Appeared in IEEE Computer Graphics and Applications, March 1994

Solving Systems of Polynomial Equations

Dinesh Manocha[1]

Department of Computer Science
University of North Carolina,
Chapel Hill, NC 27599-3175

Abstract:

Current geometric and solid modeling systems use semi-algebraic sets for defining the boundaries of solid objects, curves and surfaces, geometric constraints with mating relationship in a mechanical assembly, physical contacts between objects, collision detection. It turns out that performing many of the geometric operations on the solid boundaries or interacting with geometric constraints is reduced to finding common solutions of the polynomial equations. Current algorithms in the literature based on symbolic, numeric and geometric methods suffer from robustness, accuracy or efficiency problems or are limited to a class of problems only.

In this paper we present algorithms based on multipolynomial resultants and matrix computations for solving polynomial systems. These algorithms are based on the linear algebra formulation of resultants of equations and in many cases there is an elegant relationship between the matrix structures and the geometric formulation. The resulting algorithm involves singular value decompositions, eigendecompositions, Gauss elimination etc. In the context of floating point computation their numerical accuracy is well understood. We also present techniques to make use of the structure of the matrices to improve the performance of the resulting algorithm and highlight the performance of the algorithms on different examples.

[1]Supported in part by a Junior Faculty Award and DARPA Contract #DAEA 18-90-C-0044

1 Introduction

Geometric and solid modeling deal with issues of representation and manipulation of physical objects. These fields have received much attention throughout the industrial and academic communities for more than three decades. Most of the current formulations of geometric objects (curves and surfaces) are in terms of polynomial equations and in many application like boundary computations, the problems are reduced to manipulating a polynomial systems. Polynomial equations are used for representing semi-algebraic sets. The importance of semi-algebraic sets has been established in many other applications of geometric computation besides boundary representations. For example, the geometric constraints associated with kinematic relationships in a mechanical assembly (composed of prismatic and revolute joints) define semi algebraic sets. Other applications include the representations of voronoi diagrams of set-theoretic models [1], formulation of configuration space of a robot in motion planning applications [2] etc. A fundamental problem in these geometric computations is that solving systems of polynomial equations. In particular, the problems of curve intersection, ray tracing curves and surfaces, inverse kinematics of serial or parallel mechanisms, collision detection, computing the distance from a point to a curve, finding a point on the bisector between two curves or a point equidistant from three curves and solving geometric constraint systems are reduced to finding roots of non-linear polynomial equations [3, 4, 1]. The need for solving algebraic equations also arises in surface intersection algorithms, for finding starting points on each component and locating singularities [5, 6], manipulating offset curves and surfaces [7] and geometric theorem proving [5]. Most of these problems have been extensively studied in the literature.

The currently known techniques for solving non-linear polynomial systems can be classified into symbolic, numeric and geometric methods. Symbolic methods based on resultants and Grőbner bases algorithm can be used for eliminating variables and thereby reducing the problem to finding roots of univariate polynomials. These methods have their roots in algebraic geometry. However, the current algorithms and implementations are efficient for sets of low degree polynomial systems consisting of up to three to four polynomials only. The major problem arises from the fact that computing roots of a univariate polynomial can be ill–conditioned for polynomials of degree greater than 14 or 15, as shown by Wilkinson [8]. As a result, it is difficult to implement these algebraic methods using finite precision arithmetic and that slows down the resulting algorithm. As far as the use of algebraic methods in geometric and solid modeling is concerned, the current viewpoint is that they have led to better theoretical understanding of the problems, but their practical impact is unclear [7, 9].

The numeric methods for solving polynomial equations can be classified into iterative methods and homotopy methods. Iterative techniques, like the Newton's method, are good for local analysis only and work well if we are given good initial guess to each solution. This is rather difficult for applications like intersections or geometric constraint systems. Homotopy methods based on continuation techniques have a good theoretical background and proceed by following paths in the complex space. In theory, each path converges to a geometrically isolated solution. They have been implemented and applied to a variety of applications [10]. In practice the current implementations have many problems. The different paths being followed may not be geometrically isolated and thereby causing problems with the robustness of the approach. Moreover, continuation methods are considered to be computationally very demanding and at the moment restricted to solving dense polynomial systems only. Recently, methods based on interval arithmetic have received a great deal of attention in computer graphics and geometric modeling. The resulting algorithms are robust, though their convergence can be relatively slow.

For some particular applications, algorithms have been developed using the geometric formulation of the problem. This includes subdivision based algorithms for curves and surface intersection, ray tracing. In the general case subdivision algorithms have limited applications and their convergence is slow. Their convergence has been improved by Bézier clipping [11]. However, for low degree curve intersections algebraic methods have been found to be the fastest in practice. Similarly algorithms based on the geometric properties of mechanisms have been developed for a class of kinematics problems, constraint systems and motion planning problems.

In this paper we present an algorithm for finding roots of polynomial equations and demonstrate its performance on a number of applications. The algorithm uses resultants to eliminate variables from a system of polynomial equations. The resultant formulations linearize a non-linear polynomial system and the resultant, therefore, can be expressed in terms of matrices and determinants. In particular, we show that the resultant of a system of polynomial equations corresponds to the determinant of a *matrix polynomial* and the problem of finding its roots can be reduced to an eigenvalue problem (or a generalized eigenvalue of a matrix pencil). There is an elegant relationship between the kernels of these matrix polynomials and the variables being eliminated, which is being used for computing the rest of the variables and birational maps. We consider algorithms for sparse as well as dense polynomial systems and illustrate them on the problems of curve and surface intersections, finding distance from a point to the curve, birational maps and geometric constraint systems. Later on we make use of the sparsity of the matrices to improve the performance of the algorithm. The resulting algorithms are robust in nature and their numerical accuracy is well understood. For most of the linear algebra problems, *backward stable* algorithms are known. This is in contrast with finding roots of high degree univariate polynomials. Furthermore, efficient implementations of backward stable routines are available as part of LAPACK [12] and we used them in our applications.

The rest of the paper is organized in the following manner. In Section 2 we review the results from elimination theory. We consider sparse as well as dense polynomial systems. In Section 3, we briefly review some of the techniques from linear algebra and matrix computations used in the paper. In Section 4 we show how the resultants of polynomial systems can be expressed in terms of matrix polynomials. Furthermore, there is a direct relationship between the variables being eliminated and the kernels of the matrix polynomials. In Section 5 we illustrate the algorithm on intersection of curves and surfaces, finding distance of a point to a curve or a surface, representation of birational maps and geometric constraint systems. In Section 6 we discuss the performance and limitations of the current algorithm and highlight future directions for performance improvement and finally in Section 7 we conclude the paper.

