Algorithmic Search for Flexibility Using Resultants of Polynomial Systems

Robert H. Lewis[1] and Evangelos A. Coutsias[2]

[1] Fordham University, New York, NY 10458, USA
[2] University of New Mexico, Albuquerque, NM 87131, USA

Abstract. This paper 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 an algorithm that examines the resultant and determines ways that the structure can be *flexible*.

1 Introduction

This project results from the recent convergence of four topics: systems of polynomial equations, flexibility of three dimensional objects, computational chemistry, and computer algebra.

Protein folding has been a major research topic in computational chemistry for a number of years [9]. Proteins are long molecular chains. Proteins form as flexible chains but they quickly fold into shapes that are rigidified by the formation of additional bonds. However, they retain flexibility in certain regions, which is essential for performing their various functions [24]. As macromolecules, composed of relatively heavy atoms (Carbon, Oxygen, Nitrogen, etc.) their conformational problem is modeled in terms of frameworks, i.e., systems of ideal points (atoms) connected by rigid rods (molecular bonds), with fixed angles (the bond angles, determined by molecular orbitals) but flexible torsions (the solid angles formed by successive bonded quartets) [13]. Simple examples are easily built using plastic balls and sticks.

In 1812, Cauchy considered flexibility of three dimensional polyhedra (think of a geodesic dome) where each joint can pivot or hinge. He proved that a convex polyhedron with invariant facets must be rigid [4]. Bricard [2], in response to a question posed in 1895 by C. Stephanos [23], gave the geometric conditions under which an octahedron may be flexible. The Bricard octahedra, however, besides being non-convex are also non-embeddable in 3-dimensional space as they possess intercrossing facets. A genuine, embeddable, flexible polyhedra with rigid facets was found by Connelly in 1978 [5], and soon models appeared of a simple flexible structure [8]. It is very enlightening to hold one of these and feel it move.

As the facets are triangular, they are rigid – unlike quadrilaterals which are inherently flexible; think for example of a cube as compared to a tetrahedron. Thus the deformability is seen in terms of changes of the dihedrals formed by these facets about the edges of the polyhedron. Since the underlying description is in terms of quadratic distance constraints, expressing the conformational problem of the polyhedron in terms of cosines and sines (or half-tangents) of these dihedral angles results in systems of polynomial equations, quadratic in each of these variables, and in which the edge lengths enter as parameters. A polyhedron composed of triangular facets is subject to enough such constraints that according to classical results on rigidity that date back to Lagrange [12] and Maxwell [18] it should be rigid – generically at least! The polynomial system describing these conformations must therefore generically possess a discrete solution set. However, when conditions for flexibility are met, the solution set must acquire components of nonzero dimensionality (so called *continuous* components). Thus, the problem of detecting flexibility amounts to being able to identify conditions in the parameters for which a system of n polynomials in n variables drops in *rank* [22].

Here we present a new approach to understanding flexibility, using resultants and symbolic computation. The geometry of the object or molecule is described by a set of multivariate polynomial equations. Solving a system of multivariate polynomial equations is a classic, difficult problem. The approach via resultants was pioneered by Bezout [1], Dixon [10], and others. The resultant *res* appears as a factor of the determinant *det* of a matrix containing multivariate polynomials. But often *det* is too large to compute or factor, even though *res* is relatively small. We will describe a heuristic that overcomes the problem here, and in other cases [15]. Once we have the resultant, we describe an algorithm that examines the resultant and determines ways that the structure can be flexible. We discover in this way the conditions of flexibility for an arrangement of quadrilaterals in [2]. This system was posed by Bricard as an easily realizable, mathematically equivalent alternative to his flexible octahedra.

All computations below were done with Lewis's computer algebra system *Fermat* [14], which excels as polynomial and matrix computations [19]. We used a 1.8 ghz Macintosh G5, new in 2003.

2   Accelerating the Dixon Resultant

The Dixon Resultant method [10], following an idea of Bezout [1] and modified by Kapur et. al. [11], is presented in [11], [3], and [16]. Given a system of n polynomial equations

$$f_i(x_1, x_2, x_3, \ldots, a, b, \ldots) = 0, \; i = 1, \ldots, n$$

in $n - 1$ variables x_i and a number of parameters $a, b, \ldots$, the method computes its *resultant*, i.e. a single polynomial in the parameters encapsulating the solution (common zero) to the system. A common variation is to have n equations in n

variables. Then one of them, say x_1, is considered a parameter to bring this into the previous form. In either case, the other variables have been *eliminated*.

