

# Algorithmic Search for Flexibility using Resultants of Polynomial Systems

**Robert H. Lewis**

Fordham University, New York

<http://www.bway.net/~lewis/>

**E. A. Coutsias**

University of New Mexico

Albuquerque NM, USA

## Abstract

This talk describes the recent convergence of four topics: polynomial systems, flexibility of three dimensional objects, computational chemistry, and computer algebra. We discuss a way to solve systems of polynomial equations with *resultants*. Using ideas of Bricard, we find a system of polynomial equations that models a configuration of quadrilaterals that is equivalent to some three dimensional structures. These structures are of interest in computational chemistry, as they represent molecules. We then describe and demonstrate an algorithm that examines the resultant and determines ways that the structure can be *flexible*. We review some ideas about flexibility, from Cauchy (1813) to Bricard (1893) to Connelly (1977) to Steffan (1980).

## Four themes:

- Flexibility of three dimensional shapes
  - 2D examples (triangle, quadrilateral)
  - Cauchy (1812)
  - Bricard (1897)
  - Connelly (1978)
  - Steffan (1979)
- Computational chemistry
  - peptides
  - E. Coutsias, University of New Mexico
- Systems of polynomial equations
  - Bezout (1730 - 1783)
  - Dixon (1867 - 1955)
- Computer algebra

## Systems of Polynomial Equations

Everyone knows about systems of **linear** equations:

$$3x + 4y - 2z = 0$$

$$5x + 2y - 3z = 5$$

$$-x + 3y + z = -4$$

They become matrix problems. One way to solve is **Cramer's Rule**:

$$\text{Let } M = \begin{bmatrix} 3 & 4 & -2 \\ 5 & 2 & -3 \\ -1 & 3 & 1 \end{bmatrix}$$

and

$$\text{Let } M' = \begin{bmatrix} 0 & 4 & -2 \\ 5 & 2 & -3 \\ -4 & 3 & 1 \end{bmatrix}$$

Then

$$x = \text{Det}(M') / \text{Det}(M) = (-18) / (-9) = 2$$

i.e., the system of 3 equations in 3 vars has been replaced by 1 equation in 1 var

$$A x = B \quad \text{where } B = \text{Det}(M') \text{ and } A = \text{Det}(M)$$

$y$  and  $z$  have been **eliminated**.

Same idea to solve for  $y, z$ .

Also **Gaussian elimination**, which finds  $x, y, z$  at once.

A variation is to allow **parameters**. For example, replace the equations above with

$$3x + 4y - 2z = a$$

$$5x + 2y - 3z = b$$

$$-x + 3y + z = c$$

same idea, but  $M'$  now contains  $a, b, c$ . Answer is:

$$x = (11a - 10b - 8c)/(-9)$$

**But what about polynomial equations?**

## Resultants

A method for solving systems of polynomial equations

$$f_1(x, y, z, \dots, a, b, \dots) = 0$$

$$f_2(x, y, z, \dots, a, b, \dots) = 0$$

$$f_3(x, y, z, \dots, a, b, \dots) = 0$$

.....

**First Case:** We have  $n$  equations in  $n - 1$  variables  $x, y, z, \dots$ . We also have a number of parameters  $a, b, c, \dots$

A resultant is a single polynomial derived from the system of polynomial equations that encapsulates the solution (common zero) to the system. We want to eliminate the variables and have one polynomial –

the resultant – in the parameters. For a common solution to the system, it is necessary (and often sufficient) that this resultant vanish.

**Second case:** We have  $n$  equations in  $n$  variables  $x_i, i = 1, n$ . Then one of them, say  $x_1$ , is considered a “parameter” to bring this into the previous form. The resultant is therefore an equation in that one variable; the others have been **eliminated**.

- The *Sylvester Matrix* is the best known way to compute a resultant. Others: Macaulay, sparse, etc. (a big subject.)
- Better: Bezout, Cayley, and Dixon, via Kapur, Saxena, and Yang

As refined and expanded below to avoid the **spurious factor problem**, this method has succeeded in solving many polynomial systems that others have found intractible.

### **Analogy:**

Resultants are like Cramer’s Rule.

Grobner Bases are like Gaussian elimination.

### **We will discuss:**

- The Bezout-Cayley-Dixon-KSY resultant method.
- The spurious factor problem, and a method to attack it.
- Flexible polygons and polyhedra; applications.
- Algorithmic method to detect flexibility.
- Results and demos.

# The Cayley-Dixon-Bezout-KSY Resultant Method

Here is a brief description. More details are in [KSY].

