Solving Algebraic Systems using Matrix Computations *

Dinesh Manocha Shankar Krishnan
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
{manocha,krishnas}@cs.unc.edu

Abstract:
Finding zeros of algebraic sets is a fundamental problem in scientific and geometric computation. It arises in symbolic and numeric techniques used to manipulate sets of polynomial equations. In this paper, we outline algorithms and applications for solving zero and one dimensional algebraic sets using matrix computations. These algorithms make use of techniques from elimination theory and reduce the problem to finding singular sets of matrix polynomials. We make use of algorithms for eigendecomposition, singular value decomposition and Gaussian elimination to compute the singular sets. These algorithms have been implemented and perform very well in practice. We describe their application to computing conformations of molecular chains, inverse kinematics of serial robots, solid modeling and manufacturing.

1 Introduction

Finding the solution to a system of non-linear polynomial equations over a given field is a classical and fundamental problem in the computational literature. This problem arises in a number of symbolic and scientific applications. Furthermore, many applications in computer algebra, robotics, computer graphics, molecular modeling, geometric and solid modeling and computer vision use sets of polynomial equations for object representation and for defining constraints (as an algebraic set or semi-algebraic set). As a result, many geometric operations in these applications reduce to finding a solution to these polynomial equations. In this paper we restrict ourselves to zero and one dimensional algebraic sets.

The problem of finding numerical approximation to the solutions of a system of polynomial equations has been studied over the centuries. However, the current viewpoint is that there are *no good, general methods* for solving systems of more than one nonlinear equations, as highlighted in the book "Numerical Recipes: The Art of Scientific Computing" [PFTV90]. If the system corresponds to a set of linear equations, good algorithms and their implementations, in the form of linear algebra packages like LINPACK [BDMS79] and LAPACK [ABB$^+$92], are known and widely used. At the same time, a great deal of work has been done in the symbolic and numeric literature for finding roots of a univariate polynomial and good algorithms are known based on Sturm sequences, Uspensky's methods, Jenkins-Traub algorithm etc. When it comes to finding roots of a system of non-linear polynomial equations, symbolic methods like resultants, Gröbner bases, Ritt-Wu's algorithm can be used for eliminating variables and thereby reducing the problem to finding roots of univariate polynomials (for zero dimensional algebraic sets). A great deal of work has been done in the literature on the theory, implementation and application of these techniques. However, in the last decade most of the effort has been directed towards use of Gröbner bases. Most computer algebra systems, like Mathematica, Maple, Reduce etc. have a package for computing the Gröbner bases of an ideal. Motivated by the need of faster implementations, some special systems have been developed exclusively for Gröbner bases computation, like Macaulay and Cocoa. As far as resultants are concerned, many different formulations are known for sets of polynomials. In practice, almost all computer algebra systems have a routine for the ubiquitous Sylvester resultant of two polynomial equations. However, all these implementations have been found to be very slow for geometric applications on current hardware. Trying to compute the Gröbner bases of $3-4$ polynomials in $4-5$ variables of moderate degrees can take an incredible amount of time, if the system is able to compute it at all. Many a times the computation goes on until the machine runs out of all the virtual space (which is of the order of few gigabytes). This fact was highlighted in the context of geometric modeling applications by Hoffmann [Hof90]. Furthermore, this approach based on symbolic elimination and finding roots of a single polynomial needs to be implemented in exact

*Supported in part by Sloan Research Foundation, ARO Contract P-34982-MA, NSF Grant CCR-9319957, NSF Grant CCR-9625217 and ONR Contract N00014-94-1-0738

arithmetic as opposed to finite precision arithmetic (like IEEE floating point arithmetic available on current hardware). This is due to the fact that the problem of finding roots of a univariate polynomial may be ill–conditioned for high degree polynomials, as shown by Wilkinson [Wil59]. That slows down the overall computation.

