J. Symbolic Computation (1993) **15**, 99–122

MultiPolynomial Resultant Algorithms

DINESH MANOCHA †
Department of Computer Science, Sitterson Hall
University of North Carolina
Chapel Hill, NC 27599
manocha@cs.unc.edu
JOHN F. CANNY ‡
Computer Science Division
University of California
Berkeley, CA 94720
jfc@cs.berkeley.edu

(Received 02 June 1991)

Computational methods for manipulating sets of polynomial equations are becoming of greater importance due to the use of polynomial equations in various applications. In some cases we need to eliminate variables from a given system of polynomial equations to obtain a "symbolically smaller" system, while in others we desire to compute the numerical solutions of non-linear polynomial equations. Recently, the techniques of Gröbner bases and polynomial continuation have received much attention as algorithmic methods for these symbolic and numeric applications. When it comes to practice, these methods are slow and not effective for a variety of reasons. In this paper we present efficient techniques for computing multipolynomial resultant algorithms and show their effectiveness for manipulating system of polynomial equations. In particular, we present efficient algorithms for computing the resultant of a system of polynomial equations (whose coefficients may be symbolic variables). The algorithm can also be used for interpolating polynomials from their values and expanding symbolic determinants. Furthermore, it is possible to come up with tight bounds on the running time and storage requirements of the algorithm. Finally, we use the symbolic elimination algorithm to compute the real or complex solutions of non-linear polynomial equations. It reduces the problem to finding roots of univariate polynomials. We also discuss the implementation of these algorithms and discuss their performance on some applications.

† Supported in part by IBM Graduate Fellowship, David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (#IRI-8958577)
‡ Supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (#IRI-8958577)

0747–7171/90/000000 + 00 $03.00/0 © 1994 Academic Press Limited

1. Introduction

Finding the solution to a system of non-linear polynomial equations over a given field is a classical and fundamental problem in computational algebra. This problem arises in symbolic and numeric techniques used to manipulate sets of polynomial equations. Many applications in computer algebra, robotics, geometric and solid modeling use sets of polynomial equations for object representation (as a semi-algebraic set) and for defining constraints (as an algebraic set) (Manocha, 1992). The main operations in these applications can be classified into two types: simultaneous elimination of one or more variables from a given set of polynomial equations to obtain a "symbolically smaller" system and computing the numeric solutions of a system of polynomial equations. Elimination theory, a branch of classical algebraic geometry, investigates the condition under which sets of polynomials have common roots. Its results were known at least a century ago (Macaulay, 64; Salmon, 1885; Waerden 1950) and still appear in modern treatments of algebraic geometry, although in non-constructive form. The main result is the construction of a single resultant polynomial of n homogeneous polynomial equations in n unknowns, such that the vanishing of the resultant is a necessary and sufficient condition for the given system to have a non-trivial solution. We call this resultant the *Multipolynomial Resultant*† of the given system of polynomial equations. The multipolynomial resultant of the system of polynomial equations can be used for eliminating the variables and computing the numeric solutions of a given system of polynomial equations.

There are two other algorithmic approaches known in the literature for eliminating variables from a system of equations. The first one is based on polynomial ideal theory and generates special bases for polynomial ideals, called Grőbner bases. The algorithm of computing Grőbner bases is due to Buchberger and is surveyed in (Buchberger, 1985; Buchberger 1989). Its applications include ideal membership testing and performing algebraic operations like union, intersection on ideals in addition to eliminating a set of variables or computing the numerical solutions of a system of polynomial equations. The second approach for variable elimination has been developed by Wu Wen-Tsün (Wu, 1984) using an idea proposed by Ritt (Ritt, 1950). This approach is based on Ritt's characteristic set construction and successfully applied to automated geometry theorem proving by Wu. It is referred as the Ritt-Wu's algorithm (Gao and Chou, 1990).

One of the main difficulties in using Grőbner bases or Ritt-Wu's algorithm is that the method may be slow for even small problems. In the worst case, the running time of Grőbner bases computation can be doubly exponential in the number of variables (Mayr and Meyer, 1982). Even in special cases where this doubly exponential behavior is not observed, deriving tight upper bounds on the method's running time is difficult. This behavior is also observed in practice. In particular, two algorithms for eliminating variables using Grőbner bases have been described in (Hoffmann, 1990a; Hoffmann, 1990b). In the first algorithm a Grőbner base for the ideal, generated by given polynomial equations with respect to a term ordering, defined as *elimination order*, is constructed. The second algorithm evolved from a basis conversion method and is considered to be more efficient than the first one. Moreover, the second algorithm had been considered to be the fastest elimination method for geometric applications, among all methods implemented till that time (Hoffmann, 1990b). It has been applied for implicitizing rational parametric surfaces for geometric modeling applications. To implicitize rational parametric bicubic patches,

† Other authors have used the term multiequational resultants (Bajaj et. al., 1988)

the algorithm takes about 10^5 seconds on a Symbolics 3650, which would be considered impractical for geometric modeling applications. On the same example Ritt-Wu's algorithm takes about $28,000$ seconds on a Sun-3 (Gao and Chou, 1990). Many other examples of symbolic elimination using Grőbner bases are highlighted in (Kalkbrener, 1991) and the running times presented are too slow for geometric applications. As a result, no efficient algorithms or implementations are known for symbolic elimination of variables from a system of polynomial equations.

As far as numeric solutions of a system of polynomial equations are concerned, Grőbner bases or Ritt-Wu's algorithm can be used. These algorithms compute a triangular form of the polynomial equations by eliminating symbolic variables and use the resulting formulation for computing the roots. However, the resulting algorithms are slow. Recently the technique of polynomial continuation has gained importance for computing the full list of geometrically isolated solutions to a system of polynomial equations. They have been used in many robotics and engineering applications (Morgan, 1987; Wampler and Morgan, 1991). Although continuation methods have a good theoretical background and in some cases a high degree of computational reliability, its usage is limited to applications requiring all the solutions in the complex domain and thereby making them slow for some practical applications where only real solutions are needed. In many cases the system of polynomial equations may have a high *Bezout bound* (the total number of solutions in the complex domain), but we are only interested in the solutions in a small subset of the real domain (the domain of interest). The continuation technique requires starting with a particular system of equations having the same Bezout number as the given system and marching along to compute all the solutions of the given system. It is difficult to restrict them to computing the solutions in the domain of interest. As a result they are slow for systems with high Bezout bound. Furthermore, they are restricted to dense polynomial systems.

Multipolynomial resultant algorithms provide the most efficient methods (as far as asymptotic complexity is concerned) for solving a system of polynomial equations or eliminating variables (Bajaj et. al., 1988). Their main advantage lies in the fact that the resultant can always be expressed in terms of matrices and determinants. In the most general case, the resultant is expressed as the ratio of two determinants and for small values of n (where n is the number of equations), as the determinant of a single matrix. As a result, we are able to use algorithms from linear algebra and obtain tight bounds on the running times of multipolynomial resultant algorithms. Furthermore, in many symbolic and numeric applications, we may choose not to expand the determinants and use properties of matrices and determinants to incorporate the use of resultants in the specific applications (Manocha and Canny, 1991; Manocha and Demmel, 1992).

In this paper we present an interpolation based algorithm to compute the resultant of a system of polynomial equations to obtain a "symbolically smaller" system. The algorithm is also useful for interpolating polynomials from their values and expanding symbolic determinants. We also describe an efficient implementation of the algorithm using modular methods and present its performance for different applications. In particular, the algorithm takes about 575 and 138 seconds on a Sun-4 and IBM RS/6000, respectively, to implicitize the bicubic patch mentioned in (Hoffmann, 1990b). In general, it is possible to obtain tight bounds on the running time of the algorithm and the storage requirements are a linear function of input and output polynomial sizes and a quadratic function of the order of the matrix corresponding to the resultant formulation. Later, we extend this algorithm to compute roots of a system of polynomial equations in the domain of interest. In this case, we reduce the problem to computing roots of univariate polynomials using coefficient specializations.

The rest of the paper is organized as follows. In Section 2 we give a brief preview of different formulations used for computing the resultant of a system of polynomials. In Section 3, we present an algorithm for eliminating one or more variables from the given system of equations and express the resultant as a polynomial in the coefficients of given equations (which may be symbolic variables) and finally in Section 4, we give details of the algorithm used for computing the numeric solutions of a given system of polynomials in the domain of interest.

2. Multipolynomial Resultants

Given n homogeneous polynomials in n unknowns, the resultant of the given system of equations is a polynomial in the coefficients of the given equations. The most familiar form of the resultant is the Sylvester's formulation for the case $n = 2$. In this case, the resultant can always be expressed as determinant of a matrix. However, a single determinant formulation may not exist for any arbitrary n and the most general formulation of resultant (to the best of our knowledge) expresses it as a ratio of two determinants (Macaulay, 1902). If all the polynomials have the same degrees, an improved formulation is given in (Macaulay, 1921). Many a time both determinants evaluate to zero. To compute the resultant we need to perturb the equations and use limiting arguments. This corresponds to computing the characteristic polynomials of both the determinants (Canny, 1988). The ratio of the two characteristic polynomials is termed the *Generalized Characteristic Polynomial*, and the resultant corresponds to its constant term (Canny, 1990). If the constant term is zero, its lowest degree term contains important information and can be used for computing the proper components in the presence of excess components. This formulation has advantages for both numeric and symbolic applications. Many special cases, corresponding to $n = 2, 3, 4, 5, 6$ when the resultant can be expressed as the determinant of a matrix, are given in (Dixon, 1908; Jouanolou, 1991; Morley, 1925; Morley and Coble, 1927). Historical accounts of resultants and elimination theory are presented in (Abhyankar, 1976; White, 1909).

Most of the formulation presented in the classical literature correspond to computing the resultants of dense polynomial systems. In the last few decades, a number of results have appeared in the literature pertaining to the resultants of sparse polynomial systems (Bernshtein, 1975; Gelfand, Kapranov and Zelevinsky, 1990; Sturmfel, 1991). This is very useful due to the fact that many polynomial systems encountered in geometric modeling, robotics and vision application are rather sparse (Manocha, 1992). Resultant formulations of sparse polynomial systems, expressed in terms of matrices and determinants, have appeared in (Canny and Emiris, 1993; Sturmfel and Zelevinsky, 1992).