2 Background

A semi-algebraic set is obtained by a finite number of Boolean set operations (union, intersection, difference) applied to half spaces defined by algebraic inequalities. These sets are used to define solids and their boundaries consist of zero, one and two dimensional algebraic sets (in 3-space). Algebraic sets are defined as common zeros of a system of polynomial equations. Common examples of boundary sets include parametric curves and surfaces defined by Bézier and B-spline formulations. For many operations like intersections, offsets and blends on these boundary sets, the resulting algorithms involve computing common roots of polynomial equations. Elimination theory, a branch of classical algebraic geometry, deals with conditions for common solutions of a system of polynomial equations. Its 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 refer to this resultant as the *multipolynomial resultant* and use it in the algorithm presented in the paper.

2.1 Resultants

Resultants are used for eliminating a set of variables from given equations and the geometric, symbolic and numeric problems can be cast in that manner. Many formulations for computing the resultant of a system of polynomial equations are known in the literature. The classical literature in algebraic geometry deals with resultant of dense polynomial systems [13, 14]. The resultant of sparse polynomial systems has received a considerable amount of attention in the recent literature and many formulations have appeared [15, 16] based on the BKK bound relating the number of solutions of a sparse system to mixed volumes of Newton polytopes. All these formulations of resultants express them in terms of matrices and determinants. For many special cases, corresponding to $n = 2, 3, 4, 5, 6$, where n is the number of equations, efficient formulations of resultants expressed as the determinant of a matrix, are given in [14]. The most general formulation of resultant for dense polynomial systems expresses it as a ratio of two determinants [13]. For sparse polynomials a single determinant formulation is known for multigraded systems [15] and a formulation equivalent to Macaulay's formulation has been highlighted in [16]. The ongoing activity in sparse elimination methods is important as many polynomial systems resulting from applications in geometric constraint systems, intersection, offsets and blends are sparse [7, 17].

Resultants have gained a lot of importance in geometric and solid modeling literature. Following Sederberg's thesis [3] on implicitization of curves and surfaces, resultants have been applied to motion planning [2] and many other geometric problems, as surveyed in [18]. Most of the earlier practical applications of resultants were limited to low degree curve and surface intersections [7]. The idea of combining resultant formulations with matrix computations has been proposed with Macaulay's formulation in [19]. In particular, [19] describe a general formulation of the resultant of dense polynomial system in terms of matrices and show that the solutions can be computed by reducing it to a linear eigenvalue problem. It turns out their resultant algorithm corresponds to the Macaulay's formulation [13] and [19] do not take into account that the resultant is expressed as a ratio of two determinants. As a result, their approach may not work whenever the lower determinant vanishes. Furthermore, for many cases of polynomial systems consisting of up to $5-6$ equation, efficient formulations of resultant given by Bezout, Dixon, Morley and Coble etc. result in non-linear matrix polynomials. The algorithm presented in this paper is not based on any particular formulation of the resultant. Rather it uses the fact that resultants can be expressed in terms of determinants of non–linear matrix polynomials. This approach has been specialized to particular problems of curve and surface intersections. In [20] the problem of curve intersection is analyzed using resultants and algorithms are presented based on Bezout resultant of two polynomials and matrix computations. Higher order intersections corresponding to tangential intersections or singular points are considered as well. For intersection of triangular or tensor product surfaces, [6] use the fact that the implicit representation of such surfaces corresponds to the determinant of a matrix and proposed a representation and evaluation of the intersection curve in terms of matrix computations.

3 Matrix Computations

In this section we review techniques from linear algebra and numerical analysis used in our algorithm. We also discuss the numerical accuracy of the problems in terms of their condition numbers and the algorithms being used.

3.1 QR Factorization

The QR factorization of an $m \times n$ matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q}$ is an $m \times m$ orthogonal matrix and $\mathbf{R}$ is an $m \times n$ upper triangular matrix. More details on its computations are given in [22].

3.2 Singular Value Decomposition

The singular value decomposition (SVD) is a powerful tool which gives us accurate information about matrix rank in the presence of round off errors. Given $\mathbf{A}$, a $m \times n$ real matrix then there exist orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U}$ is a $m \times n$ orthogonal matrix, $\mathbf{V}$ is $n \times n$ orthogonal matrix and $\mathbf{\Sigma}$ is a $n \times n$ diagonal matrix of the form

$$\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n).$$

Moreover, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$. The σ_i's are called the *singular values*,

3.3 Eigenvalues and Eigenvectors

Given a $n \times n$ matrix $\mathbf{A}$, its eigenvalues and eigenvectors are the solutions to the equation

$$\mathbf{A}\mathbf{x} = s\mathbf{x},$$

where s is the eigenvalue and $\mathbf{x} \neq \mathbf{0}$ is the eigenvector. The eigenvalues of a matrix are the roots of its characteristic polynomial, corresponding to determinant$(\mathbf{A} - s\mathbf{I})$. As a result, the eigenvalues of a diagonal matrix, upper triangular matrix or a lower triangular matrix correspond to the elements on its diagonal. Efficient algorithms for computing eigenvalues and eigenvectors are well known, [22], and their implementations are available as part LAPACK [12].

3.4 Power Iterations

The largest or the smallest eigenvalue of a matrix (and the corresponding eigenvector) can be computed using the *Power method* [22]. Power method involves multiplication of a matrix by a vector and after a few steps it converges to the largest eigenvalue of a method. Given a matrix, $\mathbf{A}$, the technique starts with a vector $\mathbf{q}_0$ and performs computation of the form

$$\mathbf{z}_i = \mathbf{A}\mathbf{q}_{i-1},$$

$$\mathbf{q}_i = \mathbf{z}_i / \parallel \mathbf{z}_i \parallel,$$

$$\lambda_i = \mathbf{q}_i^T \mathbf{A} \mathbf{q}_i.$$

After a few iterations, λ_k corresponds to the eigenvalue of maximum magnitude and $\mathbf{q}_k$ is the corresponding eigenvector.

3.5 Sparse Matrix Computations

The general formulation of resultants corresponding to Macaulay formulation results in sparse matrices [13]. In such cases we want to make use of the sparsity of the matrix in computing its eigendecomposition. The order of Macaulay matrix is a function of the number of polynomials and the degrees of the polynomial. The sparsity of the matrix increases with the degrees of the polynomials or the number of equations.

Algorithms for sparse matrix computations are based on matrix vector multiplications as highlighted in the Power iterations. For our applications, we use the algorithm highlighted in [23] for computing the invariant subspaces and thereby the eigendecomposition of a sparse matrix.

3.6 Generalized Eigenvalue Problem

Given $n \times n$ matrices, $\mathbf{A}$ and $\mathbf{B}$, the generalized eigenvalue problem corresponds to solving

$$\mathbf{A}\mathbf{x} = s\mathbf{B}\mathbf{x}.$$

We represent this problem as eigenvalues of $\mathbf{A} - s\mathbf{B}$. The vectors $\mathbf{x} \neq \mathbf{0}$ correspond to the eigenvectors of this equati on. If $\mathbf{B}$ is non–singular and its condition number (defined in the next section) is low, the problem can be reduced to an eigenvalue problem by multiplying both sides of the equation by $\mathbf{B}^{-1}$ and thereby obtaining:

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = s\mathbf{x}.$$

However, $\mathbf{B}$ may have a high condition number and such a reduction can cause numerical problems.

