MultiPolynomial Resultant Algorithms

Dinesh Manocha[1] *John F. Canny*[1]

Computer Science Division
University of California
Berkeley, CA 94720

Abstract: 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 applying 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. Moreover, we use multipolynomial resultants for computing the real or complex solutions of non-linear polynomial equations. It reduces the problem to computing eigenvalues of matirces. We also discuss implementation of these algorithms in the context of certain applications.

[1]This research was supported in part by David and Lucile Packard Fellowship and in part by National Science Foundation Presidential Young Investigator Award (# IRI-8958577).

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). 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 [Ma64; Sal885; Wd50] 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*[1] 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.

Recently the technique of Gröbner bases has received much attention as an algorithmic method for determining properties of polynomial equations [Bu85; Bu89]. Its applications include ideal membership besides eliminating a set of variables or computing the numerical solutions of a system of polynomial equations. One of the main difficulties in using Gröbner bases is that the method may be slow for even small problems. In the worst case, its running time may can be doubly exponential in the number of variables [MM82]. 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 Gröbner bases have been described in [Ho89; Ho90]. In the first algorithm a Gröbner 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 [Ho90]. It has been applied for implicitizing rational parametric surfaces. To implicitize rational parametric bicubic patches, the algorithm takes about 10^5 seconds on a Symbolics 3650, which would be considered impractical for geometric modeling applications. As far as the problem of

[1]Other authors have used the term multiequational resultants [BGW88]

computing roots of polynomials is concerned, issues of numerical stability make Gröbner bases unattractive for computing numeric solutions of polynomial equations [Mo90].

For numerical applications are concerned 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 [Mo87; WM90]. Although continuation methods have a good theoretical background and in some cases a high degree of computational reliability (in some cases), 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 number* (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.

Multipolynomial resultant algorithms provide the most efficient methods (as far as asymptotic complexity is concerned) for solving system of polynomial equations or eliminating variables [BGW88]. Their main advantage lies in the fact that the resultant can always be 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 [MC90; MC91a; MC91b].

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 can be used for interpolating polynomials from their values and expanding symbolic determinants. We also describe an efficient implementation of the algorithm and present its performance for applications like implicitization. To implicitize rational bicubic patches it takes about 575 and 138 seconds[2] on a Sun-4 and IBM RS/6000, respectively. Moreover, we effectively use resultants for computing the numeric solutions of a system of polynomial equations in the domain of interest. In this case, we reduce the problem to computing eigenvalues of a matrix. Efficient algorithms for computing eigenvalues are well known and their implementations are available as part of standard packages like LAPACK and EISPACK [De89; GV89]. Furthermore, in the context of floating point computations, the numerical accuracy of these operations is well understood. The rest of the paper is organized as follows. In Section 2 we give a brief preview of different formulations used for

[2]These timings correspond to the examples in [Ho90]. The actual running time is a function of coefficient size of a parametrization.

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 polynomial 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 [Ma02]. If all the polynomials have the same degrees, an improved formulation is given in [Ma21]. Many a time both the 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 [Ca87]. The ratio of the two characteristic polynomials is termed the *Generalized Characteristic Polynomial*, and the resultant corresponds to its constant term [Ca90]. 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 [Ca90; MC90; MC91b]. 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 [Di08; Jo89; Mo25; MC27]. Historical accounts of resultants and elimination theory are presented in [Ab76; Wh09].

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 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 [MC90]. 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 the 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 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

$$C_R(n,d) = \binom{n+d-1}{d}.$$

We represent the determinant as

$$F(x_1, x_2, \ldots, x_n) = c_1 m_1 + c_2 m_2 + \ldots + c_q m_q, \tag{1}$$

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) the problem is reduced to that of *multivariate interpolation* [BT88; Zi90]. 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 $a_i = F(p_1^i, p_2^i, \ldots, p_n^i)$, $i = 1, q$. a_i's are computed by Gauss elimination. Thus, the problem of computing c_i's is reduced to solving a Vandermonde system system

$VC = A$, where

$$V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_k \\ \vdots & \vdots & \vdots & \vdots \\ b_1^{q-1} & b_2^{q-1} & \dots & b_q^{q-1} \end{pmatrix}, \quad C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_q \end{pmatrix}, \quad A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_q \end{pmatrix}. \qquad (2)$$

Computing each a_i takes $O(m^3)$ time, where m is the order of the matrix. As far as solving Vandermonde systems is concerned, simple algorithms of time complexity $O(q^2)$ and $O(q)$ space requirements are known [Zi90]. In [KL88] 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* [Zi90]. 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 is a sparse polynomial.

