, the following three polynomials are not generic $n$degree,

$$q_1 = 3 x + (a x)^2 + x^2 y$$
$$q_2 = y + x y - (a x)^2$$
$$q_3 = b + c_1 x + c_2 y$$

In q_1 the coefficient 3 is not an indeterminant (hence, q_1 is non–generic) and the coefficient of $x^0 y^0$ is missing (and not $n$degree).

Dixon's method applies to polynomials with symbolic variables, which allows for simultaneous elimination of a block of unknowns by only one calculation. This feature, along with the relatively small size of the resulting determinants (compared with other resultant methods) makes the method *very* attractive.

Unfortunately, if the polynomials are not generic and $n$degree, things can go wrong.

In the following examples we illustrate Dixon's method and its limitations on polynomials that are not necessarily generic or $n$degree.

Example 5:

$$f(x) = x y + 3 x - 3 y - 9 = (x - 3)(y + 3)$$
$$g(x) = x y - 3 x + 3 y - 9 = (x + 3)(y - 3)$$
$$h(x) = x y + 2 x - 2 y - 4 = (x - 2)(y + 2)$$

Solution: The Dixon polynomial is $30 y + 30 a$ and the Dixon matrix M is given by

$$\begin{array}{cc} & \begin{array}{cc} x^0 y^0 & x^0 y^1 \end{array} \\ \begin{array}{c} a^0 b^0 \\ a^1 b^0 \end{array} & \begin{bmatrix} 0 & 30 \\ 30 & 0 \end{bmatrix} \end{array}$$

The Dixon resultant $D = \det(M) = -900$ of M is non–zero. So, we expect no solutions (which is true).

Example 6:

$$f(x) = x\,y + 3\,x - y^2 - 3\,y = (x - y)\,(y + 3)$$
$$g(x) = x^2 + 4\,x\,y + 3\,y^2 = (x + 3\,y)\,(x + y)$$
$$h(x) = x^2 - x\,y - 2\,y^2 = (x - 2\,y)\,(x + y)$$

Solution: This time the Dixon matrix M is

$$\begin{array}{c|cccccc} & x^0\,y^0 & x^0\,y^1 & x^0\,y^2 & x^1\,y^0 & x^1\,y^1 & x^1\,y^2 \\ \hline a^0\,b^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a^0\,b^1 & 0 & 0 & 15 & 0 & 15 & 0 \\ a^1\,b^0 & 0 & 0 & -15 & 0 & -15 & 0 \\ a^1\,b^1 & 0 & 15 & 0 & -15 & -10 & 0 \\ a^2\,b^0 & 0 & 15 & 10 & -15 & 0 & 0 \\ a^2\,b^1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Its determinant is the resultant D, which is identically zero:

$$D \equiv 0$$

This situation yields no information on the common solution.

On the other hand, the system has common solutions, namely,

$$(x,\,y) = (0,\,0),\,(3,\,-3)$$

Note: We may reduce the size of the Dixon matrix by removing the zero rows and columns, as done in [Kapur, Saxena, and Yang 1994]. This in general reduces the size of the matrix in case of sparse polynomials or sparse Dixon polynomial. In this case we could write,

$$M = \begin{bmatrix} 0 & 15 & 0 & 15 \\ 0 & -15 & 0 & -15 \\ 15 & 0 & -15 & -10 \\ 15 & 10 & -15 & 0 \end{bmatrix}$$

(Note that the determinant is *still* zero, here.) This, however, may cause some temporary problems that will be fixed in Section 3.

From now on *we drop the zero columns and rows from the Dixon matrix and still call the resulting matrix Dixon matrix*.

Example 7:

$$f(x) = y\,x - 3\,y = (x - 3)\,y$$
$$g(x) = y\,x - 3\,x = x(y - 3)$$
$$h(x) = y\,x - 2\,y = (x - 2)\,y$$

Solution: The Dixon polynomial is $3\,y\,a$. The original Dixon matrix $M = \begin{bmatrix} 0 & 0 \\ 0 & 3 \end{bmatrix}$ reduces to the new one, [3]:

$$\begin{array}{c|c} & x^0\,y^1 \\ \hline a^1\,b^0 & [3] \end{array}$$

The determinant of the new Dixon matrix is non–zero, but this doesn't mean that the system has no solution! (It has the trivial solution.)

In Example 7 we saw that the vanishing of the determinant of the new Dixon matrix is *not even a necessary condition for the existence of solutions* of the system. The determinant of the original Dixon matrix is zero, as expected.

Example 8:

$$\begin{aligned} f &= x\,y + x\,A + x - A^2 - A + y^2 + y \\ &= y + A + 1 - A + x + y \\ g &= x^2 + x\,A - x + x\,y + y\,A - y \\ &= x + y\,x - 1 + A \\ h &= x^2 + x\,y + 2\,x - x\,A - y\,A - 2\,A \\ &= x + y + 2\,x - A \end{aligned}$$

Solution: The Dixon matrix M is given by

$$\begin{array}{cccccc} & x^0\,y^0 & x^0\,y^1 & x^0\,y^2 & x^1\,y^0 & x^1\,y^1 \\ a^0\,b^0 & -2\,A+2\,A^2 & A+2\,A^3+A^2 & 2\,A & A+2\,A^3+A^2 & 2\,A \\ a^0\,b^1 & -2\,A+2\,A^2 & -2+4\,A & 2\,A-1 & 2\,A & 2\,A-1 \\ a^1\,b^0 & 2\,A^2 & 5\,A-2\,A^2-4 & -3+2\,A & -2+3\,A-2\,A^2 & -3+2\,A \\ a^1\,b^1 & 2\,A & 2\,A-1 & 0 & -3+2\,A & 0 \\ a^2\,b^0 & 2\,A & 2\,A-1 & 0 & -3+2\,A & 0 \end{array}$$

Computation of the determinant yields zero. Hence, $D = \det(M) \equiv 0$.