To decide if there is a common root of  $n$  polynomial equations in  $n - 1$  variables  $x, y, z, \dots$  and  $k$  parameters  $a, b, \dots$

$$f_1(x, y, z, \dots, a, b, \dots) = 0$$

$$f_2(x, y, z, \dots, a, b, \dots) = 0$$

$$f_3(x, y, z, \dots, a, b, \dots) = 0$$

.....

- Create the Bezout-Dixon matrix,  $n \times n$ , by introducing some new variables  $t_1, \dots, t_{n-1}$  into the equations in a certain way.
- Compute  $cd =$  determinant of the Bezout-Dixon matrix (a function of the new variables, variables, and parameters).
- Form a second matrix  $M$  by extracting the coefficients of  $cd$  relative to the variables and new variables. This matrix can be large; its size depends on the degrees of the polynomials  $f_i$ . Its entries are polynomials in the parameters only.
- Ideally, let  $dx =$  the determinant of  $M$ . If the system has a common solution, then  $dx = 0$ .  $dx$  is the “Dixon Resultant” and involves only the parameters.
- Problem: the second matrix need not be square, or might have  $\det = 0$  identically. Then the method appears to fail.

- However, we may continue [KSY], [BEM]: Find any maximal rank submatrix; let  $ksy =$  its determinant. Existence of a common solution implies  $ksy = 0$ .
- $ksy = 0$  is a not quite good enough; one must be aware of *spurious factors*, i.e., the true resultant is usually only a factor of  $ksy$ .

**All computations below were done with the author's computer algebra system Fermat [Lew], which excels as polynomial and matrix computations of this sort.** Recent independent tests show that Fermat is the fastest system for gcd of multivariate polynomials. Fermat is available for Windows, Mac, Linux, and Unix.

## The Spurious Factor Problem

**To emphasize the last point above:** The solution we want is an (often) small factor of the determinant  $ksy$  as described above. Often we want very much to avoid computing the entire determinant because

- (1) it is gigantic, sometimes too large to even be stored in RAM.
  - (2) even if we had it, we would want to factor it.
- We often wish to avoid simply computing the determinant of the second Dixon matrix  $M$ . Instead we systematically begin to column normalize the matrix  $M$ .
  - To avoid creating large messy denominators (rational functions) we pull out denominators from each row as soon as they arise. Then later we factor out gcds whenever possible from the numerators in each row and column.
  - We keep track of all denominators and gcds so discovered. We check often to see if some poly in the denominator list has a common gcd with some poly in the numerator list; if so we divide it out. In the end, the denominator list must be all 1. The product of the numerator list is  $\text{Det}[M]$ .

### Simple example:

Given initially

$$M_0 = \begin{pmatrix} 9 & 2 \\ 4 & 4 \end{pmatrix}$$

numerators:      denominators:

We factor a 2 out of the second column, then a 2 from the second row.

Thus:

$$M_0 = \begin{pmatrix} 9 & 1 \\ 2 & 1 \end{pmatrix}$$

numerators: 2, 2    denominators:

We change the second row by subtracting 2/9 of the first:

$$M_0 = \begin{pmatrix} 9 & 1 \\ 0 & 7/9 \end{pmatrix}$$

numerators: 2, 2    denominators:

We pull out the denominator 9 from the second row, and factor out 9 from the first column:

$$M_0 = \begin{pmatrix} 1 & 1 \\ 0 & 7 \end{pmatrix}$$

numerators: 2, 2, 9    denominators: 9

We “clean up” by dividing out the common factor of 9 from the numerator and denominator lists; any 1 that occurs may be erased and the list compacted. Since the first column is canonically simple, we are finished with one step of the algorithm, and have produced a one-smaller  $M_1$  for the next step.

$$M_1 = (7)$$

numerators: 2, 2    denominators:

The algorithm terminates by pulling out the 7:

numerators: 2, 2, 7    denominators:

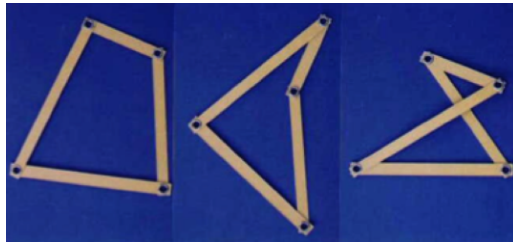
As expected (since the original matrix contained all integers) the denominator list is empty. The product of all the entries in the numerator list is the determinant, but we never needed to deal with any number larger than 9.