A sparse interpolation algorithm for such problems has been presented in [BT88]. Its time and space complexity have been improved in [KL88].

3.1.1 Sparse Interpolation

The algorithm in [BT88] needs an upper bound $T \geq q$ on the number of non-zero monomials in $F(x_1, \dots, x_n)$. Given $(p_1^j, p_2^j, \dots, p_n^j)$, it computes the monomial values $b_i = p_1^{d_{1,i}} p_2^{d_{2,i}} \dots p_n^{d_{n,i}}$ 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

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

Its coefficients, g_i's, are computed by solving a Toeplitz system of equations [BT88]. 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 to b_i's. The roots are computed using p-adic root finder [Lo83]. The running time of the resulting algorithm for polynomial interpolation is $(ndT^3log(n) + m^3T)$. The dominating step is the polynomial root finder which takes $O(T^3log(B))$, 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

[Zi90] presents a probabilistic algorithm for interpolation which does not expect any information about the non-zero terms of the polynomial being interpolated. It expects a bound on the degree of each variable in any monomial. Such a bound is easy to compute for a symbolic determinant. In fact, this bound is tight. The algorithm proceeds inductively, uses 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 [Zi90].

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. Let us assume we are at the kth stage and have interpolated a polynomial in k variables. The resulting polynomial is of the form

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

In this case, $\alpha \leq q$. 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 $F_k(x_1, \ldots, x_k)$, it represents F_{k+1} as

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

where each each $h_i(x_{k+1})$ is a polynomial of degree d_{k+1}. 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

$$F_1(x_1) = F(x_1,r_2,\ldots,r_n) = c_0 + c_1x_1^3 + \ldots + c_\beta x_1^{d_1}.$$

$F_2(x_1,x_2)$ 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

$$\frac{nd^2q^2}{S}, \tag{3}$$

where $d = max(d_1,\ldots,d_n)$ [Zi90].

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 in [KL88].

A deterministic version of this algorithm of complexity $O(ndq^2T + m^3ndqT)$ is given in [Zi90] 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, the resulting algorithm has better complexity than the deterministic sparse or dense interpolation algorithms. Due to simplicity and time complexity of the algorithm, we use the probabilistic version for our implementation. However, we give user the flexibility of imposing a bound on the probability of failure and choose the finite fields appropriately. Furthermore, we introduce simple, randomized checks in the algorithm to detect wrong answers. These checks correspond to substituting random values for $x_1, \ldots, x_n$ in the matrix entries and computing the resulting numeic determinant. We verify the result by substituting the same values for $x_1, \ldots, x_n$ in the polynomial obtained after interpolation and comparing it with numeric determinant.

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 [Kn81]. 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 [Kn81]

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

While using chinese remainder theorem it is assumed that $r_{i,k}$ are integers lying in the range $(-\frac{p_1p_2\cdots p_k}{2}, \frac{p_1p_2\cdots p_k}{2})$. At this stage we compare the bignums corresponding to the $(k-1)$ and k stage. If

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

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

$$2|c_j| < p_1p_2 \dots p_{l+1}$$

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 $c_{i,1}, c_{i,2}, \dots, c_{i,j}$, 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 $p_1, p_2, \dots, p_l$ such that

$$p_1p_2 \dots p_{l-1} < 2|c_i| < p_1p_2 \dots p_l.$$

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 $(-\frac{p_1\cdots p_j}{2}, \frac{p_1\cdots p_j}{2})$.