Actually, the system $f(x, y, A) = 0, h(x, y, A) = 0, g(x, y, A) = 0$, has several solutions over Q, namely,

$$(x, y, A) = (0, -2, 1), (3, -5, -2), (0, 0, 0), \left(\frac{1}{2}, \frac{-3}{2}, \frac{1}{2}\right), \left(\frac{1}{2}, 0, \frac{1}{2}\right)$$

We see that in this case the resultant is 0 as expected but since it is identically zero it yields no information on the solutions of the system.

Example 9:

$$\begin{aligned} f &= (a\,x)^2 - x\,y\,a + (c - 2\,a)\,x + a\,y + 3\,c - 3 \\ g &= 2\,a^2\,x^2 - 2\,a^2\,x\,y - (y\,a)^2 - a^3 \\ h &= 8\,x\,a - 4\,a^2 \end{aligned}$$

Solution: The Dixon matrix after removing the zero second row is

$$M = \begin{bmatrix} -24\,a^3 + 4\,a^4\,c + 24\,a^3\,c - 16\,a^5 & -12\,a^5 & 12\,a^5 \\ 48\,a^3\,c - 48\,a^3 + 8\,a^4\,c + 4\,a^5 & 24\,a^4 & -24\,a^4 \end{bmatrix}$$

M this time is *not even square*. So the determinant is undefined.

Possible Problems with Dixon's Method

Dixon's method applies only to generic *n*degree polynomials. If this condition fails then one can face the following problems:

- 1. The Dixon matrix may be singular (Examples 6 and 8).
- 2. After we remove the zero rows and columns the vanishing of the determinant of the Dixon matrix may not give a necessary condition for the existence of a common zero (Example 7).

- 3. After we remove the zero rows and columns the Dixon matrix may not even be square. Hence, its determinant cannot be defined (Example 9).

3. The Kapur–Saxena–Yang Approach

Kapur, Saxena and Yang addressed all three problems successfully, provided a certain precondition holds. Let us describe their main theorem and algorithm.

Suppose we have a system of $n+1$ polynomial equations in n variables such that the coefficients of the polynomials are themselves polynomials in a finite set of parameters. Let M be the Dixon matrix obtained as before.

Let M' be an echelon form matrix obtained from M by using elementary row operations except for scaling of rows. (Such a reduction is always possible.) Let D be the product of all pivots of M'.

The Precondition: Assume that the column that corresponds to the monomial $1 = x_1^0 \, x_2^0 \ldots$ of the Dixon matrix is not a linear combination of the remaining ones. (In our notation this is the first column of the Dixon matrix.)

Theorem 10:

[Kapur–Saxena–Yang] If the precondition is true, then $D = 0$ is a necessary condition for the existence of common zeros.

This theorem yields a simple *algorithm* for obtaining the necessary condition $D = 0$, which we call the Kapur–Saxena–Yang Dixon resultant.

Algorithm (Computation of the Kapur–Saxena–Yang Dixon resultant)

Input: A set of polynomials, with numeric or parametric coefficients.

- 1. Compute the Dixon matrix M. If the precondition holds continue.
- 2. Row reduce M without scaling to row echelon form M'.
- 3. Compute the product D of the pivots of M'.

Output: D is the Kapur–Saxena–Yang Dixon resultant. Its vanishing is a necessary condition for a solution of the given system.

Checking for the Validity of the Precondition: Let M be the Dixon matrix and let $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_s$, be its columns. Also let $\mathbf{w} = (w_1, \ldots, w_s)^T$ be a solution of the system

$$M \, \mathbf{w} = \mathbf{0} \;\Leftrightarrow\; w_1 \, \mathbf{m}_1 + w_2 \, \mathbf{m}_2 + \cdots w_s \, \mathbf{m}_s = \mathbf{0}$$

Clearly, $\mathbf{m}_1$ is **not** a linear combination of $\mathbf{m}_2, \ldots, \mathbf{m}_s$ if and only if

$$w_1 = 0$$

So the *precondition is equivalent to the validity of* (2). This is, usually, a simple test in practice. For example, for numerical entries the reduced row echelon form of M should contain a first row of the form $[1\ \ 0 \times 0\ \ \cdots 0]$.

Let us now rework Examples 3, 4, 8, 9 by computing the Kapur–Saxena–Yang Dixon resultant, D.

Example 11:

Redo Example 3.

Solution: Since the first column of the Dixon matrix is not a scalar multiple of the second, the precondition of the Theorem applies. Hence, we may continue by reducing without scaling to get

$$\begin{bmatrix} -21\,A^2 + 5\,A^3 + 4\,A^4 & -3\,A + 11\,A^2 + 4\,A^3 \\ 0 & \frac{-8\left(3+7\,A+2\,A^2\right)}{-7+4\,A} \end{bmatrix}$$

Factoring the product of pivots yields, $D = -8\,A^2\,(3+A)^2\,(1+2\,A)$. Setting $D = 0$, is a necessary condition for the existence of a common zero. We solve for A to get

$$A = -3,\ -3,\ \frac{-1}{2},\ 0,\ 0$$

the same answer as before.