Notes:

- The numerator list is not unique. Had we started by factoring 4 out of the second row at the first step, the final numerators would have been 4, 9.
- If the determinant is irreducible, the final list of numerators must be trivial, i.e. just that one polynomial. But if it is not irreducible, there is no guarantee that the final list of numerators will be nontrivial.
- The “clean up” step, in which we look for a common gcd among the numerator and denominator lists, can be scheduled in various ways, and this can have a noticeable affect on those lists. Frequent cleaning up tends to keep the denominator list smaller.
- It is sometimes not necessary to run the algorithm to completion. For one reason, it may be desirable to stop at a stage  $M_i$  when the numerator list is rather large but the denominator list is empty. Perhaps a determinant method applied to  $M_i$  is now more feasible. Another reason to stop early is that by analyzing the original problem, one may recognize a factor in the numerator list as the desired answer.
- We reiterate that the total CPU time with this method is not always less than that of a standard determinant method; sometimes it is much more. But there are cases when it is very successful indeed.

## Flexibility in Polygons and Polyhedra

- A very old question.
- Consider triangles, quadrilaterals, parallelograms. Rods connected by pins that are free to pivot:



- Chemical models.
- Cauchy's paper (1812). Convex polyhedra implies rigid.
- Bricard's paper (1897). What if not convex? Octahedron. His three examples are not embeddable in 3-space; they're self-intersecting.
- So: question was left unanswered, do there exist flexible polyhedra in 3-space that "hold water"? Note: flexible means continuously movable, not just "shaky" or infinitesimally movable.
- Robert Connelly (1978) gave first example, surprising a lot of people, 18 triangular faces.
- Steffen (1979) found a flexible polyhedron with only 14 triangular faces and 9 vertices. Maksimov (1995) proved that Steffen's is the simplest possible flexible polyhedron composed of only triangles.

## Why Do We Care?

Like many aspects of 19th century mathematics, all of this was considered passe for for most of the 20th century.

**But now:**

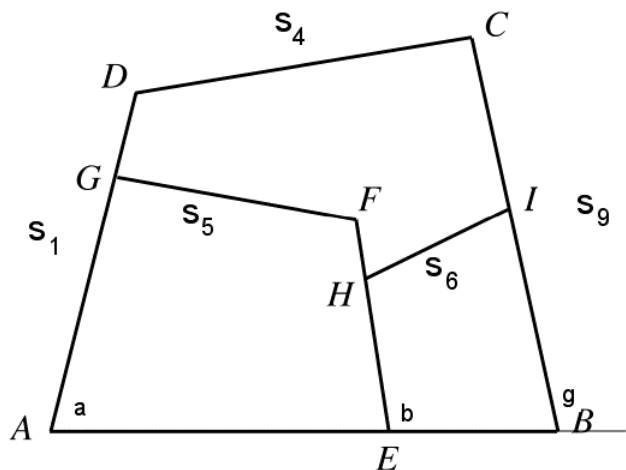
- robotics
- computational chemistry, protein folding

**Disclaimer:** I am not a chemist! Protein folding is a big field!

## Algorithmic Approach

We want to write a program that will determine conditions for a geometric figure to be flexible.

Bricard shows that his octahedron problem is equivalent to flexibility of this two-dimensional problem of three quadrilaterals:



Corners A, B, C, D, F are freely hinged. AD, DC, CB, BA, GF, FE, HI are rigid rods. The joints at G, H, I, and E can pivot.

There are three ways for this to be flexible:

- all three are parallelograms.
- two are similar, other is parallelogram.
- two are “kites” (rhomboids).

**Our Method:** Label the sides  $e, b, s_1, \dots, s_9$ . Find equations relating the sides to the three angles  $a, b, g$  at the base. Eliminate most of the variables, get resultant. Find a way to tell from the resultant if the figure is flexible.

**Solution:** Finding the equations is elementary. The variables are  $ca$ ,  $sa$ ,  $cb$ ,  $sb$ ,  $cg$ ,  $sg$  (sines and cosines). Label the vertices, write equations for each distance:

$$cx := b + s_9 * cg;$$

$$cy := s_9 * sg;$$

$$gx := s_7 * ca;$$

$$gy := s_7 * sa;$$

$$hx := e + s_8 * cb;$$

$$hy := s_8 * sb;$$

....

$$ix := b + s_3 * cg;$$

$$iy := s_3 * sg;$$

$$[d] := [( sa^2 + ca^2 - 1, \\ (dx - cx)^2 + (dy - cy)^2 - s_4^2, \\ sg^2 + cg^2 - 1, \\ (ix - hx)^2 + (iy - hy)^2 - s_6^2, \\ sb^2 + cb^2 - 1, \\ (fx - gx)^2 + (fy - gy)^2 - s_5^2 )];$$

6 equations, 6 variables, 11 parameters. Three eqs. are quite messy!

**Apply Dixon to the six equations in [d], eliminating all variables but  $ca$ .**

< first demo >

$M$  is  $44 \times 44$ , rank 29. The method described above takes 45 minutes, yields an array of numerators with more than 20 entries, ending with number of terms:

..., 190981, 190981. The last two are equal.