3. Symbolic Elimination

In this section we present an algorithm for efficiently computing the resultant of a system of polynomial equations, whose coefficients may be symbolic variables. Computing the resultant involves constructing the corresponding matrices from the given system of equations and evaluating their determinants. The entries of the matrices are polynomial functions of the coefficients of the polynomial equations. As such it should be relatively easy to implement such an algorithm within the framework of a computer algebra system. However, these systems take a lot of time for evaluating even low order symbolic determinants. Consider the problem of implicitizing a bicubic parametric surface, whose implicit representation is a degree 18 polynomial in 3 variables, say x, y and z. In this case, the resultant of the parametric equations correspond to a 18×18 determinant and each of its entries is a linear polynomial in x, y, and z. However, standard computer algebra systems (available on most workstations) are not able to evaluate such determinants in a reasonable amount of time and space (Manocha and Canny, 1992a). Mostly they run for a long period of time and crash because of their memory limitations.

There are many reasons for the failure and bad performance of symbolic determinant expansion algorithms implemented within the framework of computer algebra systems.

1 Most computer algebra systems use sparse representation for multivariate polynomials and the computations become slow whenever the polynomials generated are dense.
2 The algorithms used are symbolic in nature and perform operations like polynomial addition, multiplication etc. on the input and the intermediate expressions being generated. The arithmetic for these symbolic operations is expensive. For example, the cost of multiplying two multivariate polynomials is quadratic for most implementations. Moreover the algorithms may generate large intermediate expressions. For example when using straight-forward Gaussian elimination over the polynomial entry domain, it can happen that intermediate subdeterminants are very large polynomials while the final answer is an expression of modest size.
3 The implementations of symbolic algorithms in lisp-like environments requires a large amount of virtual memory and thereby slows down the computations.
4 These systems use exact arithmetic and represent the coefficients of intermediate expressions as *bignums*. As a result, the cost of arithmetic operations is quadratic in the coefficient size. The coefficient size is proportional to the degree of the intermediate polynomial expressions being generated and tends to grow linearly with their degree.

The bottleneck in the resultant algorithms is the symbolic expansion of determinants. We therefore chose not to work within the environment of computer algebra systems and rather use an algorithm based on multivariate interpolation to compute the symbolic determinants. As a result, the resulting algorithm involves numeric computations and no intermediate symbolic expressions are generated. This takes care of the problem of generating large intermediate symbolic expression. However, the magnitude of the intermediate numbers grows and we need to use *bignum arithmetic* for arithmetic operations. As a result the cost of each arithmetic operation is quadratic in the size of the operands. Moreover, it imposes additional memory requirements for each intermediate number, which slows down the resulting computation. To reduce the memory requirements and cost of arithmetic operations, we perform our computations over finite fields and use a probabilistic algorithm based on chinese remainder theorem to recover the actual coefficients. Thus, bignum arithmetic is restricted only to the computations related to chinese remainder theorem. The complexity of the resulting algorithm is linear in the size of the coefficients of the resultant, except for the chinese remainder step, which is quadratic in the size of the coefficients. However, the running time of the algorithm is dominated by multivariate interpolation and the chinese remainder step is a small part of the overall computation. The algorithm has been implemented in C++ (as opposed to using lisp) and we consider sparse, dense and probabilistic interpolation techniques for multivariate polynomials.

3.1. Multivariate Interpolation

Let's assume that each entry of the matrix is a polynomial in $x_1, x_2, \ldots, x_n$ and the matrix is of order m. If the entries are rational functions, they can be multiplied by suitable polynomials such that each entry of the resulting matrix is a polynomial. Furthermore, the coefficients of the matrix entries are from a field of characteristic zero. The main idea is to determine the power products that occur in the determinant, say a multivariate polynomial $F(x_1, x_2, \ldots, x_n)$. Let the maximum degree of x_i in $F(x_1, x_2, \ldots, x_n)$ be d_i. The d_i's can be determined from the matrix entries. F can have at most $q_1 = (d_1+1)(d_2+1)\ldots(d_n+1)$ monomials. In some cases, it is easier to compute a bound on the total degree of F. Any degree d polynomial in n variables can have at most $q_2 = C_R(n, d)$ coefficients, where

$$C_R(n, d) = \begin{pmatrix} n+d-1 \\ d \end{pmatrix}.$$

We represent the determinant as

$$F(x_1, x_2, \ldots, x_n) = c_1 m_1 + c_2 m_2 + \ldots + c_q m_q, \tag{3.1}$$

where q is bounded by q_1 or q_2. The $m_i = x_1^{d_{1,i}} x_2^{d_{2,i}} \ldots x_n^{d_{n,i}}$ are the distinct monomials and the c_i are the corresponding non-zero coefficients. The problem of determinant expansion corresponds to computing the c_i's. By choosing different substitutions for $(x_1, \ldots, x_n)$ and computing the corresponding $F(x_1, \ldots, x_n)$ (expressed as determinant of numeric matrices obtained after substitutions of x_i's) the problem is reduced to that of *multivariate interpolation* (Ben-or and Tiwari, 1988; Zippel, 1990). Since there are q monomials, we need to choose q distinct substitutions and solve the resulting $q \times q$ system of linear equations. The running time of the resulting algorithm is $O(q^3)$ and takes $O(q^2)$ space. To reduce the running time and space of the algorithm we perform Vandermonde interpolation.

Let $p_1, p_2, \ldots, p_n$ denote distinct primes and $b_i = p_1^{d_{1,i}} p_2^{d_{2,i}} \ldots p_n^{d_{n,i}}$ denote the value of the monomial m_i at $(p_1, p_2, \ldots, p_n)$. Clearly, different monomials evaluate to different values. Let $a_i = F(p_1^{i-1}, p_2^{i-1}, \ldots, p_n^{i-1})$, $i = 1, \ldots, q$. a_i's are computed by Gauss elimination. Thus, the problem of computing c_i's is reduced to solving a Vandermonde system $VC = A$, where

$$V = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ b_1 & b_2 & \ldots & b_k \\ \vdots & \vdots & \vdots & \vdots \\ b_1^{q-1} & b_2^{q-1} & \ldots & b_q^{q-1} \end{pmatrix}, \quad C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_q \end{pmatrix}, \quad A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_q \end{pmatrix}. \tag{3.2}$$

Computing each a_i takes $O(m^3)$ time, where m is the order of the matrix. However, Macaulay's formulation results in sparse matrices and an efficient utilizing the structure of those matrices has been presented in (Canny, Kaltofen and Laksman, 1989). As far as solving Vandermonde systems is concerned, simple algorithms of time complexity $O(q^2)$ and $O(q)$ space requirements are known (Zippel, 1990). In (Kaltofen and Laksman, 1988) an improved algorithm of time complexity $O(M(q)log(q))$ is presented, where $M(q)$ is the time complexity of multiplying two univariate polynomials of degree q. Thus, the running time of the resulting algorithm is $O(qm^3 + M(q)log(q))$ and space requirements are $O(m^2 + q)$. However, due to simplicity we use the $O(q^2)$ algorithm for Vandermonde interpolation and the rest of the time complexities are presented in terms of that.

Before we use Vandermonde interpolation, the algorithm for computing symbolic determinant needs to know q and the m_i's corresponding to nonzero c_i's. In the worst case, q may correspond to q_1 or q_2 and the resulting problem is that of *dense interpolation*. The m_i's are enumerated in some order (e.g. lexicographic) and b_i's are computed by substituting p_j for x_j. In general, d is larger than n and q_1 or q_2 tend to grow exponentially with the number of variables and as a polynomial function of the degree of each matrix entry. Furthermore, this approach becomes unattractive when the determinant of the matrix is a sparse polynomial. An alternative is to use sparse interpolation algorithms. A sparse deterministic interpolation algorithm for such problems has been described in (Ben-or and Tiwari, 1988). Its time and space complexity have been improved in (Kaltofen and Laksman, 1988).

3.1.1. Sparse Interpolation

Ben-Or and Tiwari's needs an upper bound $T \geq q$ on the number of non-zero monomials in $F(x_1, \ldots, x_n)$ (Ben-or and Tiwari, 1988). Given $(p_1^j, p_2^j, \ldots, p_n^j)$, it computes the monomial values $b_i = p_1^{d_{1,i}} p_2^{d_{2,i}} \ldots p_n^{d_{n,i}}$ by computing the roots of an auxiliary polynomial $G(z)$. These b_i's are used for defining the coefficient matrix of the Vandermonde system.

The polynomial $G(z)$ is defined as

$$G(z) = \prod_{i=1}^{T} (z - b_i) = z^k + g_{k-1} z^{k-1} + \ldots + g_0.$$

Its coefficients, g_i's, are computed by solving a Toeplitz system of equations. The Toeplitz system is formed by computing the a_i's and using the property $G(b_i) = 0$. Given $G(z)$, the algorithm computes its integer roots to compute the corresponding b_i's. The roots are computed using p-adic root finder (Loo, 1983). The running time of the resulting algorithm for polynomial interpolation is $(ndT^3 log(n) + m^3 T)$. The dominating step is the polynomial root finder which takes $O(T^3 log(B))$, where B is an upper bound on the values of the roots.

This algorithm is unattractive for expanding symbolic determinants due to the fact that it is rather difficult to come up with a sharp bound on T. We only have the choice of using $T = q_1$ or $T = q_2$ and the resulting problem corresponds exactly to dense interpolation. As a result, b_i's can be computed directly and we do not need to use any algorithm for solving a Toeplitz system or computing integer roots of univariate polynomials.

Thus, neither the deterministic sparse nor the dense interpolation algorithms are well suited for our application.

3.1.2. Probabilistic Interpolation

Zippel presents a probabilistic algorithm for interpolation which does not expect any information about the number of non-zero terms of the polynomial being interpolated

(Zippel, 1990). It only expects a bound on the degree of each variable in any monomial. Such a bound is easy to compute in our case from the matrix entries. The interpolation algorithm proceeds inductively, using the degree information and develops an estimate of the number of terms as each new variable is introduced. As a result its performance is *output sensitive* and depends on the actual number of terms in the polynomial. We present a brief outline of the algorithm below. It has been explained in detail in (Zippel, 1990).