In this paper all polynomials have coefficients in the ring of integers Z and the solutions are in the field C of complex numbers. All computations are exact.

The basic Bezout-Dixon idea is to construct a square matrix M whose determinant $det \neq 0$ is a multiple of the resultant. The factors of det that are not the resultant are called the *spurious factors*, and their product is sometimes called *the spurious factor*.

The naive way to proceed is to compute det, factor it, and separate the spurious factor from the actual resultant. Deciding what is spurious and what is the resultant is not always simple. However, when the original problem is based on geometry (as is the present problem) and one knows that the solution set is discrete, the resultant must involve all the parameters. Typically, many factors of det do not. Also, it is usually easy to simply plug in a known numerical solution and see which factor it satisfies. (The Dixon method is not guaranteed to work if the solution set is infinite; see [3]).

A graver problem is that the determinant may be so large as to be impractical or even impossible to compute, even though the resultant is relatively small; the spurious factor is huge. Further, the determinant may be so large that factoring it is impractical.

To overcome these problems, Lewis has developed several heuristic methods [15]. A method called EDF, Early Discovery of Factors, makes use of the existence of spurious factors. We reproduce it here for the convenience of the reader. By elementary row and column manipulations (Gaussian elimination), it discovers probable factors of det and plucks them out of $M_0 \equiv M$. Any denominators that form in the matrix are plucked out. This produces a smaller matrix M_1 still with polynomial entries, and a list of discovered numerators and denominators. Iterate. Here is a summary:

Algorithm EDF: Variation of Gaussian elimination to discover factors of the determinant.

Input: square matrix M. Let $n =$ number of rows of M. All entries of M are polynomials. Assume $Det(M) \neq 0$.

Output: list of polynomials whose product is $Det(M)$.

```
Let num be a list of numerator polynomials, initially empty.
Let den be a list of denominator polynomials, initially empty.
Let M[i] be the submatrix of M from entry (i,i) down to (n,n).

for i = 1 to n do
  for j = i to n do
    Find the GCD of all entries in row j.
    Factor it out; append it to num.
    Find the GCD of all entries in column j.
    Factor it out; append it to num.
```

```
    endfor;
    Find a good pivot in M[i]. Move it into position (i,i).
    Do one step of Gaussian elimination using the pivot.
    for j = i to n do
      Find the LCM of all denominators in row j.
      Multiply it by row j; append it to den.
    endfor;
    if desired(i)
      { every fifth or tenth row is  reasonable. }
      Look for common factors in num and den lists.
      Consolidate num, den by dividing out such factors.
    endif
endfor.

Consolidate num and den lists. den should be empty.
Output num.
```

Notes:

- The resultant is usually in the numerator list. It is often the last entry. The remaining entries in the numerator list are then the spurious factors. Almost always the numerator list is long and interesting.
- 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 "consolidate" 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 performance. There is no obviously best method. Experiments show that consolidation should be done every five to ten rows.
- The definition of "good pivot" is also not rigorous. Basically, one wants the "smallest" nonzero entry, so that the ensuing rational function arithmetic yields "small" entries in the rest of the matrix. Heuristics can be written depending on the number of terms, number of variables, their degree, and the size of the numerical coefficients.

EDF can work efficiently because *det* usually has many factors. This is a bad way to compute the determinant of a random matrix. But the Dixon matrices M are far from random. The total CPU time with this method is not always less than that of a standard determinant method; sometimes it is much more. We will see below that this technique can be dramatically successful. For other examples, see [15].

3   Flexibility of Polyhedra, and Computational Chemistry

This is a very old question. In two dimensions we may consider triangles, quadrilaterals, parallelograms, or more general n-gons. We imagine they are made of

rigid rods connected by pins that are free to pivot at ideal joints. Triangles are obviously rigid, and any quadrilateral is flexible, though parallelograms are more flexible in that the angle at any two sides can take on any value. In three dimensions we likewise consider elementary chemical models, or polyhedra with triangular faces (like a geodesic dome).