In the context of floating point arithmetic, the two main approaches for zero-dimensional sets are 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 guesses to the solutions. This is rather difficult for most applications. Homotopy methods have a good theoretical background and proceed by following paths in the complex space [GZ79]. In theory, each path converges to a geometrically isolated solution. They have been implemented and tried on a variety of applications [Mor87, Mor92]. Some public domain software packages, like HOMPACK, are available as well. However, in practice current implementations and algorithms for homotopy methods suffer from many problems. The different paths being followed may not be geometrically isolated. In particular, this problem has been summarized by Horn in [Hor91], who applied continuation methods for the structure from motion problem in computer vision, as: "*one problem with continuation methods is that, while in theory paths of roots should never cross, in practice they often come close enough to permit path jumping, unless the path is followed with impractically tight tolerances*". Moreover continuation methods are considered to be computationally very demanding. For example, to solve the problem of inverse kinematics of 6 revolute joints (which can be reduced to solving 6 polynomial equations in 6 unknowns), where a performance of the order of milliseconds is desired, the best implementation of continuation methods takes about 10 seconds on an IBM $370-3090$ mainframe [WM91].

Many of the symbolic and numeric techniques highlighted above have been generalized to one-dimensional algebraic sets. Based on a classic theorem in algebraic geometry, an algebraic plane curve birationally equivalent to the space curve is computed using elimination techniques. Given an algebraic plane curve, techniques for desingularisation based on quadratic transformations are given in [Wal50, Abh90, AB88, Joh87]. These are useful for computing all the branches of the curve. However, the resulting algorithm can potentially be exponential in the degree of the curve. Algorithms based on Collins' *cylindrical algebraic decomposition* (CAD), [Col75], have been used for evaluating all components of algebraic curves [Arn83, SS83]. Its worst case complexity is doubly exponential in the number of variables. For plane curves, improved polynomial time algorithms based on CAD have been presented in [AF88]. However the exponent in terms of N (the degree of the curve) is rather high. Furthermore, these algorithms are implemented using exact arithmetic, which makes them slow and memory intensive in practice. Many applications in solid modeling and manufacturing need to evaluate very high degree algebraic curves (a few hundred) and these techniques are not suitable for such curves. Numerical and finite precision algorithms based on interval arithmetic [Moo79] and homotopy methods [GZ79, Mor92] have also been used for evaluating algebraic sets. While the former are slow in practice, the latter need points all the components and suffer from problems like component jumping. There is a considerable amount of emphasis in the solid modeling literature to evaluate surface intersections and offset curves [Hof89, SN91, Man92] and in vision literature to compute aspect graphs [PK92]. This includes algorithms for computing all components including the closed loops. However, these algorithms are somewhat restrictive and cannot be used for evaluating general algebraic curves.

In this paper we outline algorithms for solving algebraic systems using multipolynomial resultants and matrix computations. The algorithms symbolically eliminate variables from a given set of equations using resultants. However the resultant is expressed as a singular set of a matrix polynomial and we make use of linear algebra algorithms to compute the solutions. There is an elegant relationship between the kernel of these matrix polynomials and the variables being eliminated, which is being used for computing the rest of the variables and the birational map. We survey algorithms for zero-dimensional algebraic sets [Man92, Man94b, Man94a], highlights their application to inverse kinematics of serial robot manipulators [Man92] and conformational analysis of molecular chains [MZW95]. We evaluate one-dimensional algebraic sets up to a desired precision using tracing methods. We present algorithms to find start points on all components of the curves, prevent component jumping and handling singularities. The resulting algorithm has been applied to compute intersection of high degree rational surfaces and boundary representation of a submarine torpedo storage and handling room model.

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 show how resultants of polynomial systems can be expressed as matrix polynomials and the zero-dimensional root finding reduces to an eigenvalue problem. We extend the algorithm to one dimensional algebraic sets in Section 4. We describe its implementation and application to kinematics of robot chains and molecules, solid modeling and computation of boundary representation in Section 5. Finally, we describe some limitations of this approach in Section 6 and describe some problems for future research.