4 Algorithm for Solving Equations

In this section, we describe the algorithm for solving a system of non-linear equations (assuming that they have finite number of common solutions). We initially show how the resultants are being used to linearize the problem in terms of matrices and determinants. In particular, we obtain matrix polynomials and the problem of roots computation is being reduced to finding eigenvalues of a matrix polynomial and the corresponding vectors in its kernel. This algorithm is illustrated on many applications in the next section.

Given a system of n equations in n unknowns,

$$\begin{array}{rcl} F_1(x_1, x_2, \ldots, x_n) & = & 0 \\ F_2(x_1, x_2, \ldots, x_n) & = & 0 \\ & \vdots & \\ F_n(x_1, x_2, \ldots, x_n) & = & 0. \end{array}$$

Let the degrees of these equations be $d_1, d_2, \ldots, d_n$, respectively. The resultant, $R(x_1)$, is obtained by eliminating the variables $x_2, x_3, \ldots, x_n$ from these equations. The resultant is a polynomial in x_1 and its roots correspond to the x_1 coordinate of each solution of the given multivariate system. The degree of the resultant is equal the total number of non-trivial solutions of the given system of equations. The degree is is bounded by the BKK bound for sparse system and Bezout bound for dense polynomial systems. Different formulations of resultant express it as determinant of a matrix or as ratio of two determinants. In either case, the entries of the resulting matrices are polynomial functions of x_1.

In case, a single matrix formulation is not possible for the given system, we use the u-resultant formulation to solve the given system of equations [24]. In particular, we add a polynomial

$$F_{n+1}(x_1, x_2, \ldots, x_n) = u_0 + u_1 x_1 + \ldots + u_n x_n$$

to the given system of equations. The u_i's are symbolic variables. The resultant is obtained by eliminating the variables $x_1, \ldots, x_n$ from the $n+1$ equations and is a polynomial in $u_0, u_1, \ldots, u_n$. It is known as the u-resultant of original system of polynomial equations [24]. Moreover, the u-resultant is expressed as a ratio of two determinants, $Det(\mathbf{M})/Det(\mathbf{D})$. However the entries of D are independent of the u_i's. This is a property of Macaulay's formulation and the u-resultants [24]. As a result, if the matrix $\mathbf{D}$ is non-singular, the resultant of the $F_1, F_2, \ldots, F_{n+1}$ corresponds exactly to the determinant of $\mathbf{M}$. In case, $\mathbf{D}$ is singular, we replace $\mathbf{M}$ by its largest non-singular minor as shown in [17].

Given $\mathbf{M}$, whose entries are polynomials in the u_i's, the u-resultant corresponding to its determinant can be factored into linear factors of the form [24]:

$$Det(\mathbf{M}) = \prod_{i=1}^{k} (\alpha_{i0} u_0 + \alpha_{i1} u_1 + \ldots + \alpha_{in} u_n)$$

where k is the total number of non-trivial solution and $(\alpha_{i0}, \alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in})$ are the projective coordinates of a solution of the given system of equations. Let us choose a specialization of the variables:

$$u_0 = x_1, \quad u_1 = -1, \quad u_2 = 0, \quad u_3 = 0, \quad \ldots, \quad u_n = 0.$$

The determinant of $\mathbf{M}$ obtained after specialization is a polynomial in x_1 and its roots correspond exactly to the x_1 coordinate of each solution of the given multivariate system. Thus, the determinant corresponds exactly to the resultant of $F_1, F_2, \ldots, F_n$, $R(x_1)$, obtained after eliminating $x_2, x_3, \ldots, x_n$. As a result, given any system of n polynomial equations whose coefficients are numeric constants, we can eliminate $n-1$ variables and express the resultant as determinant of a matrix $\mathbf{M}(x_1)$.

Multipolynomial resultants linearize a non-linear polynomial system. In other words, they take a system of non-linear polynomial equations, say $F_1, F_2, \ldots, F_n$, and reduce it to a linear system of the form

$$\mathbf{M}(x_1)(1\ x_2\ \ldots\ x_n\ \ldots\ x_2^d\ x_3^d\ \ldots\ \ldots\ x_n^d)^T = (0\ 0\ \ldots\ 0)^T. \qquad (1)$$

$\mathbf{M}(x_1)$ is a square matrix and its entries are polynomials in x_1. The entries of the vector consist of power products of $x_1, x_2, \ldots, x_n$ (the actual arrangement of the power products of these variables is a function of the degrees of the polynomial and the formulation of resultant being used). This linearization has the property that for any given solution $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, of the given system, $\mathbf{M}(\alpha_1)$ is a singular matrix and the vector in its kernel is obtained by substituting $x_1 = \alpha_1,\ x_2 = \alpha_2,\ \ldots,\ x_n = \alpha_n$ in the vector consisting of power products highlighted in (1). We use this property along with those of matrix polynomials to compute the roots of the polynomial equations.

4.1 Matrix Polynomials

The matrix $\mathbf{M}(x_1)$ highlighted in the previous section can be expressed as a matrix polynomial:

$$\mathbf{M}(x_1) = \mathbf{M}_0 + \mathbf{M}_1 x_1 + \mathbf{M}_2 x_2 + \ldots + \mathbf{M}_l x_1^l, \qquad (2)$$

where $\mathbf{M}_i$ are numeric matrices and l is the maximum degree of x_1 in any term of $\mathbf{M}(x_1)$. All $\mathbf{M}_i$ have the same order, say $m \times m$. The determinant of $\mathbf{M}(x_1)$ corresponds exactly to the resultant of the given equations and we are interested in its roots. Furthermore, for a given root α_1, the kernel of $\mathbf{M}(\alpha_1)$ is used to compute rest of the coordinates, $\alpha_2, \alpha_3, \ldots, \alpha_n$.