In 1812 Cauchy [4] proved that convex polyhedra must be rigid. In 1897 Bricard [2] investigated nonconvex octahedra and found three ways they could be flexible. However, his examples are not embeddable in 3-space; they are self-intersecting. So the question was left unanswered, do there exist flexible polyhedra in 3-space? Surprisingly, in 1978 Robert Connelly [5] gave an example, with 18 triangular faces. Steffen [17] found a flexible polyhedron with only 14 triangular faces and 9 vertices. Maksimov [17] proved that Steffen's is the simplest possible flexible polyhedron composed of only triangles. See also [21]. It was later proved by Sabitov [20] that the volume of any such flexible polyhedron is invariant as it flexes.

Coutsias et. al. [6] [7] showed that Bricard's ideas have application to computational chemistry, and generalized them to solve the problem of Loop Closure, leading to a general algorithm for computing localized torsional deformations of molecular loops in proteins and nucleic acids. Bricard [2] states that the conformational problem of the octahedron is mathematically analogous to that of a system of articulated quadrilaterals. Such systems were important in the late 19th century, with applications to the transfer of force or motion in mechanical devices like sewing machines and automobiles, and today to robotic manipulators. While generically flexible systems where the number of variables exceeds the number of constraints are ubiquitous, here we are concerned with *non-generic* flexibility, where the number of variables and constraints are equal. Then flexibility is encountered only when certain conditions hold among the parameters of the system.

In particular, we shall consider the flexibility of the planar group of three quadrilaterals in figure 1. 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.

4 Algorithmic Approach

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

- Label the sides $e, b, s_1, \ldots, s_9$ (see figure 1).
- With elementary analytic geometry find six equations relating the sides to the three angles *alpha, beta, gamma* at the base.
- Eliminate most of the variables; compute the resultant.
- Find a way to tell from the resultant when the figure is flexible.

4.1 The equations

Finding the equations is elementary. The variables are *ca*, *sa*, *cb*, *sb*, *cg*, *sg* (sines and cosines of base angles). There are eleven parameters, $e, b, s_1, \ldots, s_9$.
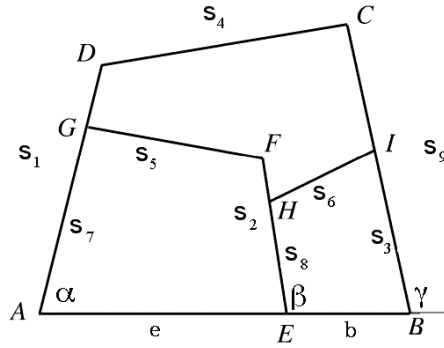
Fig. 1. Configuration of three quadrilaterals from Bricard [2].

Expressions for each x and y coordinate for each point $C, G, H, \ldots$ are easily found:

$cx := b + e + s_9 * cg;$
$cy := s_9 * sg;$
$gx := s_7 * ca;$
$gy := s_7 * sa;$
$hx := e + s_8 * cb;$
$hy := s_8 * sb;$
....

To form the six equations set each of these to 0 (the last three are just distances in the plane):

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

We have six equations, six variables, eleven parameters. The latter three equations are actually quite messy because the expressions $cx, cy, hx, hy, \ldots$ must be expanded in terms of the variables and parameters.

4.2 Solving the system with the EDF method

We now apply the Dixon resultant method to the six equations, eliminating all variables but ca. The resultant will be a function of ca and the eleven parameters.

The Dixon matrix M is 29×29. The EDF method described in section 2 takes 62 minutes on the desktop Macintosh computer, and yields a list of numerators with more than 80 entries. The last two have 275808 and 312783 terms. Dividing each by an easily found content yields the same polynomial of 201694 terms (recall that the *content* of a polynomial $f(x, \ldots)$ relative to x is the gcd of all the coefficients of $x^k, k = 0, \ldots, degree(f, x)$). This polynomial, call it

res, is easily checked to be irreducible and is the resultant. It has degree 7 in *ca*. The product of the other terms in the list has 35000 terms. The determinant of M, as the product of all these, is truly gigantic, probably not computable on any computer. Thanks to EDF, there is no need to compute it.