2 Background

Finding the zeros of a polynomial system is a fundamental problem in computational literature. This problem has been studied extensively in the classic as well as modern literature. In particular, *elimination theory*, a branch of classical algebraic geometry investigates the condition under which sets of polynomials have common roots. Some of its results were known at least a century ago [Sal85, Wae50] and still appear in modern treatments of algebraic geometry, at times in non-constructive form. The main result is the construction of a single resultant polynomial of n homogeneous polynomial equations in n unknowns, such that the vanishing of the resultant is a necessary and sufficient condition for the given system to have a non-trivial solution. We call this resultant the *Multipolynomial Resultant* of the given system of polynomial equations [Man92, MC93].

2.1 MultiPolynomial Resultants

Given a system of polynomial equations, the resultant is obtained by eliminating a set of variables and thereby computing a projection of the algebraic set in the lower dimension. The resultant is defined as a polynomial in the coefficients of the original system of equations. However, almost all formulations for computing the resultant express it in terms of matrices and determinants. The entries of the matrices are polynomial functions of the coefficient. We make use of this computational property in our algorithm for finding roots of polynomial equations, which is based on the interpretation that resultants linearize a non-linear problem.

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 [Mac02]. Many a time both determinants evaluate to zero. To compute the resultant we need to perturb the equations and use limiting arguments. This corresponds to computing the characteristic polynomials of both the determinants [Can88]. 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 [Dix08, Jou91, Mor25, MC27].

Most of the formulation presented in the classical literature correspond to computing the resultants of dense polynomial systems. More recently, resultants of sparse polynomial systems have received a lot attention in a newly developed area *sparse elimination theory.* In the late 1980's, Gel'fand and his colleagues began the study of discriminants and resultants of sparse polynomial systems [GKZ88]. Sparseness leads to a lowering of effective degree, and the sparse theory provides a simple direct method for proving bounds on the number of solutions. The main idea is to make use of sparseness to speedup equation solving and elimination of variables. The major benefit comes from the fact that the total number of solutions in the affine domain is much lower than the Bezout bound. Typical examples of sparse systems are those that describe the inverse kinematics for a 6R robot [Man92], forward kinematics for the Stewart platform [Mer92], camera motion from point matches [FM90], and geometric constraints describing two- or three-dimensional objects. As the dimension of the problem increases, the difference between the sparse and non-sparse bounds increases dramatically. A large gap between the Bezout bound and the actual number of solutions is not unusual for geometric problems. Bezout's theorem gives an exact count of the number of solutions in projective space, so most of these solutions are at infinity. The problem with trying to solve a system like this is that most methods have a complexity that depends on the Bezout bound. The Bezout bound is exact if all the coefficients of a polynomial system of some given degree are *generic*. Genericity is the requirement that the coefficients do not satisfy a set of algebraic relations. For example, random coefficients would be generic with high probability. However, many systems that arise in geometric problems are not generic.

A polynomial is sparse if many of its coefficients, compared to a generic polynomial of that degree, are zero. A bound derived from the set of non-zero coefficients is called a Bernstein bound. This bound is defined in terms of the mixed volumes of the newton polytopes corresponding to each equation. Bernstein showed that his bound is exact if all the coefficients of the polynomial system are generic [Ber75]. Resultant formulations in terms of matrices and determinants based on Bernstein bound have appeared in the literature as well [Stu91, SZ94, CE93].

Given a system of polynomial equations, sparse or dense, it is possible to express their resultant in terms of matrices and determinants. We use this linear algebra formulation in the algorithms presented in the following sections.

2.2 Matrix Computations

In this section we briefly review some techniques from linear algebra and numerical analysis. More details can be found in [GL89, Wil65].

2.2.1 Eigenvalues and Eigenvectors

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

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

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, [GL89], and their implementations are available as part of packages EISPACK, [GBDM77], and LAPACK [ABB+92].

2.2.2 Generalized Eigenvalue Problem

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

$$\mathbf{Ax} = s\mathbf{Bx}.$$

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 equation. If $\mathbf{B}$ is non–singular and its condition number 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:

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

However, $\mathbf{B}$ may have a high condition number and such a reduction may be numerically unstable. A better algorithm, called the QZ algorithm [GL89], applies orthogonal transformation to A and B to reduce A to Hessenberg form, to reduce B to upper triangular form, and then *implicitly* perform the QR algorithm on $B^{-1}A$ without ever forming it. This algorithm is in EISPACK [GBDM77] and in the most recent release of LAPACK.