The determinant of the  $29 \times 29$  matrix, as the product of these, is gigantic, not computable. The resultant  $res(ca)$  has 190981 terms, degree 14 in  $ca$  (after dividing out factor of  $ca^2 + 1$ ) but only even exponents.

### **How to determine if it's flexible?**

Plugging in any set of values for the 11 sides yields an equation in the one variable  $ca$  that could be solved numerically. So what?

**If the right values are plugged in, there are continuously many values that work for  $ca$ .**

**Idea: flexibility implies infinitely many solutions for  $res(ca)$ . Thus, every coefficient in the resultant must vanish.**

That could be thought of as yielding eight new equations in the eleven sides – big mess.

< second demo >

## Algorithm:

- (1) Kill each coefficient of  $ca$  in turn, starting at the highest, the coef of  $ca^{14}$ . Do so by looking for contents, linear parameters to solve for, or difference of squares. When a substitution is found, plug it in, reducing the degree of  $res$ . Continue. We think of this as a procedure  $Solve(res, ca)$ .
- (2) If none of the ideas in (1) works, try to kill the coefficient by invoking the entire algorithm on it, relative to each variable in the coefficient. So, this subprocedure of  $Solve$  works by calling  $Solve(cof, s_i)$  within a loop.
- (3) Use a suitable data structure to keep track of all the substitutions.

## Example:

$$(s_9 * s_8 - s_7 * s_6)ca^2 + (s_4^2 - s_3^2)ca + s_8 - s_6. \text{ irreducible.}$$

A solution is  $s_9 = s_7, s_8 = s_6, s_4 = s_3$

There is no guarantee this will work.

[It does; finds all three known solutions. Maximum stack depth is 28.]

## Future Work:

By writing everything in terms of the tangent of half-angles, we can reduce the problem from six to three equations:

$$a_1 * t_1^2 * t_2^2 + b_1 * t_1^2 + 2c_1 * t_1 * t_2 + d_1 * t_2^2 + e_1,$$

$$a_2 * t_2^2 * t_3^2 + b_2 * t_2^2 + 2c_2 * t_2 * t_3 + d_2 * t_3^2 + e_2,$$

$$a_3 * t_1^2 * t_3^2 + b_3 * t_1^2 + 2c_3 * t_1 * t_3 + d_3 * t_3^2 + e_3$$

The  $t_i$  are the half-angle tangents of the three base angles of before. The parameters  $a_i, b_i, \dots$  are quadratic functions of the eleven sides. For example,

$$a_1 = e^2 + s_2^2 + s_7^2 - s_5^2 - 2e * s_2 + 2e * s_7 - 2s_2 * s_7$$

which is a product of two linear terms.

**Idea:** Find the resultant of this system of three. It has 5685 terms. Apply algorithm as before. Should be faster?

**No!** Now we must try things like  $a_1 = 0$  or  $a_1 = -d_3 - e_2$ . Those were never used when the parameters were actually the sides. Tricky.

## References:

[B] Bricard, Raoul, Memoire sur la theorie de loctaedre articule, J. Math. Pures Appl. 3 (1897), 113-150 ( English translation: <http://www.math.unm.edu/vageli/papers/bricard.pdf>).

[BEM] L. Buse, M. Elkadi, and B. Mourrain, Generalized resultants over unirational algebraic varieties. J. Symbolic Comp. **29** (2000), p. 515-526.

[C1] E. A. Coutsias, C. Seok, M. J. Wester and K.A. Dill, Resultants and Loop Closure, International Journal of Quantum Chemistry 106 (2005), no. (1), 176-189.

[C2] E. A. Coutsias, C. Seok, M.J. Jacobson, K.A. Dill, A Kinematic View of Loop Closure, Journal of Computational Chemistry 25 (2004), no. 4, 510 - 528.

[KSY] D. Kapur, T. Saxena, and L. Yang, Algebraic and geometric reasoning using Dixon resultants. In: Proc. of the International Symposium on Symbolic and Algebraic Computation. A.C.M. Press (1994).

[Lew] Robert H. Lewis, Computer algebra system *Fermat*.  
[www.bway.net/~lewis/](http://www.bway.net/~lewis/)

[LB] R. Lewis and S. Bridgett, Conic Tangency Equations Arising from Apollonius Problems in Biochemistry. Mathematics and Computers in Simulation 61(2) (2003) p. 101-114.

[LS] R. Lewis and P. Stiller, Solving the Recognition Problem for Six Lines Using the Dixon Resultant. Mathematics and Computers in Simulation **49** (1999).