Let us assume that the leading matrix, $\mathbf{M}_l$ is non-singular and well-conditioned. As a result computation of $\mathbf{M}_l^{-1}$ does not introduce severe numerical errors. Let

$$\overline{\mathbf{M}}(x_1) = \mathbf{M}_l^{-1}\mathbf{M}(x_1), \text{ and } \overline{\mathbf{M}}_i = \mathbf{M}_l^{-1}\mathbf{M}_i, \quad 0 \le i < l.$$

$\overline{\mathbf{M}}(x_1)$ is a monic matrix polynomial. Its determinant has the same roots as does the determinant of $\mathbf{M}(x_1)$. Let $x_1 = \alpha_1$ be a root of the equation, Determinant($\overline{\mathbf{M}}(x_1)) = 0$. As a result $\overline{\mathbf{M}}(\alpha_1)$ is a singular matrix and there is at least one non trivial vector in its kernel. Let us denote that $m \times 1$ vector as $\mathbf{v}$. That is

$$\overline{\mathbf{M}}(\alpha_1)\mathbf{v} = \mathbf{0}, \tag{3}$$

where $\mathbf{0}$ is a $m \times 1$ null vector. The roots of the determinant of $\mathbf{M}(x_1)$ correspond to the eigenvalues of $\mathbf{C}$ highlighted in the following theorem [17]:

Theorem 4.1 *Given the matrix polynomial, $\overline{\mathbf{M}}(x_1)$ the roots of the polynomial corresponding to its determinant are the eigenvalues of the matrix*

$$\mathbf{C} = \begin{pmatrix} \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_m & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{I}_m \\ -\overline{\mathbf{M}}_0 & -\overline{\mathbf{M}}_1 & -\overline{\mathbf{M}}_2 & \dots & -\overline{\mathbf{M}}_{l-1} \end{pmatrix}, \tag{4}$$

where $\mathbf{0}$ and $\mathbf{I}_m$ are $m \times m$ null and identity matrices, respectively. Furthermore, the eigenvector of $\mathbf{C}$ corresponding to the eigenvalue $x_1 = \alpha_1$ has the form:

$$[\mathbf{v}\ \alpha_1\mathbf{v}\ \alpha_1^2\mathbf{v}\ \dots\ \alpha_1^{l-1}\mathbf{v}]^T,$$

where $\mathbf{v}$ is the vector in the kernel of $\overline{\mathbf{M}}(\alpha_1)$ as highlighted in (3).

Many a times the leading matrix $\mathbf{M}_l$ is singular or close to being singular (due to high condition number). Some techniques based on linear transformations are highlighted in [17], such that the problem of finding roots of determinant of matrix polynomial can be reduced to an eigenvalue problem. However, there are cases where they do not work. For example, when the matrices have singular pencils. In such cases, we reduce the intersection problem to a generalized eigenvalue problem using the following theorem [17]:

Theorem 4.2 *Given the matrix polynomial, $\mathbf{M}(x_1)$ the roots of the polynomial corresponding to its determinant are the eigenvalues of the generalized system $\mathbf{C}_1 x_1 - \mathbf{C}_2$, where*

$$\mathbf{C}_1 = \begin{pmatrix} \mathbf{I}_m & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{M}_l \end{pmatrix}$$

$$\mathbf{C}_2 = \begin{pmatrix} \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_m & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{I}_m \\ -\mathbf{M}_0 & -\mathbf{M}_1 & -\mathbf{M}_2 & \dots & -\mathbf{M}_{l-1} \end{pmatrix}, \tag{5}$$

where $\mathbf{0}$ *and* $\mathbf{I}_m$ *are* $m \times m$ *null and identity matrices, respectively.*

The roots of the determinant of $\mathbf{M}(x_1)$ correspond to the eigenvalues of $\mathbf{C}$ or $\mathbf{C}_1 x_1 - \mathbf{C}_2$. In many applications we are only interested in the real solutions or solutions lying in a particular domain. The QR or QZ algorithm for eigenvalue computation returns all the eigenvalues of a given matrix and it is difficult to restrict them to finding eigenvalues in a particular domain [22].

Let us assume that α_1 is a simple eigenvalue of $\mathbf{C}$. In the rest of the paper, we carry out the analysis on the eigenvalues of $\mathbf{C}$ and the resulting algorithm is similar for the eigenvalues of the pencil $\mathbf{C}_1 x_1 - \mathbf{C}_2$. Since α_1 is a simple eigenvalue, the kernel of $\mathbf{C} - \alpha_1 \mathbf{I}$ has dimension one represented as

$$\mathbf{V} = [\mathbf{v}\ \alpha_1\mathbf{v}\ \alpha_1^2\mathbf{v}\ \dots\ \alpha_1^{l-1}\mathbf{v}]^T.$$

Furthermore, we know that $\mathbf{v} = [v_1\ v_2\ \dots\ v_m]^T$ corresponds to the vector in the kernel of $\mathbf{M}(\alpha_1)$. Given $\mathbf{v}$, we use the relationship highlighted in (1) to compute the $x_2, \dots, x_n$ coordinates:

$$(1\ x_2\ \dots\ x_n\ \dots\ x_2^d\ x_3^d\ \dots\ \dots\ x_n^d)^T = \beta(v_1\ v_2\ \dots v_m)$$

For example, $\alpha_2 = \frac{v_2}{v_1}$.

In many cases α_1 may correspond to an eigenvalue of multiplicity greater than one of $\mathbf{C}$. There are two possibilities:

- $(\alpha_1, \alpha_2, \dots, \alpha_n)$ is a solution of multiplicity greater than one of the given system of equations. In this case, the algebraic of multiplicity of α_1 is greater than one but the geometric multiplicity corresponding to the dimension of the kernel of $\mathbf{C} - \alpha_1 \mathbf{I}$ is still one.

- There may be two solutions of the given equations of the form $(\alpha_1, \alpha_2, \dots, \alpha_n)$ and $(\alpha_1, \alpha_2', \dots, \alpha_n')$. As a result the kernel of $\mathbf{C} - \alpha_1 \mathbf{I}$ has dimension greater than one.

The problem of computing higher multiplicity roots can be numerically ill-conditioned. However, in many cases it is possible to identify higher multiplicity eigenvalues of a matrix by identifying clusters of eigenvalues and using the knowledge of the condition number of the clusters. More details of its application to finding solutions of polynomial equations are given in [17].

Given a higher multiplicity eigenvalue, α_1, we compute its geometric multiplicity by computing the SVD of $\mathbf{C} - \alpha_1 \mathbf{I}$. The geometric multiplicity corresponds to the number of singular values equal to zero. In case, the geometric multiplicity is one, the relationship highlighted in (1) is used to compute $\alpha_2, \alpha_3, \ldots, \alpha_n$ for each α_1. Otherwise there are two or more vectors in the kernel of $\mathbf{M}(\alpha_1)$. The vectors computed using linear algebra routines may correspond to any two vectors in the vector space corresponding to the kernel. As a result, it is difficult to compute α_2, α_3, $\ldots, \alpha_n$ from them. To solve the problem we substitute $x_1 = \alpha_1$ and solve the $n-1$ equations:

$$\begin{aligned} F_1(\alpha_1, x_2, \ldots, x_n) &= 0 \\ F_2(\alpha_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ F_{n-1}(\alpha_1, x_2, \ldots, x_n) &= 0, \end{aligned}$$

for $x_2, x_3 \ldots, x_n$. The solutions obtained are verified by substituting into $F_n(\alpha_1, x_2, \ldots, x_n) = 0$.