Choose n random numbers $r_1, \ldots, r_n$ from the coefficient field used for defining the polynomial coefficients. The algorithm proceeds inductively and introduces a variable at each stage. At the kth stage, we have a polynomial in k variables. It corresponds to

$$F_k(x_1, \ldots, x_k) = F(x_1, x_2, \ldots, x_k, r_{k+1}, \ldots, r_n) = c_{1,k} m_{1,k} + \ldots + c_{\alpha,k} m_{\alpha,k}.$$

Moreover, $\alpha \leq q$. In general, F_i is a polynomial in i variables $x_1, \ldots, x_i$.

To compute $F_{k+1}(x_1, \ldots, x_{k+1})$ from $F_k(x_1, \ldots, x_k)$, Zippel's algorithm represents F_{k+1} as

$$F_{k+1}(x_1, \ldots, x_{k+1}) = F(x_1, \ldots, x_{k+1}, r_{k+2}, \ldots, r_n) = h_1(x_{k+1}) m_{1,k} + \ldots + h_\alpha(x_{k+1}) m_{\alpha,k},$$

where each each $h_i(x_{k+1})$ is a polynomial of degree d_{k+1} and d_{k+1} is a bound on the degree of x_{k+1} in any term of $F(x_1, \ldots, x_n)$. It computes $h_i(x_{k+1})$ by Vandermonde interpolation. The value of each $h_i(x_{k+1})$ are obtained for $d_{k+1}+1$ values $1, p_{k+1}, p_{k+1}^2, \ldots, p_{k+1}^{d_{k+1}}$ as follows:

$$F_{k+1}(1, \ldots, 1, p_{k+1}^j, r_{k+2}, \ldots, r_n) = h_1(p_{k+1}^j) + \ldots + h_\alpha(p_{k+1}^j)$$

$$F_{k+1}(p_1, \ldots, p_k, p_{k+1}^j, r_{k+2}, \ldots, r_n) = h_1(p_{k+1}^j) b_{i,k} + \ldots + h_\alpha(p_{k+1}^j) b_{\alpha,k}$$

$$F_{k+1}(p_1^2, \ldots, p_k^2, p_{k+1}^j, r_{k+2}, \ldots, r_n) = h_1(p_{k+1}^j) b_{i,k}^2 + \ldots + h_\alpha(p_{k+1}^j) b_{\alpha,k}^2$$

$$\vdots$$

$$F_{k+1}(p_1^{\alpha-1}, \ldots, p_k^{\alpha-1}, p_{k+1}^j, r_{k+2}, \ldots, r_n) = h_1(p_{k+1}^j) b_{i,k}^{\alpha-1} + \ldots + h_{\alpha-1}(p_{k+1}^j) b_{\alpha,k}^{\alpha-1}$$

This is a Vandermonde system of α equations in α unknowns and can be solved for $h_i(p_{k+1}^j)$. The computation is repeated for $j = 0, \ldots, d_{k+1}$. Given $h_i(1), h_i(p_{k+1}), h_i(p_{k+1}^2), \ldots, h_i(p_{k+1}^{d_{k+1}})$, use Vandermonde interpolation (for univariate polynomials) to compute $h_i(x_{k+1})$. These $h_i(x_{k+1})$ are substituted to represent F_{k+1} as a polynomial in $k+1$ variables. $F_{k+1}(x_1, \ldots, x_{k+1})$ can have at most $(\alpha * (d_{k+1}+1)) \leq q$ terms.

The algorithm starts with $F(r_1, \ldots, r_n)$ and computes the n stages as shown above. There is a small chance that the answer produced by the algorithm is incorrect. This happens for a choice of $r_1 \ldots, r_n$ and can be explained in the following manner. Let the stage I of the algorithm result in a polynomial of the form

$$F_1(x_1) = F(x_1, r_2, \ldots, r_n) = c_0 + c_1 x_1^3 + \ldots + c_\beta x_1^{d_1}.$$

$F_2(x_1, x_2)$ actually is a polynomial of the form

$$F_2(x_1, x_2) = F(x_1, x_2, r_3, \ldots, r_n) = h_1(x_2) + h_2(x_2) x_1 + h_3(x_2) x_1^2 + \ldots + h_{d_1+1}(x_2) x_1^{d_1}.$$

This implies

$$h_1(r_2) = c_0, \quad h_2(r_2) = 0, \quad h_3(r_2) = 0, \ldots, h_{d_1+1}(r_2) = c_\beta.$$

In practice, $h_2(x_2)$ and $h_3(x_2)$ maybe nonzero polynomials, but for the choice of r_2, where $h_2(r_2) = 0$, $h_3(r_2) = 0$, the algorithm assumes them as zero polynomials and proceeds. As a result the algorithm may report fewer terms in $F(x_1, \ldots, x_n)$.

Such an error is possible at each stage of the algorithm. Let us assume that r_i's are chosen uniformly randomly from a set of size S, than the probability that this algorithm gives a wrong answer is less than

$$\frac{nd^2q^2}{S}, \tag{3.3}$$

where $d = max(d_1, \ldots, d_n)$ (Zippel, 1990).

Later on we use this probability bound for the choice of finite fields used for modular computation. The running time of the algorithm is $O(ndq^2 + m^3ndq)$. This has been improved using better algorithms for Vandermonde interpolation (Kaltofen and Laksman, 1988).

A deterministic version of this algorithm of complexity $O(ndq^2T + m^3ndqT)$ is given in (Zippel, 1990) as well. T is an upper bound on the number of non-zero terms in the polynomial. In the worst case, T corresponds to q_1 or q_2. However, due to simplicity and time complexity of the algorithm, we use the probabilistic version for our implementation. Furthermore the random numbers are chosen from finite fields of appropriate sizes so as to impose a tight bound on the probability of failure. We also introduce simple and randomized checks in the algorithm to detect wrong answers. These checks correspond to substituting random values for $x_1 = q_1, \ldots, x_i = q_i$ at the ith stage of the algorithm and comparing the numeric determinant, $F(q_1, q_2, \ldots, q_i, r_{i+1}, r_{i+2}, \ldots, r_n)$, with the interpolated polynomial at the ith stage, $F_i(q_1, \ldots, q_i, r_1, \ldots, r_n)$.

3.1.3. Modular Arithmetic

An important problem in the context of finite field computations is getting a tight bound on the coefficients of the resulting polynomial. Since the resultant can always be expressed as a ratio of determinants (or a single determinant), it is possible to use Hadamard's bound for determinants to compute a bound on the coefficients of the resultant (Knuth, 1981). In practice, we found such bounds rather loose and use a randomized version of chinese remainder algorithm to compute the actual bignums. The main idea is to compute the coefficients modulo different primes, use the chinese remainder theorem for computing the bignum and make a check whether the bignum is the actual coefficient of the determinant. This process stops when the products of all the primes (used for finite field representation) is greater than twice the magnitude of the largest coefficient in the output. It is possible that for a certain choice of primes, the algorithm results in a wrong answer. We show that the probability of failure is bounded by l/p, where l corresponds to the number of primes used in the chinese remainder algorithm, p is the magnitude of the smallest prime number used in this computation.

The algorithm proceeds in stages. At the kth stage a random prime number (p_k) is chosen and $G_k(x_1, \ldots, x_n) = F(x_1, \ldots, x_n)\ mod\ p_k$ is computed using the interpolation algorithm. Let

$$G_k(x_1, \ldots, x_n) = c_{1,k}m_1 + \ldots + c_{q,k}m_q.$$

Thus, the coefficients of various G_i's satisfy the relation

$$c_{i,1} = c_i \ mod \ p_1$$

$$\vdots$$

$$c_{i,k} = c_i \ mod \ p_k,$$

These $c_{i,j}$'s are used for computing the bignum, $r_{i,k}$ using chinese remainder theorem, and satisfying the relations (Knuth, 1981)

$$r_{i,k} \ mod \ p_j = c_{i,j}, \quad j = 1, k$$

While using chinese remainder theorem it is assumed that $r_{i,k}$ are integers lying in the range $(-\frac{p_1 p_2 \ldots p_k}{2}, \frac{p_1 p_2 \ldots p_k}{2})$. At this stage we compare the bignums corresponding to the $(k-1)$ and k stage. If

$$r_{i,k-1} = r_{i,k}, \qquad i = 1, q$$

it is reasonable to assume that $r_{i,m} = c_i$, $i = 1, q$ and the algorithm terminates. The base case is $k = 2$. Otherwise we repeat the computation for the $(k+1)$ stage.

In general, $l+1$ prime numbers are being used, where l is the minimum integer satisfying the relation

$$2|c_j| < p_1 p_2 \ldots p_{l+1}$$

and $|c_j|$ corresponds to the coefficient of maximum magnitude of the resultant and p_j's are randomly chosen primes. Thus, the algorithm is:

1 Compute the $c_{i,j}$'s using the probabilistic interpolation algorithm.
2 Given $c_{i,1}, c_{i,2}, \ldots, c_{i,j}$, use chinese remainder theorem to compute $r_{i,j}$ for $i = 1, q$.
3 If $r_{i,j-1} = r_{i,j}$ for $i = 1, q$, then $c_i = r_{ij}$, else repeat the steps given above.

3.1.4. Probabilistic Analysis

In this section we analyze the probability of error of the chinese remainder algorithm presented above. Let's consider a particular coefficient c_i of the determinant. The analysis can be extended to all the coefficients of the polynomial. Let us consider a sequence of prime numbers $p_1, p_2, \ldots, p_l$ such that

$$p_1 p_2 \ldots p_{l-1} < 2|c_i| < p_1 p_2 \ldots p_l.$$

We assume all the primes have the same order of magnitude. Furthermore, let

$$c_{i,j} = c_i \ mod \ p_j, \quad j = 1, l$$

and $r_{i,j}$ correspond to the bignums defined in the previous section. The main assumption is our analysis is:

Assumption $c_{i,j}$ are integers distributed uniformly in the interval $(-\frac{p_j}{2}, \frac{p_j}{2})$ and $r_{i,j}$ are integers distributed uniformly in the interval $(-\frac{p_1 \ldots p_j}{2}, \frac{p_1 \ldots p_j}{2})$.