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 $r_{i,k} \in (-\frac{p_1\cdots p_{k-1}}{2}, \frac{p_1\cdots p_{k-1}}{2})$. Since, $r_{i,k}$ are uniformly distributed over $(-\frac{p_1\cdots p_k}{2}, \frac{p_1\cdots p_k}{2})$, 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

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

In the worst case, all coefficients of $F(x_1,\dots,x_n)$ 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 be even be made to 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,
$$\begin{aligned}(a * b)\ mod\ p &= ((((r1 * 2^{32})\ mod\ p) + (r2\ mod\ p))\ mod\ p \\ &= (((r1 * c)\ mod\ p) + (r2\ mod\ p))\ mod\ p,\end{aligned}$$
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 (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)). 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 [MC90; MC91b]. Some experiments with the implementations of Gröbner bases and resultants in Macsyma 414.62 on a Symbolics lisp machine (with 16MB main memory and 120MB virtual memory) are described in [Ho90]. 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 [Ho90].

Lets consider the bicubic parametrization given in [Ho90]. 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 [Di08] and the resultant corresponds to the determinant of

$$\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} \mathrm{x} +
\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}$$

$$+ \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} \mathrm{z} +
\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

$$q = \binom{d+3}{3},$$

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.

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^2t^2	8	165	4 sec.	2 sec.
s^3t^3	18	1330	114 sec.	26 sec.
s^3t^4	24	2925	430 sec.	118 sec.

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

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 $M =$

$$\begin{bmatrix}
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{bmatrix}$$

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 [Wd50]. 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 $u_0, u_1, \ldots, u_n$. After specialization, these entries are univariate polynomials in u_0 and the problem of computing roots of the univariate polynomial corresponding to the determinant (or ratio of determinants) can be reduced to an eigenvalue problem. Efficient algorithms for computing the eigenvalues of matrices are given in [GV89] and good implementations are available as part of standard packages like EISPACK and LAPACK [De89].

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

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

where each α_{1i} corresponds to the projection on the x_1 coordinate. We are only interested in roots lying in the interval $[a_1, b_1]$. However, $f_1(u_0)$ has been expressed in terms of matrix determinants. 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 method to compute the roots of the resulting polynomial in $[a_1, b_1]$. However, this procedure is slow and becomes unattractive in the context of floating point computations. In the next section we reduce the problem of root finding to computing eigenvalues of matrices.

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 $L_2 = (\alpha_{21}, \alpha_{22}, \ldots, \alpha_{2p_2})$. 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 and 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. Let $L_{1,2} = (\beta_1, \beta_2, \ldots, \beta_{p_{1,2}})$ be its roots in the interval $[k_1 a_1 + k_2 a_2, k_1 b_1 + k_2 b_2]$. 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 Reduction to Eigenvalue Problem

In the previous section we reduced the problem of computing solutions of a system of multivariate polynomials (in the domain of interest) to finding roots of univariate polynomials in suitable intervals. The univariate polynomials, like $f_i(u_0)$, are expressed as a determinant or as a ratio of two determinants and we are interested in roots lying in the interval $[a, b]$. Let us consider the case when it is expressed as a ratio of two determinants and the corresponding matrices are denoted as $M(u_0)$ and $D(u_0)$. If the resultant corresponds to a determinant of a matrix, Determinant$(D(u_0)) = 1$. Each entry of $M(u_0)$ and $D(u_0)$ is a polynomial in u_0. Let its degree be bounded by d. Depending upon the value of *Determinant*$(D(u_0))$ there are two possible cases.

- *Determinant*$(D(u_0)) \neq 0$. Thus,

$$f_i(u_0) = \frac{\text{Determinant}(M(u_0))}{\text{Determinant}(D(u_0))}.$$

  Let S_1 and S_2 be the solution sets corresponding to the roots of Determinant$(M(u_0)) = 0$ and Determinant$(D(u_0)) = 0$ lying in the interval $[a, b]$, respectively. As a result, the roots of $f_i(u_0)$ correspond to $S_1 \setminus S_2$. We reduce the problem of computing S_1 or S_2 to an eigenvalue problem.

- *Determinant*$(D(u_0)) = 0$. As a result *Determinant*$(M(u_0)) = 0$. It is possible to use perturbation technique and thereby express the resultant as the constant term of the ratio of characteristic polynomials of $M(u_0)$ and $D(u_0)$ [Ca90]. However, this procedure may be expensive and we consider a minor of $M(u_0)$, say $\overline{M}(u_0)$ of maximum possible rank (among all its minors), such that *Determinant*$(\overline{M}(u_0)) \neq 0$. It follows from the construction, $f_i(u_0)$ divides *Determinant*$(\overline{M}(u_0))$. The roots of *Determinant*$(\overline{M}(u_0))$ are computed by reducing it to an eigenvalue problem. For each root $u_0 = \alpha$, we compute the constant term of the ratio of characteristic polynomials of $M(\alpha)$ and $D(\alpha)$. It turns out that $f_i(\alpha) = 0$ iff the constant term is zero.