We have implemented the algorithm using linear algebra libraries. The three major parts of the algorithm are:

1. Use a suitable resultant formulation to linearize the problem in terms of matrix polynomials. The entries of the Macaulay matrix are actually the coefficients of the polynomial equations. The resultant formulations of Bezout, Dixon corresponding to two or three equations result in matrix entries being polynomial functions of the coefficients.

2. Reduce the problem to an eigenvalue problem. This involves estimating the condition number of the leading matrix of the matrix polynomial. In some cases a linear rational transformation is involved on the matrices to improve the conditioning of the leading matrix. Finally, we reduce the problem to an eigenvalue or a generalized eigenvalue problem.

3. Compute the eigendecomposition of the given matrix and recover the common roots from the eigenvalues and the eigenvectors. In a few instances, this may involve identifying higher order eigenvalues using knowledge of clusters, using the SVD to know the geometric multiplicity of the eigenvalue and possibly solving a system of $n-1$ equations in $n-1$ unknowns.

All the three parts mentioned above are relatively simple to implement, given the linear algebra routines. A major feature of the algorithm is that at each stage the numerical accuracy of the operations involved is well understood. As a result, we are able to come up with tight bounds on the accuracy of the resulting solution. In fact, the higher multiplicity eigenvalues are determined from their condition numbers. For most cases, we are able to accurately compute the roots using the 64 bit IEEE floating point arithmetic available on most workstations.

5 Applications

In this section we highlight the application of the equation solving algorithm to some problems in boundary computations and geometric constraint systems. Initially, we consider an example of intersection of three surfaces from [10]. In particular, Morgan uses continuation methods to solve this problems and argues that approaches based on symbolic reduction lead to severe numeric problems. Secondly, we consider the problem of finding a distance from a point to a curve or a surface. It is well known that these problems can be reduced to solving non-linear algebraic equations and they are frequently encountered in the computation of offset curves and surfaces and voronoi surfaces [7, 1]. The matrix relationship expressed in (1) is also used for computing the birational maps. These birational maps are expressed in terms of ratio of determinants as opposed to algebraic expression. They are useful for the *inversion problem* and expressing the rational relationships between the variables being eliminated and the coefficients of the given polynomial system. Finally, we discuss the application of this algorithm to geometric constraint systems.

5.1 Intersection of Three Surfaces

Given three algebraic surfaces, $F_1(x,y,z) = 0, \quad F_2(x,y,z) = 0, \quad F_3(x,y,z) = 0$, we are interested in their common points of intersection. This problem arises frequently in the boundary computations [5]. It turns out that this problem corresponds to solving 3 equations in 3 unknowns. In case, we are given parametric surfaces, we can either implicitize them and reduce it to solving for 3 equations or deal with the parametric variables and reduce the problem to solving 6 equations in 6 unknowns.

The total number of solutions in the complex space is bounded by the product of the three degrees. Resultants and Gröbner bases have been used to eliminate two variables from these three equation. The resulting problem correspond to finding roots of a univariate polynomial, which can be ill-conditioned on low degree polynomials. We illustrate it on an example from [10] and also analyze the accuracy of the algorithm presented in this paper.

Consider the intersection of a sphere, a cylinder and a plane described by the following equations:

$$\begin{aligned} 1.6e{-}3\, x_1^2 + 1.6e{-}3\, x_2^2 - 1 &= 0 \\ 5.3e{-}4\, x_1^2 + 5.3e{-}4\, x_2^2 + 5.3e{-}4\, x_3^2 + 2.7e{-}2\, x_1 - 1 &= 0 \end{aligned}$$

$$-1.4e{-4}\,x_1 + 1.0e{-4}\,x_2 + x_3 - 3.4e{-3} \;=\; 0.$$

According to [10], these equations have two real solutions of norm about 25 and a complex conjugate pair of order 10^9. The two real solutions have a physical meaning. After eliminating x_2 and x_3 from these equations, the eliminant is [10]:

$$\begin{gathered} 6.38281970398352\,x_1^4 \;-\; 7.12554854545301e9\,x_1^3 + \\ 1.89062308408416e19\,x_1^2 \;+ 9.36558635415069e20\,x_1 \\ -\;1.15985845720186e22. \end{gathered}$$

where the coefficients have been rounded to 15 digits. In the original polynomial system there is a range of 4 orders of magnitude in the coefficients and in the eliminant the range is 22. As a result, it is very difficult to accurately compute the roots using the IEEE 64 bit arithmetic, which allows 16 digits of precision. These kind of problems are commonly faced in algebraic algorithms.

We analyze the accuracy and robustness of our algorithm on this problem. We use the Macaulay formulation of the resultant and it linearizes the problem into

$$\mathbf{M}(x_1)\mathbf{v} = \mathbf{0}. \tag{6}$$

The matrix $\mathbf{M}(x_1)$ corresponding to the resultant formulation is:

$$\begin{pmatrix}
1.6e{-3} & 0 & 0 & 0 & 0 & 0 & 0 & -1 + 1.6e{-3}\,x_1^2, & 0 & 0 \\
0 & 1.6e{-3} & 0 & 0 & 0 & 0 & 0 & 0 & -1 + 1.6e{-3}\,x_1^2 & 0 \\
0 & 0 & 1.6e{-3} & 0 & 0 & 0 & 0 & 0 & 0 & -1 + 1.6e{-3}\,x_1^2 \\
5.3e{-4} & 0 & 0 & 5.3e{-4} & 0 & 0 & 0 & 5.3e{-4}\,x_1^2 + 0.027\,x_1 - 1 & 0 & 0 \\
0 & 5.3e{-4} & 0 & 0 & 5.3e{-4} & 0 & 0 & 0 & 5.3e{-4}\,x_1^2 + 0.027\,x_1 - 1 & 0 \\
0 & 0 & 5.3e{-4} & 0 & 0 & 5.3e{-4} & 0 & 0 & 0 & 5.3e{-4}\,x_1^2 + 0.027\,x_1 - 1 \\
0 & 1.0e{-4} & 0 & 1 & 0 & 0 & -1.4e{-4}\,x_1 - 3.4e{-3} & 0 & 0 & 0 \\
0 & 0 & 1.0e{-4} & 0 & 0 & 0 & 1 & -1.4e{-4}\,x_1 - 3.4e{-3} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1.0e{-4} & 0 & -1.4e{-4}\,x_1 - 3.4e{-3} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0e{-4} & 1 & -1.4e{-4}\,x_1 - 3.4e{-3}
\end{pmatrix}$$

and the right hand side vector is

$$\mathbf{v} = (x_2^3\;x_2^2x_3\;x_2^2\;x_2x_3^2\;x_3^3\;x_3^2\;x_2x_3\;x_2\;x_3\;1)^T.$$

Although the Macaulay resultant expresses the resultant as a ratio of two determinants, the lower matrix is non-singular and consist of numerical entries only.