In our next example the precondition fails, so the theorem does *not* apply. However, if we row reduce anyway we get the A–values of *some* solutions of the system!

Example 12:

Redo Example 4.

Solution: Since the first column of the Dixon matrix is a scalar multiple of the second the precondition **fails**. So the Theorem does *not* apply. If we *reduce anyway*, we get

$$\begin{bmatrix} 6\,A + 2\,A^2 & -6\,A - 2\,A^2 \\ 0 & 0 \end{bmatrix}$$

and the pivot $2\,A^2 + 6\,A = 2\,A(A+3)$ yields $A = 0,\ -3$.

The true solution of the system is

$$(x,\ A) = (3,\ -3),\ (0,\ 0),\ (1,\ A)$$

So in this case by reducing we got the A–values for all but one solution (!). Only $(1,\ A)$ was not obtained from the pivot this time.

Example 13:

Redo Example 8.

Solution: First we show that the precondition of the Theorem holds by proving that the first column of the

Dixon matrix M is not a linear combination of the remaining

ones. Indeed, if we substitute $A = -1$ in M and row reduce, we get:

$$M_{\{A=-1\}} = \left[\begin{matrix} 4 & -2 & -2 & -2 & -2 \\ 4 & -6 & -3 & -2 & -3 \\ 2 & -11 & -5 & -7 & -5 \\ -2 & -3 & 0 & -5 & 0 \\ -2 & -3 & 0 & -5 & 0 \end{matrix}\right] \left[\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}\right]$$

So we see that the system $M_{\{A=-1\}}\,\mathbf{w} = \mathbf{0}$ has $w_1 = 0$ hence, the first column of $M_{\{A=-1\}}$ (hence, of M) is not a linear combination of the remaining ones.

We conclude that the precondition holds and the Theorem applies.

Row reduction of the Dixon matrix yields the Dixon–Kapur–Saxena–Yang resultant as a product of pivots:

$$D = 8\,(-1 + 2\,A)^2\,A(1 - A)\,(A + 2)$$

Setting $D = 0$, we get

$$A = -2, 0, 1, \frac{1}{2}, \frac{1}{2}$$

The solutions of the system $f(x, y, A) = 0, h(x, y, A) = 0, g(x, y, A) = 0$ are:

$$(x, y, A) = (0, -2, 1), (3, -5, -2), (0, 0, 0), \left(\tfrac{1}{2}, \tfrac{-3}{2}, \tfrac{1}{2}\right), \left(\tfrac{1}{2}, 0, \tfrac{1}{2}\right)$$

This time *all* A–values of the solutions were detected by D. Recall that the classical Dixon resultant in this case was identically zero and had yielded no information on the solutions.

Example 14:

Compute D for Example 9.

Solution: Recall that the Dixon matrix is rectangular. Since the first column is not a linear combination in the remaining, the precondition holds and the Theorem applies. The product of pivots after reduction yields the Dixon (D–K–S–Y) resultant:

$$D = 48\,\left(a^3 - 12\,a + 14\,a\,c + 2\,(c\,a)^2 - 12 - 8\,a^2 + 12\,c\right)a^7$$

Let us study one last example where the precondition fails.

Example 15:

$$f = x\,y\,z$$
$$g = x^2 - z^2$$
$$h = x + y + z$$

Solution: Let us eliminate x and y. The Dixon polynomial (with auxiliary variables a and b) is,

$$(y\,z)^3 - (a\,z)^3 - a^2\,z\,x - a^2\,z\,y - (a\,z)^2\,x - a^2\,z^2$$

The Dixon matrix,

$$M = \begin{bmatrix} 0 & z^3 & 0 \\ -z^3 & 0 & -z^2 \\ -z^2 & -z & -z \end{bmatrix}$$

has determinant $\equiv 0$. Note that in this case the precondition fails (the first column is z times the last), so the Theorem does not apply. If we reduce anyway and take the product of pivots we get

$$z^6$$

which is 0 at $z = 0$. The true solution of the system is

$$(x, y, z) = (-z, 0, z)\,, \quad z \text{ any}$$

In this case only the trivial solution was detected.

Extraneous Factors; A Persistent Problem

Both the classical Dixon resultant and its K–S–Y generalization suffer from a persistent problem. The answer often has extraneous factors. In this paragraph we find the extraneous factors for a few simple examples. We compute the resultant both by the K–S–Y approach and by using Gröbner bases [Buchberger 1985] and compare answers. One of the elements of a Gröbner basis with respect to pure lexicographical order is the resultant (always). This resultant *has no extraneous factors*. Alas, the computation of such Gröbner basis is resource demanding. *Mathematica*'s `GroebnerBasis` command computes Gröbner bases with respect to pure reverse lexicographical order efficiently, but even so one can rarely compute the resultant for systems with polynomial equations of degrees > 3 with more than 3 or 4 variables.

For generic polynomials, the extraneous factor (except for signs), if none of f and g is a constant, is

$$I^{|\deg(f)-\deg(g)|}$$

where I is the leading coefficient of the highest degree polynomial (see [Kapur and Lakshman]).

Examples with Non–Generic Polynomials The extraneous factors for non–generic polynomials are a lot harder to predict as shown in Table 1. The patterns get increasingly complicated, especially for multivariate polynomials.