Under normal circumstances the algorithm would compute $c_{i,j}$ for $j = 1, l$ and terminate. However, it is possible that the algorithm stops after k $(k < l)$ stages. This is due to the fact

$$r_{i,k-1} = r_{i,k}$$

and leading us to the incorrect conclusion $c_i = r_{i,k}$. This can happen if and only if $r_{i,k} \in (-\frac{p_1 \ldots p_{k-1}}{2}, \frac{p_1 \ldots p_{k-1}}{2})$. Since, $r_{i,k}$ are uniformly distributed over $(-\frac{p_1 \ldots p_k}{2}, \frac{p_1 \ldots p_k}{2})$, the probability of that happening is $1/p_k$. We need to add this over all stages of computing c_i and the overall probability of failure is bounded by

$$1 - (1 - 1/p_2)(1 - 1/p_3) \ldots (1 - 1/p_{l-1}) \approx 1/p_2 + 1/p_3 + \ldots + 1/p_{l-1}.$$

In the worst case, all coefficients of $F(x_1, \ldots, x_n)$ are equal and therefore, the probability of failure of overall algorithm is bounded by l/p, where l is number of primes used to bound the coefficient and p is the magnitude of the primes used (since all primes have nearly the same order). Since, l is generally less than 10 and p is of the order of 2^{30} (explained in next section), the algorithm has very high probability of success. This bound can be improved by choosing an additional prime p_{l+1}, computing $c_{i,l+1}$, $r_{i,l+1}$ and verifying the fact $r_{i,l} = r_{i,l+1}$, $i = 1, q$. In the latter case the probability of failure is bounded by l/p^2.

3.2. Implementation

We have implemented the above algorithm in C++ on Sun-4's. The code was ported over to an IBM RS/6000 for performance analysis. Given a system of polynomial equations, it expresses the resultants in terms of determinants. Given matrices with polynomial entries, it computes the degree bound for each variable by adding the degrees of that variable in various entries of the matrix.

We have implemented the dense as well as probabilistic versions of the interpolation algorithm. It can be easily interfaced with computer algebra systems. The total amount of space required for resultant computation is $O(m^2 + |c|q)$, where m is the order of the matrix, q is the number of terms in the determinant and $|c|$ correspond to the size of the largest coefficient in the determinant. The algorithm is not constrained by any form of memory requirements, performs efficiently on the workstations and can easily run on personal computers. Moreover given q, it is possible to come up with a tight bound on the running time of the resulting algorithm.

3.2.1. Choice of Finite Fields

The interpolation algorithm based on the randomized version of chinese remainder theorem is output sensitive. In other words the time complexity of the algorithm is directly proportional to k, the number of primes used in the finite field computation. To minimize k, we use primes of maximum magnitude possible.

Most of the workstations are 32 bit machines. In other words, all machine instructions for integer arithmetic operate on operands, whose magnitude is bounded by 2^{32}. However, if we use operands of order 2^{32}, simple operations like addition, subtraction can result in overflow and to implement them correctly we need to resort to bignum arithmetic.

In our case, we chose primes of the order 2^{30} (in fact always less than this number) to represent the finite fields. Thus, each operand of the finite field is bounded by 2^{30}. As a result the addition and subtraction operations, $(a+b) \ mod \ p$, $(a-b) \ mod \ p$, work correctly in the hardware implementation and there is no overflow error. However, multiplication can still result in overflow. A simple solution is to use finite fields of order 2^{16}, which slows down the algorithm by almost a factor of two. Depending upon the architecture of the machine on which the algorithm is being implemented, we suggest following techniques for implementing the multiplication subroutine:

> Many workstations provide a hardware implementation of the instruction $(a * b)\ mod\ p$. There may be no equivalent function defined in the higher level languages. As a result, we recommend an assembly level implementation of this subroutine, which takes as arguments a, b, p and return $(a * b)\ mod\ p$.
>
> The instruction presented above is rather sophisticated and as a result is not available on most RISC machines. However, most architectures have a multiplication instruction, which takes the operands in two different registers and return the result as a 64 bit number in two different registers, as well. Many machines, like the Sun-3, have a division instruction which assumes that the dividend is in two registers. As a result an instruction like $(a * b)\ mod\ p$ can be implemented as a combination of multiplication and division instructions. Otherwise, let the result of multiplication be contained in registers $r1$ and $r2$. Thus,
>
> $$a * b = r1 * 2^{32} + r2$$
>
> The fact $a < 2^{30}, b < 2^{30}$ implies $r1 < 2^{28}$. Thus,
>
> $$(a * b)\ mod\ p = ((((r1 * 2^{32})\ mod\ p) + (r2\ mod\ p))\ mod\ p$$
> $$= (((r1 * c)\ mod\ p) + (r2\ mod\ p))\ mod\ p,$$
>
> where $c = 2^{32}\ mod\ p$ is a precomputed constant. In fact the primes, are chosen such such that c is rather small and the multiplication $(r1 * c)$ does not result in a overflow. Otherwise the multiplication routine is called recursively. Thus, we find that the implementation of the multiplication subroutine uses two multiplication instructions, three remainder instructions (to compute the remainder of integer division) and one addition instruction for 32 bits integers. The actual impact of this multiplication routine on the speed of the overall algorithm is a function of machine's architecture. However, it seems faster (at least on Sun-3 and Sun-4's) than using finite fields of order 2^{16} and for each field we compute the coefficients of the polynomials and bignums using chinese remainder theorem.

It is not possible to choose any arbitrary prime for finite field computation. Let $V_2 = (v_{2j})$ represent the elements of the second row of the Vandermonde matrix, as shown in (3.2), used in the interpolation algorithms. p_k can be used as a prime for finite field computation, if and only if all the elements of the vector $V_{2k} = (v_{ij})$ mod p_k are distinct.

3.2.2. Choice of Random Numbers

The robustness of the Zippel's probabilistic interpolation algorithm is a function of the random numbers $r_1, \ldots, r_n$ chosen at the first stage of the algorithm. The probability of obtaining an incorrect answer is bounded by nd^2q^2/S (as shown in (3.3)). We work over finite fields of order $S \approx 2^{30}$. In many applications we expect q to be of the order of 10^4. Moreover, n is at least 3 or 4 and d can be anywhere in the range $(10, 40)$. As a result, the upper bound on the probability of incorrectness is close to one.

The probabilistic chinese remainder algorithm computes the answer over various finite fields (of order $\approx 2^{30}$). The actual number of finite fields used is a function of the size of the coefficients of the determinant. However, at least two prime fields are used. As a result we choose random numbers $R_i \in (0, 2^{60})$ and at each iteration corresponding to the chinese remainder theorem, we compute $r_i = R_i \; mod \; p_i$. As a result $S \approx 2^{60}$ and the probability of failure is bounded by 10^{-9} for these applications. Moreover, it is possible to decrease the upper bound on the probability of failure by choosing R_i's from a domain of appropriate size.

3.2.3. Implicitization

The resultant algorithm has been used for implicitizing rational parametric surfaces. Given a parametrization, expressed in projective coordinates,

$$(x, y, z, w) = (X(s,t), Y(s,t), Z(s,t), W(s,t)),$$

we formulate the parametric equations

$$wX(s,t) - xW(s,t) = 0$$

$$wY(s,t) - yW(s,t) = 0$$

$$wZ(s,t) - zW(s,t) = 0$$

and the problem of implicitization corresponds to computing the resultant of the above equations, by considering them as polynomials in s and t (Manocha and Canny, 1992a). Some experiments with the implementations of Grőbner bases and resultants in Macsyma 414.62 on a Symbolics lisp machine (with 16MB main memory and 120MB virtual memory) are described in (Hoffmann, 1990b). For many cases of bicubic surfaces (whose highest monomial is of the form s^3t^3), these systems are unable to implicitize such surfaces and fail due to insufficient virtual memory. Only a new algorithm for basis conversion is able to implicitize such surfaces, however it takes about 10^5 seconds, which would be considered impractical for most applications (Hoffmann, 1990b).

Lets consider the bicubic parametrization given in (Hoffmann, 1990b). It is

$$\begin{aligned} x &= -3t(t-1)^2 + (s-1)^3 + 3s \\ y &= 3s(s-1)^2 + t^3 + 3t \\ z &= -3(s(s^2-5s+5)t^3 + (s^3+6s^2-9s+1)t^2 - (2s^3+3s^2-6s+1)t + s(s-1)). \end{aligned}$$