2.2.3 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. The rank of a matrix can also be computed by Gauss elimination. However, there arise many situations where near rank deficiency prevails. Rounding errors and fuzzy data make rank determination a non–trivial exercise. In these situations, the numerical rank is easily characterized in terms of the SVD.

Given $\mathbf{A}$, an $m \times n$ 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 an $m \times n$ orthogonal matrix, $\mathbf{V}$ is an $n \times n$ 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* and columns of $\mathbf{U}$ and $\mathbf{V}$, denoted as u_i's and v_j's, are known as the left and right singular vectors, respectively [GL89].

2.2.4 Power Iterations

The largest or the smallest eigenvalue of a matrix (and the corresponding eigenvector) can be computed using the *Power method* [GL89]. 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.

2.2.5 Sparse Matrix Computations

The general formulation of resultants corresponding to Macaulay formulation results in sparse matrices [Mac02]. 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 as highlighted in the Power method and inverse iteration. For our applications, we use the algorithm highlighted in [Ste76] for computing the invariant subspaces and thereby the eigendecomposition of a sparse matrix.

3 Resultants and Matrix Polynomials

In this section, we show how the resultant of a system of polynomial equations can be expressed in terms of matrix polynomials. common solutions). In particular, resultants are being used to linearize the problem in terms of matrices and determinants. Initially we consider zero-dimensional algebraic sets and the same formulation is extendible to higher dimensional algebraic sets. More details are given in [Man92, Man94b].

Given a system of n equations in n unknowns,

$$\begin{aligned} F_1(x_1, x_2, \ldots, x_n) &= 0 \\ F_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ F_n(x_1, x_2, \ldots, x_n) &= 0. \end{aligned} \tag{3.1}$$

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 and in general, corresponds to the Bernstein bound for sparse systems 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 polynomials in 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. In particular, we append 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 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 turns that the resultant is expressed as a ratio of two determinants, say $Det(\mathbf{M})/Det(\mathbf{D})$. However the entries of D are independent of the u_i's. 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-vanishing minor.

Given $\mathbf{M}$, whose entries are polynomials in the u_i's, the resultant corresponding to its determinant can be factored into linear factors of the form [Wae50]:

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

where k is the total number of non-trivial solutions 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

$$\mathbf{M}(x_1) \begin{pmatrix} 1 \\ x_2 \\ \vdots \\ x_n \\ \vdots \\ x_2^d \\ x_3^d \\ \vdots \\ x_n^d \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.2}$$

$\mathbf{M}(x_1)$ 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 (3.2). We use this property along with those of matrix polynomials to compute the common solution of the polynomial equations.

3.1 Matrix Polynomials

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

$$\mathbf{M}(x_1) = \mathbf{M}_0 + \mathbf{M}_1 x_1 + \mathbf{M}_2 x_2 + \ldots + \mathbf{M}_l x_1^l, \tag{3.3}$$

where $\mathbf{M}_i$ are $m \times m$ 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, $\alpha_2, \alpha_3, \ldots, \alpha_n$.

It turns out that (3.3) corresponds to a matrix polynomial and our problem of computing the roots of the original system of polynomial equations corresponds to computing *generalized eigenvalues and eigenvectors of a matrix polynomial*. The given matrix polynomial is regular, if $det(\mathbf{M}(x_1)) \neq 0$, otherwise it is singular. The eigenvalues of a matrix polynomial consists of finite and infinite eigenvalues. In particular, when rank $M_l = m$, the determinant of $\mathbf{M}(x_1)$ is a polynomial of degree ml and all the eigenvalues of $\mathbf{M}(x_1)$ are finite. When rank $M_l < m$, the degree of $det(\mathbf{M}(x_1))$, say k, is less than ml. In this case $\mathbf{M}(x_1)$ has k finite eigenvalues and $(ml - k)$ infinite eigenvalues. In most applications we are only interested in computing the affine solutions of the original system of polynomial equations. Therefore, we are interested in computing the finite eigenvalues of a matrix polynomial. It is possible that the resultant formulation for a sparse polynomial system can result in a singular matrix polynomial. In such cases, we are still interested in computing the finite eigenvalues only.