4.3   Determining flexibility

Classically, one would use the resultant by plugging in numerical values for the eleven parameters. That yields an equation in only the one variable *ca* that could be solved numerically. But how to detect flexibility in the quadrilaterals? Answer: If the parameters have the right relations to each other to produce flexibility, there are infinitely many values that work for *ca*. But *res* is a polynomial, so the only way to have that many roots is for every coefficient relative to *ca* in the resultant to vanish. That can be thought of as yielding eight new equations in the eleven sides, but those equations would be too complicated to use (their number of terms ranges from 198 to over 53000),

Instead, we have developed an algorithm *Solve* to produce a list of relations among the sides that will kill all eight coefficients. If the algorithm is good enough, any relationship producing flexibility will be on this list. However, it is not clear that all relationships on the list must produce flexibility. The issue of converse implication with the Dixon resultant is discussed in [3]. If a set of relations force all eight coefficients of *res* to vanish, when these relations are plugged into the original six equations, there is a positive-dimensional component to the solution set (i.e., a continuous family of solutions). But this may not be a geometrically meaningful situation. We will see an example of this later.

To describe the algorithm, let us first rename $e \equiv s_{10}, b \equiv s_{11}$. We present *Solve* in terms of general inputs f and x. f is a polynomial in x and N parameters s_i.

Algorithm *Solve(f, x):* Given a polynomial f in a variable x and a number N of parameters s_i, find relations on the parameters that make the entire polynomial vanish. Our problem is solved by invoking $Solve(res, ca)$, $N = 11$.

Outline:

1. Kill each coefficient *coef* of x in turn, starting at the highest degree. Do so by looking for contents, linear parameters to solve for, or a difference of squares. When a substitution is found, plug it in, reducing the degree of f. Continue.
2. Also try to kill the coefficient *coef* by invoking the entire algorithm on it, relative to each variable in *coef*. So, this step of *Solve* works by calling $Solve(coef, s_i)$ within a loop.
3. Use suitable data structures to keep track of all the substitutions.

Here is a simple example. If *res* were $(s_9 * s_8 - s_7 * s_6)ca^2 + (s_4{}^2 - s_3{}^2)ca + s_8 - s_6$, one solution would be the collection (or *table*) of the three relations $s_9 = s_7, s_8 = s_6, s_4 = s_3$.

The relations may be described as follows: Partition the set of N parameters into nonempty subsets $X = \{x_i\}_{i=1}^n$, $Y = \{y_j\}_{j=1}^m$, $n + m = N$. Each relation is

an equation $y_j = g_j(x_{i_1}, x_{i_2}, \ldots)$ where g_j is a rational function. A collection of m of these for $j = 1, \ldots, m$ is a *solution table* if f evaluated at them all is 0. In the example above $X = \{s_3, s_6, s_7\}$ and $Y = \{s_4, s_8, s_9\}$.

Input: multivariate polynomial f in a primary variable x and N parameters s_i.

Output: list of solution tables, as defined above.

```
Let lst be the output list of solution tables, initially empty.
Let cc = leading coefficient in f(x).
{ cc is a polynomial in the parameters. }
Get factors of cc by finding content relative to all s_i.
{ Optionally, also do more complete factoring. }
Use the factors to produce a list ls of s_j to solve for:
    Within each factor, find all s_j of degree 1.
    Look for factors that are differences of squares.
{ Note: the list ls may be empty. }
while not done with the list ls do
    Solve for s_j, getting s_j = g(s_i_1, s_i_2, ...).
    Use the relation g to replace s_j in f.
    This yields fj(x), say, of lower degree.
    Compute lstj = Solve(fj,x).
    if lstj is not empty
        Insert the relation s_j = g into each table of lstj.
        Append the resulting lstj to lst.
    endif;
enddo;
for every s_i in cc not in ls do
    Compute lsti = Solve(cc, s_i).
    for every table T in lsti do
        plug the relations of T into f, yielding ft(x).
        Compute lstt = Solve(ft, x).
        Combine T with each table in lstt.
        Append the resulting lstt to lst.
    endfor;
endfor;

Look for duplicates in lst; "clean up" lst.
Output lst.
```

In creating the relations $s_j = g(s_{i_1}, s_{i_2}, \ldots)$, we reject any relation of the form $s_j = 0$, or in which all the numerical coefficients in g are negative, such as $s_3 = -s_2 - s_5 s_7$. Since the s_i are lengths on a geometric figure, these are meaningless.

Details of combining and managing the table lists are left to the programmer.

There is no guarantee this method will work. However it does, in about 3 minutes. It finds the two ways to make the quadrilaterals flexible: all three

are parallelograms, and one is a parallelogram and the other two are similar. Interestingly, it also finds a degenerate yet still meaningful arrangement when two of them are rhomboids.