TABLE 1

f, g	D-Resultant	GB-Resultant	Extraneous
ax^2+bx+c Ax^2	c^2A^2	c^2A^2	A
ax^2+bx+c Bx	$-B^2ac$	cB	aB
ax^2+bx+c Ax^2+Bx	$-c\left(aB^2+cA^2-AbB\right)$	$-$Same	-1
ax^2+bx+c Ax^2+C	$-b^2AC-C^2a^2+2CacA-c^2A^2$	$-$Same	-1

TABLE 2

Polys	Eliminate	GB-Resultant	Extraneous
$2xy-ay^4$ a^2x^5y-y+1	x	$a^7y^{16}-32y+32$	a^8y^9
Same	a	$4x^7-y^6+y^5$	x^5y^4
Same	y	$a+2a^6x^{16}-6a^4x^{11}+6a^2x^6-2x$	$-a^3$
$(a-b)x^2+1$ $(b-1)x^2-b$	x	$-1+b-b^2+ab$	$(\text{Same})^2$
$4x^2+xy^2-z+\frac{1}{4}$ $2x+xy^2+\frac{1}{2}$ $x^2z-\frac{1}{2}x-y^2$	x, y	@	$\frac{1}{4096}@^2$

More complicated cases In Table 2 we display examples where the extraneous factors are harder to predict. In the fourth and fifth examples the extra factor is, up to constant, the entire Dixon resultant! The last example was used by Bruno Buchberger in several of his publications to explain how Gröbner bases are computed.

Note that,

$$@ = 64z^5 + 48z^4 + 1100z^3 + 225z^2 + 3812z - 1796$$

Conclusions

The Dixon resultant especially as modified by Kapur, Saxena and Yang is a very effective *elimination* method. Kapur and Saxena run experiments that place it well above other elimination techniques ([Kapur and Saxena]). Our limited testing certainly agrees with their claims for the topics we treated. The method is very fast. According to [Kapur and Saxena], it is up to 70 times faster than the Sparse method (which is faster than the Macaulay method) as improved by Gauss elimination, for the examples of [Kapur and Saxena]. Dixon works with parametric coefficients. The size of the Dixon matrix is small. It is smaller by a factor of $\mathbf{O}(e^{n+1})$ and $\mathbf{O}(e^{2n+1})$ (n is the number of variables) compared with the size of the Sparse and Macaulay resultant matrices, respectively. (These estimates were again taken from [Kapur and Saxena].) The method is easy to implement. Unlike Gröbner bases the algorithm here is not hard to analyze, count the number of operations, etc.. Because its core is a simple Gauss elimination.

On the negative side, the method yields extraneous factors, that are hard to identify. Also, while we can eliminate a block of variables we do not get a triangularization of the original polynomial system. This may be the *worst aspect* of the method. Triangularization is desirable, since a form of back–substitution may lead to the solution of the system. (Triangularization can be done by computing Gröbner bases with respect to pure lexicographical orders. It may be also achieved by the step–by–step use of the Sylvester resultant but in an non–automated way.) Also, the method can be slow if the polynomials are either of high degree or of a wide range of degrees. Finally, it is not clear how the method works and how sparsity is built in to it. Let us summarize:

Advantages of the Dixon method:

- 1. It is *fast*.
- 2. It eliminates a *block of variables* in only one step.
- 3. It works with *parametric coefficients*.
- 4. The size of the Dixon matrix is *small*.
- 5. It is *easy to implement*.
- 6. The algorithm is *easy to analyze*, count operations, etc..

Disadvantages of the Dixon method:

- 1. It has *extraneous factors* that are hard to identify.
- 2. It offers *no triangularization* of the polynomial system.
- 3. It can be *slow for large degree polynomials*.
- 4. It is not intuitively clear *how* is works.

4. DixonResultant and Eliminate

In this section we briefly compare our main function, `DixonResultant`, and *Mathematica*'s `Eliminate`. It *seems* that for small easily factorable polynomials `Eliminate` is at least as good as `DixonResultant`. However, if the polynomials are of higher degree, or cannot be factored, or have constant terms, `Eliminate` is either very slow or fails, while `DixonResultant` may yield an answer in seconds. This is expected since `Eliminate` is based on Gröbner bases computations which are often slow or resource demanding. On the negative side, the computation of the Dixon resultant can introduce extraneous factors that are avoided by `Eliminate` and by `GroebnerBasis`. All calculations in this section were performed on a SPARC2 with *Mathematica* 2.2.

In Examples 16-17 we eliminate x from the system $f(x) = 0$, $g(x) = 0$ for the given f and g by using a) `DixonResultant` and b) `Eliminate`. In each case we time the calculation.

Example 16:

$f(x) = (a\,x)^3 + (b\,x)^2 + c\,x + d,\, g(x) = 3\,(a\,x)^2 + 2\,b\,x + c.$

Mathematica Session:

```
F = {a x^3+b x^2 +c x + d,
     3 a x^2 + 2 b x +c};
```

```
Timing[DixonResultant[F,{x},{X}]]
```

```
                   2  2  2      3  3
{0.816667 Second, a  b  c  - 4 a  c  -

   2  3         3             4  2
4 a  b  d + 18 a  b c d - 27 a  d }
```

```
Timing[Eliminate[F=={0,0},{x}]]
```

```
                        2
{0.283333 Second, b (4 b  - 18 a c) d +

      2  2     2   2
  27 a  d  == c  (b  - 4 a c)}
```

In this example `Eliminate` was faster than `DixonResultant`.