Let us first consider the case, when the matrix polynomial is regular. Furthermore, we start our analysis by assuming that the leading matrix of the matrix polynomial, $\mathbf{M}_l$ 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

$$\overline{\mathbf{M}}(\alpha_1)\mathbf{v} = \mathbf{0}, \tag{3.4}$$

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 [Man92]:

Theorem 3.1 *Given the matrix polynomial,* $\overline{\mathbf{M}}(x_1)$ *the roots of the polynomial corresponding to its determinant are the eigenvalues of the matrix*

$$\mathbf{C} = \begin{pmatrix} \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_m & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{I}_m \\ -\overline{\mathbf{M}}_0 & -\overline{\mathbf{M}}_1 & -\overline{\mathbf{M}}_2 & \dots & -\overline{\mathbf{M}}_{l-1} \end{pmatrix}, \tag{3.5}$$

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 (3.4).*

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 [Man92], such that the problem of finding roots of the determinant of a matrix polynomial can be reduced to an eigenvalue problem. However there are cases where they may 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 [Man92]:

Theorem 3.2 *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*

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

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

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 it to eigenvalues in a particular domain [GL89]. Algorithms to compute selected eigenvalues of the these matrices based on power iterations and their structure are given in [Man94a].

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 finite 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 (3.2) 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. 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.
- 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 [BDM89]. More details of its application to finding solutions of polynomial equations are given in [Man92]. Such analysis is well developed for eigenvalues of a matrix and no equivalent analysis is known for higher multiplicity roots of a polynomial.

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 (3.2) 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 $x_1 = \alpha_1$ in the n equations 3.1 and solve them for the rest of the unknowns. This procedure is applied recursively.

4 One Dimensional Algebraic Sets

A set of polynomial equations whose solution corresponds to a one-dimensional algebraic set is given by:

$$\begin{aligned} F_1(u, v, w_1, w_2, \ldots, w_{n-1}) &= 0 \\ F_2(u, v, w_1, w_2, \ldots, w_{n-1}) &= 0 \\ &\vdots \\ F_n(u, v, w_1, w_2, \ldots, w_{n-1}) &= 0. \end{aligned}$$

We assume that this algebraic set consists of one dimensional components only. In case excess components are present, we can use perturbation techniques to compute the one-dimensional components. Moreover, we limit ourselves to evaluating all the components of the curve inside the region $D = [U_1, U_2] \times [V_1, V_2] \times [W_{(1,1)}, W_{(1,2)}] \times [W_{(2,1)}, W_{(2,2)}] \times \ldots \times [W_{(n-1,1)}, W_{(n-1,2)}] \in \Re^{n+1}$. We eliminate $n-1$ variables from these equations using multipolynomial resultant algorithms. Almost all the projections are one-to-one and result in a birationally equivalent curve. In our case, we perform a generic linear transformation and eliminate $w_1, \ldots, w_{n-1}$ from the resulting set. The resultant can be expressed as the determinant of a matrix polynomial [Dix08], $\mathbf{M}(u, v)$ say (u and v are the two remaining variables). The algorithm evaluates the resulting algebraic set and substitutes the values back into the original equations to discard the extraneous solutions. The

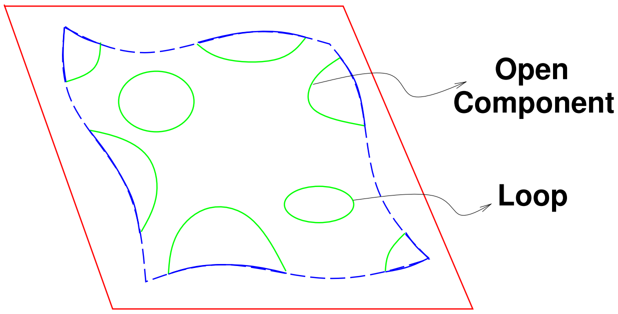


Figure 1: Multiple components of algebraic curves

degree of the algebraic curve, N, is given by the Bezout or Bernstein bound of the given system of equations.