It follows from Bezout's theorem that the system has 4 solutions in the complex space. In fact, the determinant of the matrix, say $\mathbf{M}(x_1)$, highlighted above corresponds to the eliminant (up to a constant). We treat it as a matrix polynomial and reduce the problem to an eigenvalue problem.

In this case, $\mathbf{M}(x_1)$ is a 10×10 matrix polynomial of degree 2 in x_1. Let us express it as:

$$\mathbf{M}(x_1) = \mathbf{M}_2 x_1^2 + \mathbf{M}_1 x_1 + \mathbf{M}_0.$$

The fact that the determinant of $\mathbf{M}(x_1)$ has degree 4 implies that $\mathbf{M}_2$ is singular. As a result, we cannot reduce the problem to an eigenvalue problem using Theorem 4.1. It turns out that the condition number of $\mathbf{M}_0$ is $4.0911e03$. As a result, we perform a linear transformation $y = 1/x_1$ and we reduce the problem to finding eigenvalues of the matrix

$$\mathbf{C} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}_0^{-1}\mathbf{M}_2 & -\mathbf{M}_0^{-1}\mathbf{M}_1 \end{pmatrix}.$$

It follows that $y = 0$ is an eigenvalue of multiplicity 6 of $\mathbf{C}$. We use this information in choosing the appropriate shifts in the double shift QR algorithm for eigendecomposition, as described in [22]. This knowledge of some of the eigenvalues speeds up its convergence. The 4 nonzero eigenvalues are

$$0.04037383,\ 0.04037383,\ -1.35035361e-10+5.773741e-10\,i,\ -1.35035361e-10-5.773741e-10\,i,$$

where $i = \sqrt{-1}$. Substituting $x_1 = 1/y$ and computing x_2 and x_3 from the eigenvectors of $\mathbf{C}$ results in two real solutions to the original equations:

$$(x_1, x_2, x_3) = (4.76851749893, -3.39419223, 0.00720701519),$$

$$(x_1, x_2, x_3) = (24.76851768, 3.3941908959, 0.006528173).$$

The solutions computed are accurate up to 8 digits of accuracy. Further accuracy can be achieved by a few iterations of Newton's method on the original equations and using the solutions from eigendecomposition procedure as the start points for Newton's iteration. We have used this algorithm on the intersection of three surfaces with more than 100 intersections and are able to compute the solutions with good accuracy.

5.2 Distance from a Point to a Curve or Surface

The problem of computing the distance of a point to a curve or a surface comes up repeatedly in solid modeling computations. Some examples include computation of the *medial axis transform*, voronoi surfaces, offset curve and surfaces etc. [7, 1]. It is well known in literature that this problem can either be posed as an optimization problem (minimizing the distance function) or reduce it to solving algebraic equations. Let (X, Y) be a point and $F(x, y) = 0$ be the curve. Let (x_p, y_p) be the point on the curve closest to (X, Y). Then there are two equations:

$$F(x_p, y_p) = 0$$

$$(Y - y_p)F_x(x_p, y_p) - (X - x_p)F_y(x_p, y_p) = 0.$$

It turns out that these equations have more than one real solution and therefore, any algorithm based on local methods may converge to a wrong solution [1]. Algorithms based on constrained optimization may converge to a local minima of the function. Thus, none of the previous technique is able to solve this problem in a reasonable manner.

Using the algebraic formulation the problem reduces to solving for x_p, y_p in these two equations. For a curve of degree n there are n^2 solutions to these equations. We used the Bezout formulation of the resultant of two equations and solved the problem using the eigenvalue formulation. In particular, we eliminate x_p from the given equations and the Bezout formulation results in a $n \times n$ matrix, whose entries are a polynomial of degree n in y_p. This is eventually reduced to an eigenvalue problem of a matrix of order n^2. The resulting algorithm works well in practice. We know from the equations that the roots of the given system contain a point on the curve closest to (X, Y). This point is identified by computing the distances between the roots and (X, Y) and finding the root corresponding to the minimum distance. Similarly the problem of finding the distance from a point to a surface is reduced to solving three polynomials in three unknowns.

Many other problems like ray-tracing parametric curves and surfaces, finding singular points on algebraic curves and surface can be reduced to solving two polynomial equations in two or three unknowns. This algorithm has been successfully applied to these problems [17].

5.3 Birational Maps

Birational maps play a fundamental role in algebraic geometry. They have also gained importance in solid modeling for their use in many applications. Many problems related to boundary computation are easily solved using birational maps. The most common example is that of *inversion problem* for rational curves and surfaces. Given a rational parametric surface, expressed in projective coordinates as:

$$\mathbf{F}(s,t,u) = (x,y,z,w) = (X(s,t,u), Y(s,t,u), Z(s,t,u), W(s,t,u)),$$

where $X(s,t,u), Y(s,t,u), Z(s,t,u)$ and $W(s,t,u)$ are homogeneous polynomials of degree n. Common examples of this formulation are the triangular and tensor product Bézier patches used in geometric and solid modeling. A rational surface defines a map from the s,t,u projective plane to the projective space defined by x,y,z,w. In applications involving parametric surfaces an important problem is that of computing the s,t,u coordinates, given a point on the surface, (x_0, y_0, z_0, w_0). This is the inversion problem.

It turns out that the exact relationship between the points on the surface, $\mathbf{F}(s,t,u)$, and the parameters s,t,u can be expressed as a rational map as well:

$$\mathbf{F}^{-1}(x,y,z,w) = (s,t,u) = (S(x,y,z,w), T(x,y,z,w), U(x,y,z,w)),$$

where each of $S(x,y,z,w)$, $T(x,y,z,w)$ and $U(x,y,z,w)$ is a homogeneous polynomial. As a result, we see that $\mathbf{F}$ and $\mathbf{F}^{-1}$ define rational maps between the s,t,u and x,y,z,w space.

Algorithms to compute the birational maps are known in the literature. They are based on Grőbner bases [5] or multipolynomial resultants [18]. However, direct applications of these algorithms suffer from accuracy and efficiency problems as highlighted in [7]. We make use of the fact that multipolynomial resultants linearize a non-linear problem by eliminating some variables as shown in (1). In particular, we express the birational maps in terms of ratio of determinants and use matrix computations for their computation. At the moment, the algorithm is limited to systems of polynomial equations, such that their resultant can be expressed as a single determinant. Such formulations are known for $2,3$ or 4 equations and most applications of curve and surface modeling involving birational maps fall in this category.