Let us assume that $M(u_0)$ is a matrix of order n. Each entry of $M(u_0)$ is a polynomial of degree d and it can therefore, be represented as

$$M(u_0) = u_0^d M_d + u_0^{d-1} M_{d-1} + \ldots + u_0 M_1 + M_0, \tag{4}$$

where M_i's are matrices of order n with numeric entries. Let us assume that M_d is a non-singular matrix. As a result, the roots of the following equations are equivalent

$$\text{Determinant}(M(u_0)) = 0,$$
$$\text{Determinant}(M_d^{-1})\ \text{Det}(M(u_0)) = 0.$$

Let

$$\overline{M}(u_0) = u_0^d I_n + u_0^{d-1}\overline{M}_{d-1} + \ldots + u_0\overline{M}_1 + \overline{M}_0, \tag{5}$$

where

$$\overline{M}_i = M_d^{-1} M_i, \qquad 0 \leq i < d$$

and I_n is an $n \times n$ identity matrix. Given $\overline{M}(u_0)$, we use Theorem 1.1 [GLR82] to construct a matrix of the form

$$C = \begin{bmatrix} 0 & I_n & 0 & \ldots & 0 \\ 0 & 0 & I_n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & I_n \\ -\overline{M}_0 & -\overline{M}_1 & \overline{M}_2 & \ldots & \overline{M}_{d-1} \end{bmatrix}, \tag{6}$$

such that the eigenvalues of C correspond exactly to the roots of $\text{Det}(\overline{M}(u)) = 0$. C is a numeric matrix of order dn.

If M_d is a singular matrix the roots of the matrix polynomial represented by (4) can be obtained by constructing companion matrices [GLR82, Section 7.2]

$$C_1 = \begin{bmatrix} 0 & I_n & 0 & \ldots & 0 \\ 0 & 0 & I_n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & I_n & 0 \\ 0 & 0 & \ldots & M_d & 0 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 0 & -I_n & 0 & \ldots & 0 \\ 0 & 0 & -I_n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & -I_n \\ M_0 & M_1 & \ldots & M_{d-2} & M_{d-1} \end{bmatrix}.$$

The roots of the polynomial $Determinant(C_1 u_0 + C_2) = 0$ correspond to the roots of $Determinant(M(u_0)) = 0$. However, C_1 is a singular matrix and the problem of computing roots correspond to a *generalized eigenvalue problem* [GV89].

4.1.1 Implementation

The bottleneck of the root finding algorithm are the eigenvalue routines. We used EISPACK routines for computing the eigenvalues of matrices. Many special purpose algorithms are available for computing the eigenvalues of matrices, which make use of the structure of the matrix. As far as matrix C in (6) is concerned, we treat it as a general unsymmetric matrix. We used the routine RG from EISPACK for computing the eigenvalues [GBDM77]. Given a general unsymmetric matrix, it makes use of balancing techniques, reduces it to upper Hessenberg form and uses the shifted QR algorithm on the resulting matrix to compute the eigenvalues [GV89]. The current implementation of these routines compute all the eigenvalues. The performance of eigenvalue computation routines for matrices of different order (generated randomly) are given in Table II. The timings correspond to the implementation on an IBM RS/6000. As far as generalized eigenvalue problems are concerned, we used the routine RGG from EISPACK. Details of the algorithm being used are given in [GV89].

Order of Matrix	Time in seconds
15	$8631.839844 \times 10^{-6}$
20	$15717.63965 \times 10^{-6}$
25	$25753.00000 \times 10^{-6}$
30	$38763.23828 \times 10^{-6}$
35	$57124.16016 \times 10^{-6}$
40	$77398.03906 \times 10^{-6}$
45	$103343.5234 \times 10^{-6}$
50	$133956.2344 \times 10^{-6}$
55	$165395.0469 \times 10^{-6}$
60	$212041.2812 \times 10^{-6}$
65	$262103.1250 \times 10^{-6}$

Table II
The performance of eigenvalue computation routines

4.1.2 Example

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

$$F_1(x, y, z, w) = x^2 + y^2 + z^2 - w^2$$

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

We are only interested in the real solutions.