We use the Dixon formulation of computing the resultant of bicubic formulation (Dixon, 1908) and the resultant corresponds to the determinant of 18×18 matrix, and each entry of the matrix is a linear function of x, y, z, w. The resulting matrix is:

$$\mathbf{M} = \mathbf{M_X} x + \mathbf{M_Y} y + \mathbf{M_Z} z + \mathbf{M_W},$$

where

$$
\mathbf{M_x} = \begin{bmatrix}
0 & 15 & -28 & 21 & -9 & 5 & -3 & -3 & 19 & -18 & 6 & -5 & 3 & -9 & 3 & 1 & 1 & 1 \\
-3 & -1 & 6 & -9 & 5 & 0 & 6 & 1 & -3 & 6 & -5 & 0 & -3 & 0 & -2 & 1 & 1 & 0 \\
-1 & 6 & -9 & 5 & 0 & 0 & 1 & -3 & 6 & -5 & 0 & 0 & 0 & -2 & 1 & 1 & 0 & 0 \\
-21 & 24 & 4 & -18 & 6 & -5 & 39 & -63 & 33 & 1 & 1 & 1 & -18 & 27 & -15 & 0 & 0 & 0 \\
33 & -14 & -3 & 6 & -5 & 0 & -57 & 30 & -2 & 1 & 1 & 0 & 27 & -15 & 0 & 0 & 0 & 0 \\
-14 & -3 & 6 & -5 & 0 & 0 & 30 & -2 & 1 & 1 & 0 & 0 & -15 & 0 & 0 & 0 & 0 & 0 \\
48 & -81 & 48 & 1 & 1 & 1 & -36 & 63 & -45 & 0 & 0 & 0 & 9 & -18 & 15 & 0 & 0 & 0 \\
-75 & 45 & -2 & 1 & 1 & 0 & 63 & -45 & 0 & 0 & 0 & 0 & -18 & 15 & 0 & 0 & 0 & 0 \\
45 & -2 & 1 & 1 & 0 & 0 & -45 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 \\
-30 & 60 & -48 & 0 & 0 & 0 & -3 & -12 & 21 & 0 & 0 & 0 & 6 & -3 & -3 & 0 & 0 & 0 \\
60 & -48 & 0 & 0 & 0 & 0 & -12 & 21 & 0 & 0 & 0 & 0 & -3 & -3 & 0 & 0 & 0 & 0 \\
-48 & 0 & 0 & 0 & 0 & 0 & 21 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 \\
-3 & -12 & 21 & 0 & 0 & 0 & 6 & -3 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-12 & 21 & 0 & 0 & 0 & 0 & -3 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
21 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & -3 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-3 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
\mathbf{M_y} = \begin{bmatrix}
-3 & -18 & 66 & -87 & 57 & -15 & 0 & 12 & -39 & 60 & -48 & 15 & -1 & 7 & -15 & 9 & 3 & -3 \\
0 & 39 & -72 & 57 & -15 & 0 & 3 & -21 & 45 & -48 & 15 & 0 & 1 & -12 & 12 & 3 & -3 & 0 \\
3 & -18 & 27 & -15 & 0 & 0 & -3 & 9 & -18 & 15 & 0 & 0 & 0 & 6 & -3 & -3 & 0 & 0 \\
36 & -42 & -9 & 60 & -48 & 15 & -19 & 34 & -30 & 9 & 3 & -3 & 6 & -9 & 5 & 0 & 0 & 0 \\
-51 & 9 & 45 & -48 & 15 & 0 & 28 & -27 & 12 & 3 & -3 & 0 & -9 & 5 & 0 & 0 & 0 & 0 \\
27 & 9 & -18 & 15 & 0 & 0 & -15 & 6 & -3 & -3 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\
-37 & 70 & -60 & 9 & 3 & -3 & 15 & -27 & 20 & 0 & 0 & 0 & -3 & 6 & -5 & 0 & 0 & 0 \\
64 & -57 & 12 & 3 & -3 & 0 & -27 & 20 & 0 & 0 & 0 & 0 & 6 & -5 & 0 & 0 & 0 & 0 \\
-45 & 6 & -3 & -3 & 0 & 0 & 20 & 0 & 0 & 0 & 0 & 0 & -5 & 0 & 0 & 0 & 0 & 0 \\
3 & -21 & 26 & 0 & 0 & 0 & 3 & 3 & -8 & 0 & 0 & 0 & -2 & 1 & 1 & 0 & 0 & 0 \\
-21 & 26 & 0 & 0 & 0 & 0 & 3 & -8 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
26 & 0 & 0 & 0 & 0 & 0 & -8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 3 & -8 & 0 & 0 & 0 & -2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & -8 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
\mathbf{M_z} = \begin{bmatrix}
9 & 18 & -3 & 0 & 0 & 0 & 9 & -36 & 15 & 0 & 0 & 0 & -6 & 18 & -8 & 0 & 0 & 0 \\
18 & -3 & 0 & 0 & 0 & 0 & -36 & 15 & 0 & 0 & 0 & 0 & 18 & -8 & 0 & 0 & 0 & 0 \\
-3 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 & -8 & 0 & 0 & 0 & 0 & 0 \\
9 & -36 & 15 & 0 & 0 & 0 & -6 & 18 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-36 & 15 & 0 & 0 & 0 & 0 & 18 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
15 & 0 & 0 & 0 & 0 & 0 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-6 & 18 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
18 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Parametrization	Implicit Degree	Terms	Sun-4	IBM RS/6000
$s^2 + t^2$	4	10	1 sec.	1 sec.
$s^3 + t^3$	9	220	6 sec.	3 sec.
$s^2 t^2$	8	165	4 sec.	2 sec.
$s^3 t^3$	18	1330	114 sec.	26 sec.
$s^3 t^4$	24	2925	430 sec.	118 sec.

Table 1. The performance of resultant algorithm for implicitization (a single iteration over a finite field)

$$\mathbf{M_W}=\begin{bmatrix}
0 & 15 & -28 & 21 & -9 & 5 & -3 & -3 & 19 & -18 & 6 & -5 & 3 & -9 & 3 & 1 & 1 & 1\\
-3 & -10 & -105 & 189 & -139 & 30 & 6 & 10 & 42 & -120 & 121 & -30 & -3 & -12 & 42 & -29 & -11 & 6\\
-1 & 3 & 18 & -13 & -24 & 30 & 1 & -12 & 9 & 13 & 6 & -30 & 0 & 6 & -19 & -5 & 12 & 6\\
-21 & -30 & -23 & 117 & -111 & 10 & 48 & -45 & 21 & -71 & 109 & -14 & -27 & 42 & 20 & -33 & -15 & 3\\
33 & 76 & 159 & -237 & 136 & -30 & -66 & -75 & -30 & 79 & -26 & 6 & 24 & 83 & -42 & -15 & 3 & 0\\
-14 & -57 & -81 & 28 & 6 & -30 & 18 & 42 & 44 & -20 & 12 & 6 & -7 & -24 & -15 & 3 & 0 & 0\\
48 & -108 & 345 & -530 & 424 & -89 & -207 & 348 & -307 & 273 & -285 & 78 & 96 & -120 & 1 & 57 & 21 & -15\\
-75 & 60 & -390 & 394 & -101 & 6 & 303 & -136 & 219 & -285 & 78 & 0 & -138 & -8 & 60 & 21 & -15 & 0\\
45 & 6 & 251 & -95 & 12 & 6 & -172 & 21 & -195 & 78 & 0 & 0 & 82 & 24 & 27 & -15 & 0 & 0\\
-30 & 141 & -345 & 513 & -447 & 118 & 168 & -309 & 240 & -105 & 159 & -55 & -57 & 102 & -51 & -28 & -10 & 8\\
60 & -165 & 426 & -447 & 118 & 0 & -309 & 258 & -111 & 159 & -55 & 0 & 114 & -87 & -12 & -10 & 8 & 0\\
-48 & 66 & -288 & 118 & 0 & 0 & 231 & -39 & 126 & -55 & 0 & 0 & -93 & 6 & -18 & 8 & 0 & 0\\
-3 & -12 & 30 & -105 & 159 & -55 & 15 & 48 & -63 & -28 & -10 & 8 & -21 & -3 & 30 & 0 & 0 & 0\\
-12 & 48 & -111 & 159 & -55 & 0 & 60 & -99 & -12 & -10 & 8 & 0 & -3 & 30 & 0 & 0 & 0 & 0\\
21 & -39 & 126 & -55 & 0 & 0 & -105 & 6 & -18 & 8 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0\\
6 & -15 & 39 & -28 & -10 & 8 & -30 & 15 & 15 & 0 & 0 & 0 & 6 & -3 & -3 & 0 & 0 & 0\\
-3 & 3 & -12 & -10 & 8 & 0 & 15 & 15 & 0 & 0 & 0 & 0 & -3 & -3 & 0 & 0 & 0 & 0\\
-3 & 6 & -18 & 8 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Straightforward use of the determinant computation function available on Mathematica and Maple fails to expand such a determinant. The computer algebra systems run for a long period of time and fail due to memory requirements on a Sun-4.

Our interpolation algorithm takes about 575 and 138 second on a Sun-4 and IBM RS/6000, respectively. The implicit computations are performed over various finite fields and the algorithm works over 5 different fields in all. This is due to the fact the coefficient of maximum magnitude lies in the interval $(-2^{60}, -2^{45})$ or $(2^{45}, 2^{60})$. Since the implicit formulations are dense polynomials, in general, we use a dense interpolation algorithm. The total number of monomials is given by

$$q = \begin{pmatrix} d+3 \\ 3 \end{pmatrix},$$

where d is the degree of the implicit equation. As a result it is possible to derive a tight bound on the running time of the algorithms. In Table I, the running time of various parametrizations, over a single iteration of a finite field are presented. The actual number of iterations is a function of the size of the maximum coefficient of the determinant.

3.2.4. Symbolic Determinants

We used the algorithm for expanding symbolic determinants, where each entry of the matrix is a polynomial. In particular, we present a symbolic determinant arising in a high energy particle physics calculation. Each entry of the matrix M (shown below) is a polynomial in e, k and r (after substituting the various variables). However, the entries can be complex polynomials. We treat $j = \sqrt{-1}$ as a symbolic variable and substitute the value in the final determinant.

The entries of the matrix are polynomials in the intermediate variables defined below. However, we find it useful to express the determinant as a polynomial in as few variables as possible. This gives us a better bound on the number of terms used for dense interpolation algorithm, and reduces the number of stages (and hence the overall running cost) for the probabilistic interpolation algorithm. We therefore, treat the determinant as a polynomial $F(\delta, k, \eta, e)$, where $\delta = kr$ and $\eta = jk$. The maximum degrees of δ, k, η and e in $F(\delta, k, \eta, e)$ are $32, 16, 16, 8$, respectively. For dense interpolation, the bound on the number of terms is $q_1 = 85833$.