Lets consider the resultant of the n equations, $F_1, F_2, \ldots, F_n$ expressed as a single matrix in (1). The entries of the matrix are functions of the coefficients of the given polynomial equations and x_1. To compute, $x_2, \ldots, x_n$ we make use of the kernel of $\mathbf{M}(x_1)$. In particular, we represent $\mathbf{M}(x_1)$ as

$$\mathbf{M}(x_1) = \begin{pmatrix} M_{1,1}(x_1) & M_{1,2}(x_1) & \ldots & M_{1,m}(x_1) \\ M_{2,1}(x_1) & M_{2,2}(x_1) & \ldots & M_{2,m}(x_1) \\ \vdots & \vdots & \ldots & \vdots \\ M_{m,1}(x_1) & M_{m,2}(x_1) & \ldots & M_{m,m}(x_1) \end{pmatrix}.$$

Let us consider the points which make $\mathbf{M}(x_1)$ singular. In this case, x_1 correspond to the roots of the determinant of $\mathbf{M}(x_1)$. For a generic choice of the point the kernel of $\mathbf{M}(x_1)$ has dimension one (this has to be the case for birational maps to be defined). Lets treat each power product in the vector highlighted in (1) as a separate variable and denote the resulting vector as

$$\mathbf{v} = [1 \ v_2' \ v_3' \ \ldots \ v_m']^T.$$

It follows from this notation that $x_i = v_i'$, for $i \leq n$. Taking the n equations denoted by $\mathbf{M}(x_1)\mathbf{v}$ and applying the Cramer's rule it follows that

$$\mathbf{v_i'} = \frac{\begin{vmatrix} M_{1,2}(x_1) & M_{1,3}(x_1) & \dots & M_{1,i-1}(x_1) & -M_{1,1}(x_1) & M_{1,i+1}(x_1) & \dots & M_{1,m}(x_1) \\ M_{2,2}(x_1) & M_{2,3}(x_1) & \dots & M_{2,i-1}(x_1) & -M_{2,1}(x_1) & M_{2,i+1}(x_1) & \dots & M_{2,m}(x_1) \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ M_{m-1,2}(x_1) & M_{m-1,3}(x_1) & \dots & M_{m-1,i-1}(x_1) & -M_{m-1,1}(x_1) & M_{m-1,i+1}(x_1) & \dots & M_{m-1,m}(x_1) \end{vmatrix}}{\begin{vmatrix} M_{1,2}(x_1) & M_{1,3}(x_1) & \dots & M_{1,i-1}(x_1) & -M_{1,i}(x_1) & M_{1,i+1}(x_1) & \dots & M_{1,m}(x_1) \\ M_{2,2}(x_1) & M_{2,3}(x_1) & \dots & M_{2,i-1}(x_1) & -M_{2,i}(x_1) & M_{2,i+1}(x_1) & \dots & M_{2,m}(x_1) \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ M_{m-1,2}(x_1) & M_{m-1,3}(x_1) & \dots & M_{m-1,i-1}(x_1) & -M_{m-1,i}(x_1) & M_{m-1,i+1}(x_1) & \dots & M_{m-1,m}(x_1) \end{vmatrix}}.$$

This formulation can therefore, be used to represent birational maps in terms of matrices and determinants.

Let us illustrate this on rational parametric surface. Given a parametric surface

$$(x, y, z) = \left(\frac{s^2t - t - s^2 - 1}{s^2 + s^2t}, \frac{s^2t - t + s}{s^2 + s^2t}, \frac{2s^2 - 2t - 2}{s^2 + s^2t}\right),$$

we formulate the equations:

$$\begin{aligned} x(s^2 + s^2t) - s^2t + t + s^2 + 1 &= 0 \\ y(s^2 + s^2t) - s^2t - s + t &= 0 \\ z(s^2 + s^2t) - 2s^2 + 2t + 2 &= 0. \end{aligned}$$

Using Dixon's formulation, the resultant of these three equations can be expressed as:

$$\begin{vmatrix} 0 & 0 & 4 + 2x - z & -2 + 2x - z \\ 4 + 2x - z & -2 + 2x - z & 4 + 2x - z & -2 + 2x - z \\ -2 + 2x - z & -2 + 2x - z & -4 - 2x + 6y + z & -2 - 4x + 6y - z \\ -4 - 2x + 6y + z & -2 - 4x + 6y - z & -2 + 2x + 2z & 0 \end{vmatrix} \begin{pmatrix} 1 \\ t \\ s \\ st \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Using the Cramer's rule, we are able to compute the *inverse maps* as

$$t = \frac{\begin{vmatrix} 0 & 4 + 2 * x - z & -2 + 2 * x - z \\ -4 - 2 * x + z & 4 + 2 * x - z & -2 + 2 * x - z \\ 2 - 2 * x + z & -4 - 2 * x + 6 * y + z & -2 - 4 * x + 6 * y - z \end{vmatrix}}{\begin{vmatrix} 0 & 4 + 2 * x - z & -2 + 2 * x - z \\ -2 + 2 * x - z & 4 + 2 * x - z & -2 + 2 * x - z \\ -2 + 2 * x - z & -4 - 2 * x + 6 * y + z & -2 - 4 * x + 6 * y - z \end{vmatrix}}$$

and

$$s = \frac{\begin{vmatrix} 0 & 0 & -2+2*x-z \\ -2+2*x-z & -4-2*x+z & -2+2*x-z \\ -2+2*x-z & 2-2*x+z & -2-4*x+6*y-z \end{vmatrix}}{\begin{vmatrix} 0 & 4+2*x-z & -2+2*x-z \\ -2+2*x-z & 4+2*x-z & -2+2*x-z \\ -2+2*x-z & -4-2*x+6*y+z & -2-4*x+6*y-z \end{vmatrix}}.$$

The birational maps are expressed in terms of matrices and determinants and are very useful in the boundary computation algorithms. To accurately compute the inverse coordinates of a point x_0, y_0, z_0, we substitute the values and compute the determinants using Gaussian elimination with pivoting.

5.4 Geometric Constraint Systems

Solving geometric constraint systems is fundamental for many applications in mechanical assemblies, constraint-based sketching and design and kinematic analysis of robots and other mechanisms. An important class of the problems involve finding the positions, orientations and dimensions of a set of geometric entities that satisfy a set of geometric constraints. Earlier approaches to solve large constraint systems using this notion have either relied on the user to specify the sequence of operations. Recently, automated approaches based on kinematic simulation of mechanical linkages have appeared in the literature. The geometric constraints formulated in kinematics are applicable to tolerance analysis, assembly planning and constrained based design.

It turns out that kinematic solution to mechanical linkages has been extensively studied in robotics and mechanics. While the problem of direct kinematics of serial mechanisms and inverse kinematics of parallel mechanisms are simple, the inverse kinematics of serial mechanisms and direct kinematics of parallel manipulator has been relatively difficult. The latter problems reduce to solving algebraic equations in the most general case. Some other approaches to geometric constraint systems using algebraic formulation have been proposed in [25] and they reduce the problem to solving non linear algebraic equations as well.