Example 17:

$f(x) = (a\,x)^3 + (b\,x)^2 + c\,x + d,\; g(x) = (b\,x)^3 + (c\,x)^2 + a\,x + d.$

Mathematica Session:

```
F={a x^3+b x^2 +c x + d,
   b x^3 + c x^2 + a x +d};
```

```
Timing[DixonResultant[F,{x},{X}]]
                  5      2  3      4
{1.45 Second, a  d + a  b  d - a  c d -

      3            3          2    2
   3 a  b c d - a b  c d + 3 a  b c  d +

    2  3        4      3    2    2  2  2
   a  c  d - a c  d + a  b d  + a  b  d  -

        3  2    4  2      3    2
   2 a b  d  + b  d  - 3 a  c d  +

      2      2        2    2    3    2
   3 a  b c d  - 4 a b  c d  + b  c d  +

      2  2  2        2  2      3  2
   2 a  c  d  + a b c  d  - a c  d  -

    3  3      2    3        2  3    3  3
   a  d  + 3 a  b d  - 3 a b  d  + b  d }
```

```
Timing[Eliminate[F=={0,0},{x}]]
{5.31667 Second, a

        4      3    3        2
     (-a  - a b  + a  c + 3 a  b c +

        3            2      3    4
       b  c - 3 a b c  - a c  + c ) d +

        3       2  2        3    4
    (-(a  b) - a  b  + 2 a b  - b  +

          3        2            2
       3 a  c - 3 a  b c + 4 a b  c -

        3        2  2        2      3   2
       b  c - 2 a  c  - a b c  + a c ) d  +

      3      2          2    3   3
    (a  - 3 a  b + 3 a b  - b ) d  == 0}
```

In this example `Eliminate` was slower than `DixonResultant`.

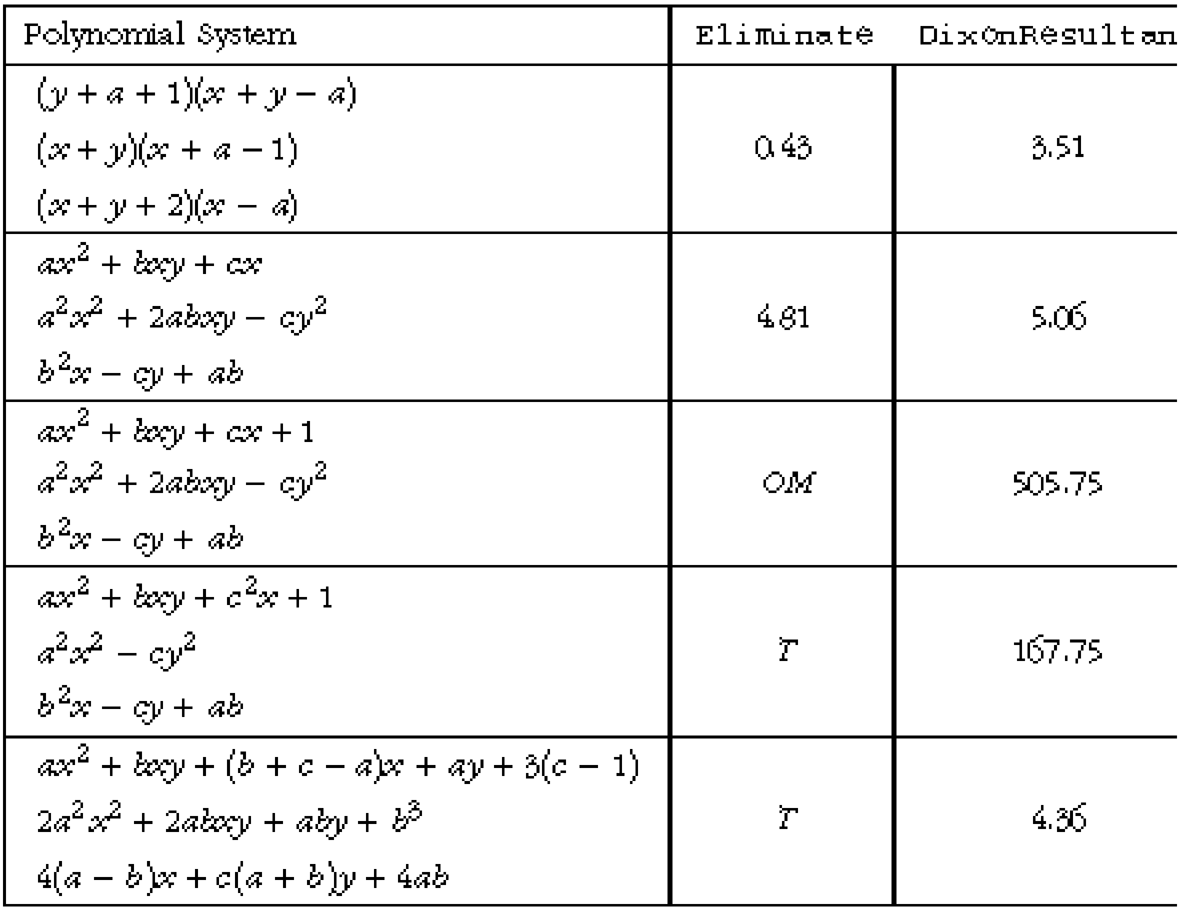
The following *Mathematica* session shows that `DixonResultant` in contrast with `Eliminate` may introduce extraneous factors. Both answers are compared with that from `GroebnerBasis`.