Let

$$F_4(x,y,z,w) = u_0 w + u_1 x + u_2 y + u_3 z.$$

We use the Macaulay's formulation [Ma02] to compute the resultant, which expressed it as a ratio of two determinants, $R(u_0, u_1, u_2, u_3) = Determinant(M)/Determinant(D)$, 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 numeric. As a result, $R(u_0, u_1, u_2, u_3) = Determinant(M)$. To compute the roots we take the projections along different coordinates and reduce the problem to a generalized eigenvalue problem.

Let $f_1(u_0) = R(u_0, 1, 0, 0)$. We specialize these values into the matrix and break it up into a generalized eigenvalue problem of the form $C_1 u_0 + C_2$, where C_1 and C_2 are numeric matrices. The resulting system has 27 eigenvalues at infinity and there are only two real solutions in the affine domain

$$L_1 = (-0.2594548, 0.10494635).$$

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 $L_{1,2}$. This results in matches of the form

$$(1, -0.2594548, -0.87748883, u_3), \quad (1, 0.10494635, -0.51308763, u_3).$$

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))

$$(-0.2594548, -0.87748883, 0.32553770, 1) \quad (-0.2594548, -0.87748883, 0.32553770, 1).$$

5 Conclusion

In this paper we have presented algorithms to efficiently compute the resultants of polynomial equations and using properties of matrices and determinants used them to compute the roots of a system of polynomial equations. As a result, it is possible to perform symbolic elimination from a given set of polynomial equations in a reasonable amount of time and space requirements. We have used these algorithms for implicitizing parametric surfaces, inverse kinematics, computing the configuration space for curved objects for robot motion planning and solving systems of non-linear equations.

6 Acknowledgements

We are grateful to Prof. J. Demmel and Prof. W. Kahan for productive discussions.

7 References

[Ab76] Abhyankar, S.S. (1976) "Historical ramblings in algebraic geometry and related algebra", *American Mathematical Monthly,* vol. **83**, pp. 409–448.

[BGW88] Bajaj, C., Garrity, T. and Warren, J. (November 1988) "On the applications of multi-equational resultants", Tech. report CSD-TR-826, Computer Science Deptt., Purdue University.

[Bu85] Buchberger, B. (1987) "Gröbner bases: An algorithmic method in polynomial ideal theory", in *Multidimensional Systems Theory,* edited by N.K. Bose, pp. 184–232, D. Reidel Publishing Co..

[Bu89] Buchberger, B. (1989) "Applications of Gröbner bases in non-Linear computational geometry", in *Geometric Reasoning,* eds. D. Kapur and J. Mundy, pp. 415–447, MIT Press.

[BT88] Ben-Or, M. and Tiwari, P. (1988) "A deterministic algorithm for sparse multivariate polynomial interpolation", *20th Annual ACM Symp. Theory of Comp.,* pp. 301–309.

[BSY91] Bernard, C., Soni, A. and Yee ,K. (1991) "Gauge Dependence of Quark Mass in Strong Coupling", to be published.

[Ca87] Canny, J. F. (1987) *The complexity of robot motion planning,* ACM Doctoral Dissertation award, MIT Press.

[Ca90] Canny, J. F. (1990) "Generalized characteristic polynomials", *Journal of Symbolic Computation,* vol. **9**, pp. 241–250.

[De89] Demmel, J. (1989) "LAPACK: A portable linear algebra library for supercomputers", *IEEE Control systems society workshop on computer-aided control system design,* Tampa, Florida.

[Di08] Dixon, A.L. (1908) "The eliminant of three quantics in two independent variables", *Proceedings of London Mathematical Society,* vol. **6**, pp. 49–69, 473–492.

[GLR82] Gohberg, I., Lancaster, P. and Rodman, L. (1982) *Matrix polynomials,* Academic Press, New York.

[GV89] Golub, G.H. and Van Loan, C. F. (1989) *Matrix computations,* The John Hopkins Press, Baltimore, Maryland.