We have applied our equation solving algorithm to the problem of inverse kinematics of general $6R$ manipulators. This had been a long standing problem in robotics literature. The absence of good solutions for the inverse kinematics of general $6R$ manipulators lead to all commercial manipulators being designed with geometric simplicity such that a closed form solution exists. In particular, the problem reduces to solving 6 equations in 6 unknowns and it had been recently shown that there can be at most 16 solutions. In particular, they reduce the problem to finding roots of a 16 degree univariate polynomial. However, these algorithm are of theoretical interest and their practical implementation suffers from numerical problems. Known implementations of inverse kinematics in the general case are based on continuation methods and they take about 10 sec. on an average (on an IBM 370-3090 mainframe) for a given pose of the end effector [26]. We applied our algorithm based on resultant, matrix polynomials and eigendecompositions to this problem. More details are given in [17]. The average running time for the given pose of the end effector is 10 milliseconds (on an IBM RS/6000 workstation) and giving up to 8 or more digits of accuracy. The robotics applications desire this kind of real time performance and the other algorithms have been found to be too slow for that. The technique is applicable for inverse kinematics of all serial manipulators and direct kinematics of parallel manipulators.

6 Performance Improvement

The matrices $\mathbf{C}$, $\mathbf{C}_1$ and $\mathbf{C}_2$ corresponding to the eigenvalue formulation seem to have a specific structure. The algorithm highlighted in Section 4 treats them as general non-symmetric matrices and uses the QR or QZ algorithm for eigendecomposition. In many cases, we may know a few of the eigenvalues depending upon the problem formulation and this information is being used in choosing the appropriate shifts along with the double shift QR algorithm for eigendecomposition. The order of these matrices correspond to the total number of non-trivial solution of a given system (Bezout bound for a dense system and BKK bound for the sparse system) and in most applications we are only interested in the eigenvalues lying in a particular domain. For example, many algorithms for boundary computations, intersection, ray tracing on Bézier curves and surfaces need the eigenvalues in the $[0, 1]$ domain only.

The matrices $\mathbf{C}, \mathbf{C}_1$ and $\mathbf{C}_2$ are relatively sparse. It turns out that the Macaulay formulation results in sparse matrices as well. In other words, the matrices $\mathbf{M}_i$ arising from Macaulay's formulation are sparse. As a result, it is worthwhile to use eigendecomposition algorithms for sparse matrices as opposed to the QR or QZ algorithm. In particular, we have tried the algorithm presented in [23] to compute invariant subspace of a real matrix by simultaneous iterations up to a user specified tolerance. The eigenvalues of the matrix are approximated from the invariant subspace. Although this algorithm is relatively fast as compared to the QR algorithm for eigendecomposition, its accuracy is not as good. We are currently investigating the tradeoffs between the accuracy and efficiency based on the choice of eigendecomposition algorithm.

6.1 Limitations of the Current Algorithm

The equation solving algorithm makes use of the resultant formulation of polynomial equations and reduces the problem to matrix computations. It turns out that good resultant formulations are known for systems containing up to 5 or 6 polynomial equations. Macaulay's formulation for general systems results in large and sparse matrices. The order of the matrix growns exponentially with the degrees of the equations and the number of equations. For example, polynomial systems with 6 or more equations arise in the computations of offsets, blends and voronoi diagrams and at the moment efficient resultant formulations are not known for these systems. The performance of algorithms based on Macaulay formulation and eigendecomposition is relatively slow. In many ways good resultant formulations are fundamental to the efficiency of this algorithm. The current algorithm performs well for polynomial systems consisting of up to 4 or 5 polynomials.

The matrices corresponding to the eigenvalue formulation are relatively structured. We have only been able to utilize the fact that they are sparse for systems with high Bezout bound. Furthermore, we are only interested in eigenvalues in a particular domain and no good sequential algorithms are known for that for these matrices.

7 Conclusion

The problem of solving a system of polynomial equations arises repeatedly in geometric and solid modeling applications. Algorithms based on resultants are well known in the literature. However, it is a widely conceived notion that algebraic approaches based on resultants suffer from numerical and efficiency problems, when it comes to application. In this paper, we utilized the fact that resultant of a system of polynomial equations is expressed in terms of matrices and determinants. As a result, we used algorithms and results from linear algebra and reduced the problem to matrix computations like Gauss elimination, eigendecomposition and SVD. Good implementations of the latter are available as part of linear algebra libraries and in the context of floating point computation their numerical accuracy is well understood. For most cases, we are able to compute accurate solutions using 64 bit IEEE floating point arithmetic. The algorithm has been successively applied to curve and surface intersections, finding distance from a point to a curve or a surface, locating singularities etc.

Due to the editorial policies of the journal, many related references could not be cited.

8 Acknowledgements

The author is grateful to James Demmel for productive discussions. Due to the limit on the number of references, many related references were not included.

References

[1] D. Lavender, A. Bowyer, J. Davenport, A. Wallis, and J. Woodwark. Voronoi diagrams of set-theoretic solid models. *IEEE Computer Graphics and Applications*, pages 69–77, September 1992.

[2] J.F. Canny. *The Complexity of Robot Motion Planning.* ACM Doctoral Dissertation Award. MIT Press, 1988.

[3] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces.* PhD thesis, Purdue University, 1983.

[4] J. Kajiya. Ray tracing parametric patches. *Computer Graphics*, 16(3):245–254, 1982.

[5] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[6] D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

[7] C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532, 1990.

[8] J.H. Wilkinson. The evaluation of the zeros of ill–conditioned polynomials. parts i and ii. *Numer. Math.*, 1:150–166 and 167–180, 1959.

[9] A.A.G. Requicha and J.R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, pages 31–44, September 1992.

[10] A. P. Morgan. Polynomial continuation and its relationship to the symbolic reduction of polynomial systems. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 23–45, 1992.

[11] T.W. Sederberg and T. Nishita. Curve intersection using bézier clipping. *Computer-Aided Design*, 22:538–549, 1990.

[12] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0*. SIAM, Philadelphia, 1992.

[13] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[14] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York, 1885.

[15] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 1993. To appear.

[16] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAECC*, 1993.

[17] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

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

[19] W. Auzinger and H.J. Stetter. An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. In *International Series of Numerical Mathematics*, volume 86, pages 11–30, 1986.

[20] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves. In *Graphics Interface '92*, pages 232–241, 1992. Revised version to appear in *ACM Transactions on Graphics*.

[21] D. Manocha and J.F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.

[22] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins Press, Baltimore, 1989.

[23] G.W. Stewart. Simultaneous iteration for computing invariant subspaces of non-hermitian matrices. *Numerische Mathematik*, 25:123–136, 1976.

[24] B.L. Van Der Waerden. *Modern Algebra (third edition)*. F. Ungar Publishing Co., New York, 1950.

[25] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407, 1991.

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