The matrix $M =$

$$
\left[\begin{array}{cccccccccccccccccccccccccccccccc}
e_1 & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & e_1 & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & e_2 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & e_2 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 \\
\delta & 0 & \eta & 0 & \gamma & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & \eta & 0 \\
0 & \delta & 0 & -\eta & 0 & \gamma & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & -\eta \\
-\eta & 0 & \delta & 0 & -\eta & 0 & \gamma & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -\eta & 0 & \delta & 0 \\
0 & \eta & 0 & \delta & 0 & \eta & 0 & \gamma & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & 0 & 0 & 0 & 0 & \eta & 0 & \delta \\
\delta & 0 & 0 & k & \delta & 0 & 0 & k & \gamma & 0 & 0 & k & \delta & 0 & 0 & k & \delta & 0 & 0 & k & 0 & 0 & 0 & 0 & \delta & 0 & 0 & k & \delta & 0 & 0 & k \\
0 & \delta & -k & 0 & 0 & \delta & -k & 0 & 0 & \gamma & -k & 0 & 0 & \delta & -k & 0 & 0 & \delta & -k & 0 & 0 & 0 & 0 & 0 & 0 & \delta & -k & 0 & 0 & \delta & -k & 0 \\
0 & -k & \delta & 0 & 0 & -k & \delta & 0 & 0 & -k & \gamma & 0 & 0 & -k & \delta & 0 & 0 & -k & \delta & 0 & 0 & 0 & 0 & 0 & 0 & -k & \delta & 0 & 0 & -k & \delta & 0 \\
k & 0 & 0 & \delta & k & 0 & 0 & \delta & k & 0 & 0 & \gamma & k & 0 & 0 & \delta & k & 0 & 0 & \delta & 0 & 0 & 0 & 0 & k & 0 & 0 & \delta & k & 0 & 0 & \delta \\
\delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \gamma & 0 & 0 & \eta & 0 & 0 & 0 & 0 & \delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta \\
0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \gamma & \eta & 0 & 0 & 0 & 0 & 0 & 0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 \\
0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \gamma & 0 & 0 & 0 & 0 & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 \\
-\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \gamma & 0 & 0 & 0 & 0 & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta \\
\delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & 0 & 0 & 0 & 0 & 1+\delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta & \delta & 0 & 0 & -\eta \\
0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & 0 & 0 & 0 & 0 & \gamma & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 & 0 & \delta & -\eta & 0 \\
0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 & 0 & 0 & 0 & 0 & 0 & \eta & \gamma & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 & 0 & \eta & \delta & 0 \\
\eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta & 0 & 0 & 0 & 0 & \eta & 0 & 0 & \gamma & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta & \eta & 0 & 0 & \delta \\
\delta & 0 & 0 & -k & \delta & 0 & 0 & -k & 0 & 0 & 0 & 0 & \delta & 0 & 0 & -k & \delta & 0 & 0 & -k & \gamma & 0 & 0 & -k & \delta & 0 & 0 & -k & \delta & 0 & 0 & -k \\
0 & \delta & k & 0 & 0 & \delta & k & 0 & 0 & 0 & 0 & 0 & 0 & \delta & k & 0 & 0 & \delta & k & 0 & 0 & \gamma & k & 0 & 0 & \delta & k & 0 & 0 & \delta & k & 0 \\
0 & k & \delta & 0 & 0 & k & \delta & 0 & 0 & 0 & 0 & 0 & 0 & k & \delta & 0 & 0 & k & \delta & 0 & 0 & k & \gamma & 0 & 0 & k & \delta & 0 & 0 & k & \delta & 0 \\
-k & 0 & 0 & \delta & -k & 0 & 0 & \delta & 0 & 0 & 0 & 0 & -k & 0 & 0 & \delta & -k & 0 & 0 & \delta & -k & 0 & 0 & \gamma & -k & 0 & 0 & \delta & -k & 0 & 0 & \delta \\
\delta & 0 & -\eta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \gamma & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 \\
0 & \delta & 0 & \eta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \gamma & 0 & \eta & 0 & \delta & 0 & \eta \\
\eta & 0 & \delta & 0 & 0 & 0 & 0 & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \delta & 0 & \eta & 0 & \gamma & 0 & \eta & 0 & \delta & 0 \\
0 & -\eta & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \delta & 0 & -\eta & 0 & \gamma & 0 & -\eta & 0 & \delta \\
0 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & 1+e_3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & e_3 & 0 & 0 & 0 & 1+e_3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & 1+e_4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & e_4 & 0 & 0 & 0 & 1+e_4
\end{array}\right]
$$

where

$$\alpha{=}-k(1{+}r),\ \beta{=}-k(-1{+}r),\ \gamma{=}1{-}kr,\ e_1{=}e{+}\alpha,\ e_2{=}e{+}\beta,\ e_3{=}e\beta,\ e_4{=}e\alpha,\ \delta{=}-kr,\ \eta{=}jk,\ j{=}\sqrt{-1}$$

We used dense as well as probabilistic interpolation algorithm for this problem. The dense interpolation algorithm takes about 280 and 94 minutes on a Sun-4 and an IBM RS/6000, respectively. As far as the probabilistic interpolation is concerned the running time of the algorithm is a function of the ordering of the variables. We recommend ordering the variables in the order of non-increasing degrees. Since the determinant can have up to 85833 terms, the primes are chosen randomly from a field of order 2^{60}, so that the probability of obtaining a wrong answer is bounded by 2.5×10^{-9}. For this determinant the various stages of the algorithm proceed as:

$F(\delta, r_2, r_3, r_4)$ has 33 terms and the upper bound is 33 as well.
$F(\delta, k, r_3, r_4)$ has 225 terms, whereas the upper bound is $33 * 17 = 561$ terms.
$F(\delta, k, \eta, r_4)$ has 1376 terms, whereas the upper bound is $225 * 17 = 3825$ terms.
Finally, $F(\delta, k, \eta, e)$ has 8746 terms, whereas the upper bound is $1376 * 9 = 12384$ terms.

The algorithm takes 64 and 21 minutes on a Sun-4 and an IBM RS/6000 respectively. The upper bound at each stage gives a bound on the running time of the algorithm for that stage. After substituting for δ, η, j and simplifying the resulting expression, the determinant is

$$(-1-k^2+k^2r^2)^{12}(-7ek^2+7ek^2r^2+k-kr-e^2k-e^2kr-6ekr+e)^2$$
$$(-7ek^2+7ek^2r^2-k-kr+e^2k-e^2kr-6ekr+e)^2.$$

The performance can be improved by using complex arithmetic. The main idea is to treat the coefficients of the determinant as complex entires. As a result the number of variables doubles. Moreover, we work over finite fields of the form $a+bj$, where a, b are integers in the set $[0, p)$ and $j^2 = -1$. The operations of addition, subtraction and multiplication are defined as a combination of complex and finite field operations. To compute the inverse of $a + bj$, we represent is as $c + dj$ and

$$ac - bd = 1 \quad mod\ p,$$

$$ad + bc = 0 \quad mod\ p.$$

As a result,

$$(c, d) = \frac{1}{q}(a, -b),$$

where $q = a^2 + b^2$. All these operations are performed modulo p.

4. Numeric Solutions

In this section we present an algorithm to compute the numerical solutions of a given system of polynomial equations. However we are only interested in the solutions lying in a subset of the real domain.

Given n homogeneous equations in $n+1$ unknowns, $F_1(x_0, x_1, \ldots, x_n)$, $\ldots$, $F_n(x_0, x_1, \ldots, x_n)$, where the domain of variables is limited to $1 \times [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$. It is assumed that the algebraic set defined by these equations has no excess components, otherwise we perturb the given equations and use limiting arguments for computing the roots. x_0 is the homogenizing variable and we are only interested in the affine solutions. Let

$$F_0(x_0, x_1, \ldots, x_n) = u_0x_0 + u_1x_1 + \ldots + u_nx_n,$$

be a linear polynomial and $R(u_0, u_1, \ldots, u_n)$ be the resultant of $F_0, F_1, \ldots, F_n$ obtained by considering them as polynomials in x_0, x_1, $\ldots$, x_n. It is a homogeneous polynomial in u_i's and its degree is equal to the product of the degrees of F_i's. $R(u_0, \ldots, u_n)$ is the u-resultant of the given system of equations and factors into linear factors of the form

$$\alpha_0u_0 + \alpha_1u_1 + \ldots + \alpha_nu_n,$$

where $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ correspond to the solution of the original system (Waerden, 1950). However, computing the expression $R(u_0, \ldots, u_n)$ and factoring into linear factors can be a time consuming task, even for low degree polynomials. We therefore, specialize some of the u_i's and reduce the problem to computing roots of univariate polynomials in a real interval. For many small values of n and certain combinations of the degrees of F_i's, the resultant can be expressed as determinant of a matrix. Otherwise the resultant can be expressed as a ratio of two determinants and in either case the entries of the matrices are polynomials in u_0, u_1, ..., u_n. After specialization, these determinants are univariate polynomials in u_0 and the problem of computing roots of the multivariate systems is reduced to computing roots of a series of univariate polynomial systems.

We apply a generic linear transformation to the coordinates, $(x_0, x_1, \ldots, x_n)$. This is to insure that the u-resultant does not vanish identically after we have specialized some of the variables. The domain of the modified system of equations is suitably adjusted.

Let

$$f_1(u_0) = R(u_0, 1, 0, \ldots, 0)$$

be a polynomial of degree d. It is assumed that $f_1(u_0)$ does not vanish identically, owing to the linear transformation on the coordinates. Since it corresponds to a projection of the u-resultant, it can be factored into linear factors of the form

$$q{u_0}^m(u_0 + \alpha_{11})(u_0 + \alpha_{12})\ldots(u_0 + \alpha_{1q}),$$

where each α_{1i} corresponds to the projection on the x_1 coordinate. Furthermore, $\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1q}$ are the roots of $f_1(u_0)$. However, we are only interested in roots corresponding to the interval, $x_1 \in [a_1, b_1]$. $f_1(u_0)$ has been expressed in terms of matrix determinants and to compute its roots we need to expand the determinants and compute the roots of the resulting univariate polynomials. As a result we use the interpolation algorithm presented in the previous section for computing the determinants and the Sturm sequence or Uspensky's algorithm method to compute the roots of the resulting polynomial in $[a_1, b_1]$.