Mathematica Session:

```
F={x^5 + a,  x^3 -a x +a^2 x^2};

Factor[DixonResultant[F,{x},{X}]]
 3          3      6    9
a  (-1 + 6 a  + 5 a  + a )

Factor[Eliminate[F=={0,0},{x}][[1]]]
           3      6    9
a (-1 + 6 a  + 5 a  + a )

Factor[GroebnerBasis[F,{x,a}]]
            3      6    9
{a (-1 + 6 a  + 5 a  + a ),


        2    5
  a (3 a  + a  + x),


     3      6    9    3
  4 a  + 4 a  + a  + x }
```

In Table 3 we display some timings for `DixonResultant` and `Eliminate`. The few examples were meant to give the reader some assurance for our claims. In all cases *x* and *y* were eliminated. The time is in seconds truncated to 2 decimal places. *OM* stands for "Out of memory" and *T* for "terminating the job after 3 hours". The performance of the last example (whose system was taken from [Kapur and Saxena]) is perhaps the most impressive of this set.

TABLE 3



Polynomial System	Eliminate	DixonResultan
$(y+a+1)(x+y-a)$ $(x+y)(x+a-1)$ $(x+y+2)(x-a)$	0.43	3.51
$ax^2+bxy+cx$ $a^2x^2+2abxy-cy^2$ $b^2x-cy+ab$	4.81	5.06
$ax^2+bxy+cx+1$ $a^2x^2+2abxy-cy^2$ $b^2x-cy+ab$	*OM*	505.75
$ax^2+bxy+c^2x+1$ $a^2x^2-cy^2$ $b^2x-cy+ab$	*T*	167.75
$ax^2+bxy+(b+c-a)x+ay+3(c-1)$ $2a^2x^2+2abxy+aby+b^3$ $4(a-b)x+c(a+b)y+4ab$	*T*	4.36

5. DixonResultant and Resultant

In this section we briefly compare `DixonResultant` with *Mathematica*'s `Resultant` which is the Sylvester resultant. Our conclusions are that each command has its merits and limitations.

For relatively low degree polynomials with few terms `Resultant` seems to be faster. For example:

```
F1 = {d + c x + b x^2 + a x^3,
      c + 2 b x + 3 a x^2};
```

```
Timing[DixonResultant[F1,{x},{X}]]
                   2  2  2      3  3
{0.783333 Second, a  b  c  - 4 a  c  -

       2  3         3
    4 a  b  d + 18 a  b c d -

          4  2
      27 a  d }
```

```
Timing[Resultant[F1[[1]],F1[[2]],x]]
                  2  2      2  3
{0.6 Second, -(a b  c ) + 4 a  c  +

     3         2
4 a b  d - 18 a  b c d +

          3  2
      27 a  d }
```

For polynomials with more terms `Resultant` may be slower:

```
F2 = {d + c x + b x^2 + a x^3,
      a + b x + c x^2 + d x^3};
```

```
Timing[DixonResultant[F2,{x},{X}]]
                   6    4  2      3  2
{1.63333 Second, -a  + a  b  - 2 a  b  c +

      4  2    2  2  2    2  4
   2 a  c  + a  b  c  - a  c  +

      2  3        3              3
   2 a  b  d - 6 a  b c d - 2 a b  c d +

      2    2            3        4  2
   4 a  b c  d + 2 a b c  d + 3 a  d  +

    2  2  2    4  2        2    2
   a  b  d  + b  d  - 4 a b  c d  -

    2  2  2    2  2  2        3  2
   a  c  d  - b  c  d  - 2 a c  d  +

            3        2  3      2  4
   6 a b c d  + 2 b c  d  - 3 a  d  -

      2  4    2  4    6
   2 b  d  - c  d  + d }
```

```
Timing[Resultant[F2[[1]],F2[[2]], x]]
                  6    4  2      3  2
{2.86667 Second, a  - a  b  + 2 a  b  c -

      4  2    2  2  2    2  4
   2 a  c  - a  b  c  + a  c  -

      2  3        3              3
   2 a  b  d + 6 a  b c d + 2 a b  c d -

      2    2            3        4  2
   4 a  b c  d - 2 a b c  d - 3 a  d  -

    2  2  2    4  2        2    2
   a  b  d  - b  d  + 4 a b  c d  +

    2  2  2    2  2  2        3  2
   a  c  d  + b  c  d  + 2 a c  d  -

            3        2  3      2  4
   6 a b c d  - 2 b c  d  + 3 a  d  +

      2  4    2  4    6
   2 b  d  + c  d  - d }
```

To eliminate two variables from three equations requires two steps using `Resultant` and only one by using the `DixonResultant`.

```
F3 = Expand[{(y+a+1)(x+y+a),
             (x+y) (x+a-1),(y+x+2)(x-a)}];
```

```
Timing[DixonResultant[F3,{x,y},{X,Y}]]
                               2
{13.3167 Second, 8 (-2 a + 11 a  -

       3       4      5
   21 a  + 16 a  - 4 a )}
```

```
Factor[%]
{13.3167 Second,

                                 2
  8 (1 - a) (-2 + a) a (-1 + 2 a) }
```

```
Timing[
 Resultant[Resultant[F3[[1]],F3[[2]],x],
  Resultant[F3[[2]],F3[[3]],x],y]]
```

```
                   2        3
{0.15 Second, 64 a  - 608 a  +

         4          5          6
    2400 a  - 5088 a  + 6240 a  -

           7          8         9
      4416 a  + 1664 a  - 256 a }
```

```
Factor[%]
```

```
{0.15 Second,

                            3   2           3
  32 (2 - a) (-1 + a)  a  (-1 + 2 a) }
```

Note that in the last example the combination of the Sylvester resultants was still a lot faster than the one–step `DixonResultant`. However, it yielded more extraneous factors!