For example, the case where the lower left quadrilateral is a parallelogram and the other two are similar is expressed by the table

$$s_9 = s_3(e + b)/b,$$
$$s_8 = s1\ b/(e + b),$$
$$s_7 = s_2,$$
$$s_6 = s_4\ b/(e + b),$$
$$s_5 = e$$

All three parallelograms is

$$s_9 = s_1,$$
$$s_4 = e + b,$$
$$s_7 = s_2,$$
$$s_5 = e$$
$$s_6 = b,$$
$$s_8 = s_3$$

One case of two rhomboids is

$$s_9 = s_4,$$
$$s_7 = e,$$
$$s_1 = e + b,$$
$$s_5 = s_2$$

Let us look more closely at the two rhomboids case. If those relations are substituted into the original six equations, one equation becomes extremely simple: $(e + s2 * cb)(1 - ca) - s2 * sa * sb$. We are led to setting $ca = 1$ and $sa = 0$, which kills three (not just two) of the six equations. Three remain, in the four variables sg, cg, sb, cb:

$$sg^2 + cg^2 - 1,$$
$$sb^2 + cb^2 - 1,$$
$$2*s3*s8*sb*sg + 2*s3*s8*cb*cg - 2*b*s3*cg + 2*b*s8*cb - s8^2 + s6^2 - s3^2 - b^2$$

We expect therefore an infinite family of solutions, which can be demonstrated by numerical experiments, or by computing the (bi-variate) resultant of the three equations, eliminating sg and sb:

$8*b*s3^2*s8*cb*cg^2 - 4*s3^2*s8^2*cg^2 - 4*b^2*s3^2*cg^2 - 8*b*s3*s8^2*cb^2*cg + 4*s3*s8^3*cb*cg - 4*s3*s6^2*s8*cb*cg + 4*s3^3*s8*cb*cg + 12*b^2*s3*s8*cb*cg - 4*b*s3*s8^2*cg + 4*b*s3*s6^2*cg - 4*b*s3^3*cg - 4*b^3*s3*cg - 4*s3^2*s8^2*cb^2 - 4*b^2*s8^2*cb^2 + 4*b*s8^3*cb - 4*b*s6^2*s8*cb + 4*b*s3^2*s8*cb + 4*b^3*s8*cb - s8^4 + 2*s6^2*s8^2 + 2*s3^2*s8^2 - 2*b^2*s8^2 - s6^4 + 2*s3^2*s6^2 + 2*b^2*s6^2 - s3^4 - 2*b^2*s3^2 - b^4.$

The choice $ca = 1$, $sa = 0$ is actually geometrically meaningful. It corresponds to two degenerate rhomboids, with the points G, E and B, D falling on top of each other. Thus, the flexibility in this case is just the flexibility of

the single quadrilateral $EBIH$. The original six equations do indeed fit this picture. See figure 2. If, on the other hand, we exclude the vanishing of the angle $\alpha = \angle BAD$, then it can be shown that the resulting problem has a resultant of lower degree, leading to additional, non-degenerate, discrete conformations. The identical vanishing of the resultant of the full problem for this case would completely mask the existence of these discrete components of the solution set.
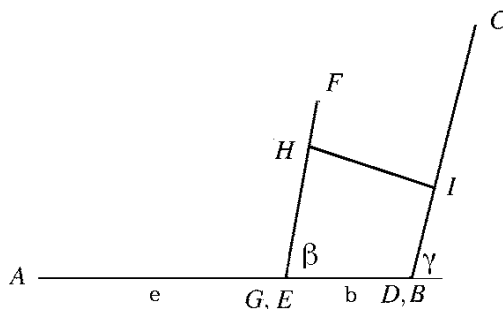

Fig. 2. Degenerate rhomboids.

4.4 Future work

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

$$a_1 * t_1^2 * t_2^2 + b_1 * t_1^2 + 2c_1 * t_1 * t_2 + d_1 * t_2^2 + e_1 = 0,$$
$$a_2 * t_2^2 * t_3^2 + b_2 * t_2^2 + 2c_2 * t_2 * t_3 + d_2 * t_3^2 + e_2 = 0,$$
$$a_3 * t_1^2 * t_3^2 + b_3 * t_1^2 + 2c_3 * t_1 * t_3 + d_3 * t_3^2 + e_3 = 0$$

The t_i are the half-angle tangents of the three base angles. As before, these equations result from elementary analytic geometry. The parameters $a_i, b_i, \ldots$ 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. This is the form of the equations as derived by Bricard.