Let there be p_1 such roots, $L_1 = (\alpha_{1j}, \ldots, \alpha_{1p_1})$ of $f_1(u_0)$ in $[a_1, b_1]$. Similarly we compute the roots of

$$f_2(u_0) = R(u_0, 0, 1, 0, \ldots, 0)$$

in the interval $[a_2, b_2]$, say $L_2 = (\alpha_{21}, \alpha_{22}, \ldots, \alpha_{2p_2})$, where p_2 corresponds to the number of roots in the interval. However, two projections are not enough for establishing the correspondence between the projections on u_1 and u_2 coordinates and therefore, we take a generic combinations of these two coordinates. Let

$$f_{1,2}(u_0) = R(u_0, k_1, k_2, 0, \ldots, 0),$$

where k_1 and k_2 are two positive random numbers. Moreover, $f_{1,2}(u_0)$ can be factored into linear factors of the form

$$f_{1,2}(u_0) = qu_0^m(u_0 + \beta_1)\ldots(u_0 + \beta_q),$$

where $\beta_i = \alpha_{1i}k_1 + \alpha_{2i}k_2$. Let $L_{1,2} = (\beta_1, \beta_2, \ldots, \beta_{p_{1,2}})$ be its roots in the interval $[k_1a_1 + k_2a_2, k_1b_1 + k_2b_2]$. To establish the correspondence between the projections on u_1 and u_2 of the actual roots, we compute all the combinations of the form $k_1\alpha_1 + k_2\alpha_2$, where $\alpha_1 \in L_1$ and $\alpha_2 \in L_2$ and compare them with the elements in $L_{1,2}$. Since our projection on u_1 and u_2 is a generic projection, it is reasonable to assume that the exact matches correspond to the projections of the roots of F_i's on x_1 and x_2 coordinates.

In a similar manner, we compute the roots of $f_3(u_0), f_{1,2,3}(u_0), \ldots, f_n(u_0), f_{1,2,\ldots,n-1,n}(u_0)$ in the corresponding intervals, where

$$f_i(u_0) = R(u_0, u_1, \ldots, u_n)_{u_1=0,\ldots,u_{i-1}=0,u_i=1,u_{i+1}=0,\ldots,u_n=0}$$

and

$$f_{1,2,\ldots,j}(u_0) = R(u_0, u_1, \ldots, u_n)_{u_1=k_1,\ldots,u_j=k_j,u_{j+1}=0,\ldots,u_n=0}.$$

k_1, k_2,...,k_j are random positive integers. These roots can be used to compute the rest of the x_i coordinates of the solutions of the original system of equations. It is possible that the resulting solution set contains some extraneous solutions. As a result, we back substitute the roots in the original system of equations to eliminate the extraneous roots from the solution set.

4.1. Resultant Formulations

In the previous section we reduced the problem of computing solutions of a system of multivariate polynomials (in a domain of interest) to finding roots of univariate polynomials in suitable intervals. In this section, we show that the univariate polynomials correspond exactly to a determinant. This hold for special formulations of resultants, known for n = 2,3,4,5,6, which express it as a determinant of a matrix.

Given n homogeneous equations, $F_1(x_0, x_1, \ldots, x_n), F_2(x_0, x_1, \ldots, x_n), \ldots, F_n(x_0, x_1, \ldots, x_n)$, we let

$$F_{n+1}(x_0, x_1, \ldots, x_n) = u_0x_0 + u_1x_1 + \ldots + u_nx_n.$$

Macaulay's formulation expresses the resultant of $F_1, \ldots, F_{n+1}$ as a ratio of determinants of two matrices, say M and D. However, the entries of D are independent of the coefficients of F_{n+1} and therefore consist of numeric constants only. There are two possible cases depending on the fact whether D is singular or not.

D is non-singular: The resultant corresponds exactly to the determinant of M. The univariate polynomials, $f_i(u_0)$ are obtained by specializing the variables $u_1, u_2, \ldots, u_n$ in the matrix M and computing the determinant using the interpolation algorithm.

D is singular: As a result matrix M is singular, too. The resultant of the given system is computed using perturbation techniques (Canny, 1988). In particular, it corresponds exactly to the constant term of the ratio of characteristic polynomials of M and D. In our case, we can compute the characteristic polynomial of M, corresponding to Determinant$(M - sI)$, after substituting for $u_1, \ldots, u_n$ and the problem corresponds to interpolating a bivariate determinant. Divide this polynomial by Determinant$(D - sI)$ and $f_i(u_0)$ corresponds to the term independent of s in

$$\frac{\text{Determinant}(M - sI)}{\text{Determinant}(D - sI)}.$$

We illustrate the algorithm on the following example.

4.2. Example

Let's consider the intersection of a sphere and two paraboloids, represented by the following equations

$$\begin{aligned} F_1(x,y,z,w) &= x^2+y^2+z^2-w^2 \\ F_2(x,y,z,w) &= -x^2-y^2+zw \\ F_3(x,y,z,w) &= -x^2-z^2+yw. \end{aligned}$$

We are only interested in all the real solutions of the given system.

Lets use a linear transformation on the system:

$$\begin{aligned} x &= \overline{x}+\overline{y}+2\overline{z} \\ y &= \overline{x}-\overline{y} \\ z &= -2\overline{x}+\overline{y}+3\overline{z} \\ w &= \overline{w} \end{aligned}$$

and the transformed equations are

$$\begin{aligned} F_1^{'}(\overline{x},\overline{y},\overline{z},\overline{w}) &= -\overline{w}^2+6\overline{x}^2-4\overline{xy}+3\overline{y}^2-8\overline{xz}+10\overline{yz}+13\overline{z}^2 \\ F_2^{'}(\overline{x},\overline{y},\overline{z},\overline{w}) &= -2\overline{xw}-2\overline{x}^2+\overline{yw}-2\overline{y}^2+3\overline{zw}-4\overline{xz}-4\overline{yz}-4\overline{z}^2 \\ F_3^{'}(\overline{x},\overline{y},\overline{z},\overline{w}) &= \overline{xw}-5\overline{x}^2-\overline{yw}+2\overline{xy}-2\overline{y}^2+8\overline{xz}-10\overline{yz}-13\overline{z}^2 \end{aligned}$$

Let us use the additional equation

$$F_4^{'}(\overline{x},\overline{y},\overline{z},\overline{w}) = u_0\overline{w}+u_1\overline{x}+u_2\overline{y}+u_3\overline{z}.$$

We use the Macaulay's formulation (Macaulay, 1902) to compute the resultant of $F_i^{'}(\overline{x},\overline{y},\overline{z},\overline{w})$, which expresses it as a ratio of two determinants, $R(u_0,u_1,u_2,u_3) = \text{Determinant}(M)/\text{Determinant}(D)$, where

$$M =$$

$$
\begin{bmatrix}
6&-4&-8&0&3&10&0&13&0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&6&0&0&-4&-8&0&0&0&0&3&10&0&0&0&0&0&0&0&13&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0\\
0&0&6&0&0&-4&0&-8&0&0&0&3&0&0&0&0&0&0&0&10&13&0&0&0&0&0&0&0&0&-1&0&0&0&0&0\\
0&0&0&6&0&0&-4&0&-8&0&0&0&3&0&0&0&0&0&0&0&0&13&0&0&0&0&0&10&0&0&-1&0&0&0&0\\
0&0&0&0&6&0&0&0&0&0&-4&-8&0&3&10&0&13&0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&6&0&0&0&0&0&-4&0&0&3&0&10&0&0&-8&0&0&13&0&0&0&0&0&0&0&0&-1&0&0&0\\
0&0&0&0&0&0&6&0&0&0&0&0&-4&0&0&3&0&10&0&0&0&0&0&13&0&0&0&-8&0&0&0&0&-1&0&0\\
0&0&0&0&0&0&0&6&0&0&0&0&0&0&0&0&3&0&0&-4&-8&0&10&0&13&0&-1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&6&0&0&0&0&0&0&0&0&3&0&0&0&-8&0&10&0&13&0&-4&0&0&0&0&0&-1&0\\
0&0&0&0&0&0&0&0&0&6&0&0&0&0&0&0&0&0&3&0&0&0&0&0&0&0&13&0&-4&-8&0&10&0&0&-1\\
0&-2&0&0&0&-4&-2&0&0&0&-2&-4&1&0&0&0&0&0&0&-4&0&0&0&0&0&0&0&3&0&0&0&0&0&0&0\\
0&0&-2&0&0&0&0&-4&-2&0&0&-2&0&0&0&0&0&0&0&-4&-4&3&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&-2&0&0&0&0&-4&-2&0&0&-2&0&0&0&0&0&0&0&0&-4&0&0&0&0&0&-4&1&3&0&0&0&0&0\\
0&0&0&0&-2&0&0&0&0&0&0&-4&-2&-2&-4&1&-4&3&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&-2&0&0&0&0&0&0&0&0&-2&0&-4&1&0&-4&0&0&-4&3&0&0&0&-2&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-2&0&0&0&0&0&0&0&0&-2&0&-4&1&0&0&0&0&-4&0&0&0&-4&-2&0&0&3&0&0&0\\
0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&0&-2&0&0&0&-4&-2&-4&1&-4&3&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&0&-2&0&0&0&-4&0&-4&0&-4&3&0&0&-2&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&-4&0&0&-4&-2&-4&1&3&0\\
0&-5&0&0&2&8&1&0&0&0&-2&-10&-1&0&0&0&0&0&0&-13&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&-5&0&0&2&0&8&1&0&0&-2&0&0&0&0&0&0&0&-10&-13&0&0&0&0&0&0&-1&0&0&0&0&0&0&0\\
0&0&0&-5&0&0&2&0&8&1&0&0&-2&0&0&0&0&0&0&0&0&-13&0&0&0&0&0&-10&-1&0&0&0&0&0&0\\
0&0&0&0&0&-5&0&0&0&0&0&2&0&0&-2&0&-10&-1&0&8&0&0&-13&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-5&0&0&0&0&0&2&0&0&-2&0&-10&-1&0&0&0&0&-13&0&0&0&8&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&-5&0&0&0&0&0&0&0&0&-2&0&0&2&8&1&-10&-1&-13&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&-5&0&0&0&0&0&0&0&0&-2&0&0&0&8&0&-10&0&-13&0&2&0&1&0&-1&0&0&0\\
0&0&0&0&0&0&0&0&0&-5&0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&-13&0&2&8&1&-10&-1&0&0\\
0&0&0&0&0&u_1&0&0&0&0&0&u_2&0&0&0&0&0&0&0&u_3&0&0&0&0&0&0&0&u_0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&u_1&0&0&0&0&0&u_2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_3&u_0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&u_1&0&0&0&0&0&0&0&0&0&0&0&0&u_3&0&0&0&0&0&u_2&0&u_0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&u_1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_2&u_3&u_0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_2&0&0&0&0&0&u_3&0&0&0&u_1&0&0&0&u_0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_2&0&0&0&0&0&0&0&0&0&u_1&0&0&u_3&u_0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_3&0&0&u_1&0&u_2&0&u_0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&u_1&0&u_2&u_3&u_0
\end{bmatrix}
$$