While in general, it is hard to tell, it seems that `Dixon- Resultant` is faster for larger systems, especially those whose equations cannot be factored. Because the Dixon matrix has smaller size that the Sylvester matrix. `Resultant` is expected to be faster when the end determinant has numerical entries. Because there is no slow symbolic Gauss elimination. On the other hand, `DixonResultant` is ideal in eliminating a block of variables in one step.

6. Symbolic Gauss Elimination

In this paragraph we show how to use the package's command `GaussElimination`. This is a symbolic Gauss elimination without scaling as discussed in Section 3. We have made this command user accessible in the package because it may be of independent interest to some readers. The code can be easily appended/-modified to allow scaling and back substitution. We exclude these extra features as they would be a diversion from our topic.

Numeric:

```
m1 = {{1,2,3},{2,2,3},{4,4,4}};
```

```
GaussElimination[m1]
```

```
{{1, 2, 3}, {0, -2, -3}, {0, 0, -2}}
```

```
m2 = {{1,2,3,4},{2,2,3,4},{4,4,4,4}};
```

```
GaussElimination[m2]
{{1, 2, 3, 4}, {0, -2, -3, -4},

      {0, 0, -2, -4}}
```

Symbolic:

```
m3 = {{x,1,0},{z,0,1},{1,y,1}};

GaussElimination[m3] // MatrixForm
x            1            0

             z
          -(-)
0            x            1

                    -1 + x y + z
                    ------------
0           0             z
```

7. Implementation in *Mathematica*

We have implemented the Kapur–Saxena–Yang algorithm in *Mathematica*. In general, the program performs quite well. However, the absence of built–in symbolic Gauss Elimination prompted us to write our own routine, which, as interpreted, is sometimes slow. We have also implemented a very fast probabilistic test for the precondition. If the test fails, the precondition fails. If the test succeeds, the precondition very likely holds. This is done by a random specialization of the parameters. The code here can be easily modified to increase the reliability of the test by using more random specializations. A non–probabilistic test for the precondition would demand the reduction of the entire Dixon matrix. This is wasted calculation in case of failure.

In the next two paragraphs we describe the program in sufficient detail so that it can be easily run by a reader with very little prior knowledge of *Mathematica*. We divide the functions into *primary* and *secondary*. We then only discuss the primary functions, their usage and examples. The secondary functions are subroutines leading to the definitions of the primary ones. Some of them may stand alone and can be useful in other applications. (See comments before the code of each of these functions.)

The Main Functions of the Program

The *Mathematica* Program is actually made into a *package* with user accessible functions the primary functions and on line help on them. The secondary functions are hidden and cannot be accessed from the package. They can, however, be copied and used independently. The entire program can be "unpacked" by removing the `BeginPackage[``Dixon`"];`, `Begin[```Private`''];`, `End[];` and `EndPackage[];` commands.

Primary Functions: `DixonSub`, `DixonPolynomial`, `DixonMatrix`, `DixonResultant`, `ClassicalDixonResultant`, `PreconditionQ`.

All primary functions have 3 arguments:

- ▪ [(a)] A list of polynomials. (The system.)
- ▪ [(b)] A list of variables. (The "unknowns", or all symbols that are not parameters.)
- ▪ [(c)] A list of auxiliary variables to substitute for the variables in the Dixon process. (This argument was added to keep the notation flexible, so that one is free to use any names for parameters.)

`DixonSub[polys, xlist, alist]` forms the (Dixon substitution) matrix by successively substituting the `a- list` into the `xlist` of `polys`.

`DixonPolynomial[polys, xlist, alist]` computes the Dixon polynomial of the system `polys` with respect to variables `xlist` and auxiliary variables `alist`.

`DixonMatrix[polys, xlist, alist]` computes the Dixon matrix of the system `polys` with respect to variables `xlist` and auxiliary variables `alist`. (NOTE: At this stage the zero rows or columns are *not* removed. This is only done for clarity. The zeros do get removed in the computation of the Dixon resultant.)

`ClassicalDixonResultant[polys, xlist, alist]` computes the classical Dixon resultant of the system `polys` with respect to variables `xlist` and auxiliary variables `alist`, as the determinant of the Dixon matrix. (NOTE: The Dixon matrix has to be square. The answer may or may not coincide with that from `DixonResultant`.)

`DixonResultant[polys, xlist, alist]` This is the **main** function of the program. It computes the D–K–S–Y resultant of the system `polys` with respect to variables `xlist` and auxiliary variables `alist`. (NOTE: Sometimes the answer needs simplification. This can be done by using the `Simplify` command.)

`PreconditionQ[polys, xlist, alist]` This is a probabilistic test for the precondition. If the test fails the precondition fails. If the test is true the precondition is very likely to be true.