The resultant of this system has 5685 terms. Shall we apply our flexibility searching algorithm as before? It is more subtle, as now we must try relations like $a_1 = 0$ or $a_1 = -d_3 - e_2$. When the parameters were actually the sides, substitutions like this made no sense and were excluded, thereby streamlining the search. We have recently modified algorithm *Solve* to consider these cases, and

the work is ongoing. Success on this set of three equations would be significant because the identical set of equations arises in other contexts, and a variant (including also the "missing" terms, such as $t_1^2 * t_2$, t_3, ...) gives the conformational equations of a protein or nucleic acid backbone [6], [7].

References

1. Bikker, P. On Bezout's method for computing the resultant. RISC-Linz Report Series 1995. Johannes Kepler University A-4040 Linz, Austria.
2. Bricard, Raoul, Memoire sur la theorie de loctaedre articule, J. Math. Pures Appl. 3 (1897), p. 113 - 150 (English translation: http://www.math.unm.edu/vageli/papers/bricard.pdf).
3. Buse, L., M. Elkadi, and B. Mourrain, Generalized resultants over unirational algebraic varieties. J. Symbolic Comp. **29** (2000), p. 515 - 526.
4. Cauchy, A. L. Sur les polygones et les polyhedres. Second Memoire. J. Acole Polytechn. **9** (1813), pp. 8.
5. Connelly, R. A counterexample to the rigidity conjecture for polyhedra. Publ. Math. I. H. E. S. **47** (1978), p. 333 - 338.
6. Coutsias, E. A., C. Seok, M. J. Wester and K.A. Dill, Resultants and loop closure, International Journal of Quantum Chemistry 106 (2005), no. (1), p. 176 - 189.
7. Coutsias, E. A., C. Seok, M.J. Jacobson, K.A. Dill, A Kinematic view of loop closure, Journal of Computational Chemistry 25 (2004), no. 4, p. 510 - 528.
8. Cromwell, P. R. *Polyhedra*. New York: Cambridge University Press, 1997. p. 222 - 224.
9. Dill, K. A. and H. S. Chan, From Levinthal to pathways to funnels: The "new view" of protein folding kinematics, Nat. Struct. Biol., 4:10-19, 1997.
10. Dixon, A. L. The eliminant of three quantics in two independent variables. Proc. London Math. Society, **6** (1908) p. 468 - 478.
11. Kapur, D., 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).
12. Lagrange, Joseph-Louis, *Mécanique Analytique*, Paris (1788).
13. Leach, A., *Molecular Modeling and Simulation*, Cambridge, 2004.
14. Lewis, Robert H. Computer algebra system *Fermat*. www.bway.net/~lewis/
15. Lewis, Robert H. Heuristics to accelerate the Dixon resultant. to appear in Mathematics and Computers in Simulation.
16. Lewis, R. and P. Stiller, Solving the recognition problem for six lines using the Dixon resultant. Mathematics and Computers in Simulation **49** (1999).
17. Maksimov, I. G. Polyhedra with bendings and Riemann surfaces. Uspekhi Matemat. Nauk **50** (1995) p. 821 - 823.
18. Maxwell, James Clerk, On the calculation of equilibrium and stiffness of frames, Phil. Mag., **27** (1864) p. 294 - 299.
19. Robertz, D. and V. Gerdt, comparison of software systems at http://www.bway.net/~lewis/fermat/gcdcomp. (2004).
20. Sabitov, I. Kh., A proof of the "bellows" conjecture for polyhedra of low topological genus, Dokl. Acad. Nauk, 358 **(6)** (1998) p. 743 - 746.
21. Stachel, H. Higher order flexibility of octahedra, Period. Math. Hung. **39** (1999) p. 225 - 240.

22. Sommese, A. J. and C. W. Wampler II, *The Numerical Solution of Systems of Polynomial arising in Engineering and Science*; World Scientific: New York, 2005.
23. Stephanos, Cyparissos, problem 376, L' Intermédiaire des Mathématiciens, **2** (1895) 243 - 244.
24. Thorpe, M., M. Lei, A.J. Rader, D.J. Jacobs, L. Kuhn. Protein flexibility and dynamics using constraint theory. Journal of Molecular Graphics and Modelling **19** (2001) p. 60 - 69.