and

$$
D = \begin{bmatrix}
6&0&-4&0&-8&0&3&0&0&0&0&13&0&0&0\\
0&6&0&0&0&0&0&0&13&0&-1&0&0&0&0\\
0&0&6&0&0&0&-4&3&0&10&0&0&13&0&0\\
0&0&0&6&0&0&0&0&3&0&0&0&0&0&-1\\
0&0&0&0&6&0&0&0&0&3&0&-8&10&13&0\\
0&0&0&0&0&6&0&0&0&0&3&0&0&0&13\\
-2&0&0&0&-4&-2&-2&0&0&0&0&-4&0&0&0\\
0&0&-2&0&0&0&0&-2&0&-4&1&0&-4&0&0\\
0&0&0&-2&0&0&0&0&-2&0&0&-2&1&3&0\\
0&0&0&0&-2&0&0&0&0&-2&0&-4&-4&-4&3\\
0&0&0&0&0&-2&0&0&0&0&-2&0&0&0&-4\\
-5&0&2&0&8&1&-2&0&0&0&0&-13&0&0&0\\
0&0&-5&0&0&0&2&-2&0&-10&-1&0&-13&0&0\\
0&0&0&0&-5&0&0&0&0&-2&0&8&-10&-13&0\\
0&0&0&0&0&-5&0&0&0&0&-2&0&0&0&-13
\end{bmatrix}.
$$

In this case, D is a non-singular matrix and its entries are all numeric. As a result, $R(u_0, u_1, u_2, u_3) = \text{Determinant}(M)$. To compute the roots we take the projections along different coordinates, use interpolation algorithms followed by root finding.

Let $f_1(u_0) = R(u_0, 1, 0, 0)$. The resulting polynomial has only two real solutions:

$$L_1 = (0.48586827, -0.48586827).$$

Similarly, we compute the real roots of $f_2(u_0) = R(u_0, 0, 1, 0)$ and

$$L_2 = (-0.87748883, -0.51308763).$$

To establish a correspondence between the roots, we consider $f_{1,2}(u_0) = R(u_0, 3, 4, 0)$, (where 3 and 4 are chosen randomly) and its real roots are

$$L_{1,2} = (-4.28831990, -1.73751148).$$

We consider all pairs of the form $(3\alpha_1 + 4\alpha_2)$, where $\alpha_1 \in L_1$ and $\alpha_2 \in L_2$ and match them with the elements of $L_{1,2}$. The matches result in two possible solutions of the form

$$\begin{aligned} (\overline{w}, \overline{x}, \overline{y}, \overline{z}) &= (1, -0.2594548, -0.87748883, u_3) \\ (\overline{w}, \overline{x}, \overline{y}, \overline{z}) &= (1, 0.10494635, -0.51308763, u_3). \end{aligned}$$

Finally we compute the roots of $f_3(u_0) = R(u_0, 0, 0, 1)$ and $f_{1,2,3}(u_0) = R(u_0, -2, 3, 1)$ and express them as

$$L_3 = (0.44700477, 0.32553770)$$

and

$$L_{1,2,3} = (-1.78801910, -1.30215083).$$

After considering the possible matches, the real roots of the original system are (in the order (x, y, z, w))

$$\begin{aligned} (\overline{w}, \overline{x}, \overline{y}, \overline{z}) &= (1, -0.2594548, -0.87748883, 0.32553770) \\ (\overline{w}, \overline{x}, \overline{y}, \overline{z}) &= (1, 0.10494635, -0.51308763, 0.44700477). \end{aligned}$$

Given the solutions of the transformed equations, the corresponding solutions of the original equations are

$$\begin{aligned} (x, y, z, w) &= (0.4858682723, 0.618033989, 0.618033989, 1) \\ (x, y, z, w) &= (-0.4858682723, 0.618033989, 0.618033989, 1). \end{aligned}$$

5. Conclusion

In this paper we have presented algorithms to efficiently compute the resultants of polynomial equations and used them along with algorithms for finding roots of univariate polynomials to find roots of multivariate systems. We demonstrated the performance of the algorithm on known benchmarks of implicitizing rational parametric surfaces and show that using this algorithm it is possible to implicitize parametric surfaces used in geometric modeling systems in reasonable time. Using properties of multivariate interpolation algorithms, modular arithmetic and matrices and determinants, it is possible to derive a tight bound on the running time of the computation. The algorithm is numeric in nature and no intermediate symbolic expressions are generated. As a result, the storage requirements of the algorithm are functions of input and output polynomials and matrices corresponding to resultant formulations. In (Manocha and Canny, 1992b), the linear algebra formulation of resultants has been utilized along with matrix computations and multivariate interpolation.

References

S. S. Abhyankar (1976). Historical ramblings in algebraic geometry and related algebra. *American Mathematical Monthly*, 83:409–448.

C. Bajaj, T. Garrity, and J. Warren (1988). On the applications of multi-equational resultants. Technical Report CSD-TR-826, Department of Computer Science, Purdue University.

M. Ben-or and P. Tiwari (1988). A deterministic algorithm for sparse multivariate polynomial interpolation. In *ACM Symposium on Theory of Computing*, pages 301–309.

D. N. Bernshtein (1975). The number of roots of a system of equations. *Funktsional'nyi Analiz i Ego Prilozheniya*, 9(3):1–4.

B. Buchberger (1985). Grőbner bases: An algorithmic method in ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Co.
B. Buchberger (1989). Applications of groebner bases in non–linear computational geometry. In D. Kapur and J. Mundy, editors, *Geometric Reasoning*, pages 415–447. MIT Press.
J. Canny and I. Emiris (1993). An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAECC*. To appear.
J. Canny, E. Kaltofen, and Y. Laksman (1989). Solving system of nonlinear polynomial equations faster. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*.
J.F. Canny (1988). *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press.
J.F. Canny (1990). Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9:241–250.
A.L. Dixon (1908). The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236.
X.S. Gao and S.C. Chou (1990). Implicitization of rational parametric equations. Technical report, Department of Computer Science TR-90-34, University of Texas at Austin.
I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky (1990). Discriminants of polynomials in several variables and triangulations of newton polytopes. *Algebra i analiz*, 2:1–62.
C.M. Hoffmann (1990a). A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532.
C.M. Hoffmann (1990b). Algebraic and numeric techniques for offsets and blends. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computations of Curves and Surfaces*, pages 499–528. Kluwer Academic Publishers.
Jean-Pierre Jouanolou (1991). *Le Formalisme du Résultant*, volume 90 of *Advances in Mathematics*.
M. Kalkbrener (1991). *Three Contributions to Elimination Theory*. PhD thesis, Johannes Kepler Universitat, Linz, Austria.
E. Kaltofen and Y.N. Lakshman (1988). Improved sparse multivariate polynomial interpolation algortihms. In *Lecture Notes in Computer Science*, volume 358, pages 467–474. Springer-Verlag.
D. Kapur and Y.N. Laksman (1992). Elimination methods: An introduction. In D. Kapur B. Donald and J. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*. Academic Press.
D. Knuth (1981). *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley.
R. Loos (1983). Computing rational zeros of integral polynomials by p-adic expansion. *SIAM Journal on Computing*, 7:286–293.
F.S. Macaulay (1902). On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.
F.S. Macaulay (1921). Note on the resultant of a number of polynomials of the same degree. *Proceedings of London Mathematical Society*, pages 14–21, June 1921.
F.S. Macaulay (1964). *The Algebraic Theory of Modular Systems*. Stechert–Hafner Service Agency, New York.
D. Manocha (1992). *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley.
D. Manocha and J.F. Canny (1991). A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516. Special issue on Solid Modeling.
D. Manocha and J.F. Canny (1992a). Algorithms for implicitizing rational parametric surfaces. *Computer Aided Geometric Design*, 9:25–50.
D. Manocha and J.F. Canny (1992b). Multipolynomial resultant algorithms and linear algebra. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 158–167.
D. Manocha and J. Demmel (1992). Algorithms for intersecting parametric and algebraic curves. In *Graphics Interface '92*, pages 232–241.
E. Mayr and A. Meyer (1982). The compleixty of the word problem in commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329.
A.P. Morgan (1987). *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice–Hall, Englewood Cliffs, New Jersey.
F. Morley (1925). The eliminant of a net of curves. *American Journal of Mathematics*, 47:91–97.
F. Morley and A.B. Coble (1927). New results in elimination. *American Journal of Mathematics*, 49:463–488.
J.F. Ritt (1950). *Differential Algebra*. AMS Colloquium Publications.
G. Salmon (1885). *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York.
B. Sturmfels (1991). Sparse elimination theory. In D. Eisenbud and L. Robbiano, editors, *Computational Algebraic Geometry and Commutative Algebra*. Cambridge University Press.
B. Sturmfels and A. Zelevinsky (1992). Multigraded resultants of sylvester type. *Journal of Algebra*.
B.L. Van Der Waerden (1950). *Modern Algebra (third edition)*. F. Ungar Publishing Co., New York.

C. Wampler and A.P. Morgan (1991). Solving the 6r inverse position problem using a generic-case solution methodology. *Mechanisms and Machine Theory*, 26(1):91–106.

H.S. White (1909). Bezout's theory of resultants and its influence on geometry. *Bulletin of the American Mathematical Society*, 112:59–68.

W. Wu (1984). On the decision problem and the mechanization of theorem proving in elementary geometry. *Scientia Sinica*, 21:150–172. Also in Bledsoe and Loveland, eds. *Theorem Proving: After 25 years, Contemporary Mathematics*, **29**, 213–234.

R. Zippel (1990). Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9:375–403.