[Ho90] Hoffmann, C. (1990) "Algebraic and numeric techniques for offsets and blends", in *Computation of Curves and Surfaces,* eds. W. Dahmen et. al., pp. 499–529, Kluwer Academic Publishers.

[Jo89] Jouanolou, Jean-Pierre (1989) "Le Formalisme Du Résultant", Department of Mathemetics, Université Louis Pasteur, France.

[KL88] Kaltofen, E. and Lakshman, Y.N. (1988) "Improved sparse multivariate polynomial interpolation algorithms", in *Lecture Notes in Computer Science,* vol. **358**, pp. 467–474, Springer-Verlag.

[KLW90] Kaltofen, E., Lakshman, Y.N. and Wiley, J. (1990) "Modular rational sparse multivariate polynomial interpolation", in Proceedings of *ISSAC'90* pp. 135–140, Addison-Wesley, Reading, Massachusets.

[Kn81] Knuth D. (1981) *The art of computer programming: seminumerical algorithms,* Addison-Wesley.

[Lo83] Loos, R. (1983) "Computing rational zeros of integral polynomials by p-adic expansion", *SIAM Journal on Computing,* vol. **7**, pp. 286–293.

[Ma02] Macaulay, F. S. (May 1902) "On some formula in elimination", *Proceedings of London Mathematical Society,* pp. 3–27.

[Ma21] Macaulay, F. S. (June 1921) "Note on the resultant of a number of polynomials of the same degree", *Proceedings of London Mathematical Society,* pp. 14–21.

[Ma64] Macaulay, F. S. (1964) *The algebraic theory of modular systems,* Stechert-Hafner Service Agency, New York.

[MC27] Morley, F. and Coble, A.B. (1927) "New results in elimination", *American Journal of Mathematics,* vol. **49**, pp. 463–488.

[MC90] Manocha, D. and Canny, J. (1990) "Algorithms for implicitizing rational parametric surfaces", to appear in Proc. of *IV IMA Conference on Mathematics of Surfaces,* Claredon Press, Oxford. Also available as Tech. report UCB/CSD 90/592, Computer Science Division, Univeirsity of California, Berkeley.

[MC91a] Manocha, D. and Canny, J. (1991) "A new approach for surface intersection", in proceedings of *First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications.* Also available as RAMP memo. 90-11/ERSC 90-23, Engineering System Research Center, University of California, Berkeley.

[MC91b] Manocha, D. and Canny, J. (1991) "The implicit representation of rational parametric surfaces", to appear in *Journal of Symbolic Computation* .

[Mi90] Milne, P. (1990) "On the solutions of a set of polynomial equations", manuscript, Department of Computer Science, University of Bath, England.

[MM82] Mayr E. and Meyer A. (1982) "The complexity of the word problem in commutative semigroups and polynomial ideals", *Advances in Mathematics,* vol. **46**, pp.

305–329.
[Mo25] Morley, F. (1925) "The eliminant of a net of curves", *American Journal of Mathematics,* vol. **47**, pp. 91–97.
[Mr87] Morgan, A.P. (1987) *Solving polynomial systems using continuation for scientific and engineering problems,* Prentice-Hall, Englewood Cliffs, New Jersey.
[Mr90] Morgan, A.P. (1990) "Polynomial continuation and its relationship to the symbolic reduction of polynomial systems", presented at the workshop on *Integration of Numeric and Symbolic Computing Methods,* Saratoga Springs, New York.
[Sa1885] Salmon, G. (1885) *Lessons introductory to the modern higher algebra,* G.E. Stechert & Co., New York.
[Wd50] van der Waerden B. L. (1950) *Modern algebra,* (third edition) F. Ungar Publishing Co., New York.
[Wh09] White, H.S. (1909) "Bezout's theory of resultants and its influence on geometry", *Bulletin of the American Mathematical Society,* pp. 325–338.
[WM90] Wampler, C. and Morgan, A. (1990) "Numerical continuation methods for solving polynomial systems arising in kinematics", *ASME Journal on Design,* vol. **112**, pp. 59–68.
[Zi90] Zippel, R. (1990) "Interpolating polynomials from their values", *Journal of Symbolic Computation,* vol. **9**, 375–403.