Sample Session: (Input–Output)

```
GW2 = {x - a z + b,y - c z + d,
       x^2 + y^2 + z^2 - R^2}
{b + x - a z, d + y - c z,

          2     2     2     2
        -R  + x  + y  + z }
```

```
DixonSub[GW2,{x,y},{X,Y}]
```

```
{{b + x - a z, d + y - c z,

      2    2    2    2
    -R  + x  + y  + z },

  {b + X - a z, d + y - c z,

      2    2    2    2
    -R  + X  + y  + z },

  {b + X - a z, d + Y - c z,

      2    2    2    2
    -R  + X  + Y  + z }}
```

```
DixonPolynomial[GW2,{x,y},{X,Y}]
```

```
  2
-R  - b x - b X - x X - d y - d Y - y Y +

                                      2
    a x z + a X z + c y z + c Y z + z
```

```
DixonMatrix[GW2,{x,y},{X,Y}]
```

```
    2    2
{{-R  + z , -d + c z, -b + a z, 0},

   {-d + c z, -1, 0, 0},

   {-b + a z, 0, -1, 0},

   {0, 0, 0, 0}}
```

```
ClassicalDixonResultant[GW2,{x,y},{X,Y}]
```

```
0
```

```
DixonResultant[GW2,{x,y},{X,Y}]
```

```
 2    2    2
b  + d  - R  - 2 a b z - 2 c d z +

       2    2 2    2 2
      z  + a  z  + c  z
```

```
PreconditionQ[{x y z, x  - z ,x + y + z},
                    {x, y}, {X, Y}]
```

The precondition is probabilistically TRUE.

```
DixonMatrix[{x y z, x  - z , x + y + z},
            {x, y}, {X, Y}] // MatrixForm
```

```
         2
0       z


          2
-2 z    -z
```

```
PreconditionQ[{x y z, x^2 - z^2,
          x + y + z}, {x, y}, {X, Y}]
```

The precondition is FALSE.

```
DixonMatrix[{x y z, x^2 - z^2, x + y + z},
               {x, y}, {X, Y}] // MatrixForm
```

```
       3
0     z     0     0


  3           2
-z    0     -z    0


  2
-z    -z    -z    0
```

Acknowledgments

We would like to thank the Office of Naval Research for financial support. Many thanks to Deepak Kapur and Tushar Saxena for making preprints of their work available to us. We would also like to thank Janet McShane for help with the testing and commenting of our programs. She also offered valuable suggestions to improve the presentation of the material. Finally, we thank the following U. S. Naval Academy midshipmen who have pointed out typographical errors in the first version of the manuscript: Marius Harrison, C. Johnson, Terrence Nawara, Scott Powell and Richard Stephens.

References

Buchberger B., Gröbner bases: An Algorithmic method in Polynomial Ideal theory, *Multidimensional Systems Theory*, N.K. Bose, ed., D. Reidel Publ. Co., 1985.

Canny J., Generalized Characteristic Polynomials, *Journal of Symbolic Computation*, 1990, 241-250.

Cayley A., On the theory of elimination. *Cambridge and Dublin Mathematical Journal,* III, 1865, 210-270.

Chionh E., *Base points, resultants, and the implicit representation of rational Surfaces*. Ph. D. thesis, Dept. Comp. Sci., Univ. of Waterloo, 1990.

Dixon A. L., The eliminant of three quantics in two independent variables. *Proc. London Mathematical Society,* 6, 1908, 468-478.

Gantmacher F., R., *Matrix Theory*, vol. 1. Chelsea Publishing, 1959, Ch. 2.

Gelfand I., M., Kapranov M. M., and Zelevinsky A. V., *Discriminants, Resultants and Multidimensional Determinants*, Birkhäuser, Boston, 1994.

Gleason R. F., Williams R. M., A Primer On Polynomial Resultants, Technical Report, *Naval Air Development Center*, Report No. NADC-91112-50.

Kapur D., Lakshman Y. N., Elimination Methods: an Introduction. *Symbolic and Numerical Computation for Artificial Intelligence* B. Donald et. al. (eds.), Academic Press, 1992.

Kapur D., Saxena T., Comparison of Various Multivariate Resultant Formulations, *Preprint*.

Kapur D., Saxena T. and Yang L., Algebraic and Geometric Reasoning using Dixon Resultants, *Proc. ACM ISSAC*, Oxford, England, July 1994.

Knuth D. E., *The Art of Computer Programming*, v. 1-3, Addison-Wesley Publishing Co., Reading, Mass, 1969.

Macaulay F. S., *The Algebraic Theory of Modular Systems*, Cambridge Tracts in Math. and Math. Phys., 19, 1916.

Macaulay F. S., *The Algebraic Theory of Modular Systems*, Cambridge Mathematical Library, Cambridge University Press, 1994.

Mostowski A., Stark M., *Introduction to Higher Algebra*, Pergamon Press, New York, 1964.

Salmon G., *Lessons Introductory to the Modern Higher Algebra*, Chelsea Publishing Company, Bronx, NY, 1964.

van der Waerden B. L., *Modern Algebra*, volume 2, Fredric Ungar Publishing Co., New York, 1950.

About the Authors

Dr. Nakos received his Ph.D. from the Johns Hopkins University in 1985. He is currently a professor at the U. S. Naval Academy where he often uses *Mathematica*, Maple and MATLAB in the classroom. His research interests include algebraic topology and computer algebra.

Mathematics Department, M/S 9E

U.S. Naval Academy

572 Holloway Road

Chauvenet Hall

Annapolis, MD 21402-5002

gcn@sma.usna.navy.mil

Dr. Williams received his Ph.D. in Electrical Engineering from the University of Pennsylvania. He currently leads a research group using advanced types of algorithms to increase pilot awareness in confusing situations. The group is a heavy user of computer algebra systems. His research interests are in using advanced arithmetics with nonlinear algebras as architectures for novel algorithms.

Aircraft Division

Naval Air Warfare Center

Patuxent River Naval Air Station, MD

bobw@henry.nawcad.navy.mil