A *singular set* of a matrix polynomial is defined as the set of all possible assignments to the variables of the polynomials that make the resulting matrix singular. We represent the plane curve (birationally equivalent to the algebraic curve) as the singular set of $\mathbf{M}(u, v)$. Given a point on the plane curve, (u_0, v_0), the corresponding point on the space curve, $(w_{1_0}, w_{2_0}, \ldots, w_{n-1_0})$, is computed using the *kernel* of $\mathbf{M}(u_0, v_0)$ [Man92].

In most applications, the one-dimensional sets encountered are of high degree, and are usually characterized by the presence of a number of *curve components* inside the domain of interest. The algorithm we propose uses *tracing* to evaluate the algebraic curves. The *tracing* algorithm uses the local geometry of the curve (like derivative information) to determine successive points on it. We use *inverse power iterations* to carry out curve tracing. Before applying the tracing method, however, it is important that at least one starting point is found on each of the (curve) components. This problem is similar to evaluating a zero-dimensional algebraic set and we employ eigenvalue methods to solve it. Because of the high degree nature of the algebraic curve, it is possible that it contains *singular points* or components which very close to each other. Unless proper care is taken, it is likely for the tracing algorithm to give incorrect results. We have developed an algorithm to subdivide the initial domain into smaller regions such that restricting the tracing step inside each region eliminates most of these problems.

4.1 Computation of Start Points

In this section, we shall describe algorithm for computing the start points on every component of the curve. This is very critical for the tracing algorithm. Inside the domain $[U_1, U_2] \times [V_1, V_2]$, the plane curve consists of two types of components: *open* and *closed*. *Open* components have at least one point lying on the boundary of the domain. *Closed* components, on the other hand, lie completely within the rectangular domain. We refer to them as

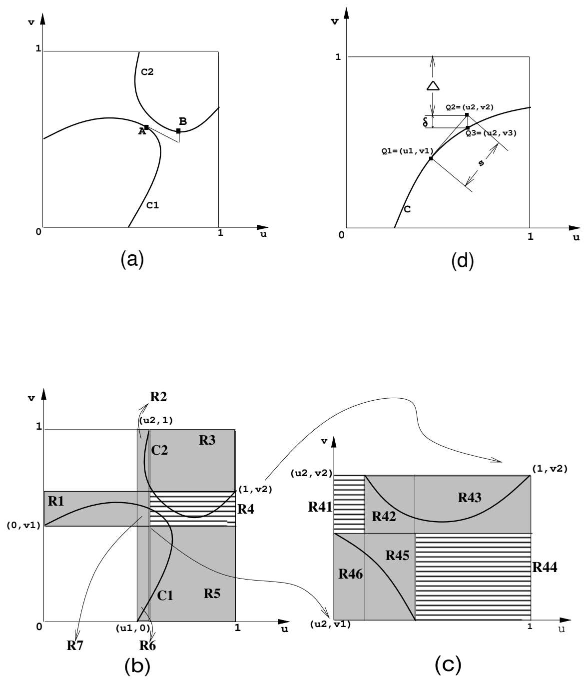


Figure 2: A single tracing step

loops (see Fig.1). In practice, finding start points on loops is significantly harder than those on open components.

From the definition of open components, it is clear that its endpoints must lie on the boundary of the domain. Starting points on all these components can be obtained by solving the system after substituting one of $u = U_1$, $u = U_2$, $v = V_1$ and $v = V_2$ into $\mathbf{M}(u,v)$ and solving the zero dimensional system using eigendecomposition.

The difficulty in identifying start points on closed components lies in the fact that loops have no such simple characterization as the one for open components. However, we show that we can use a simple algebraic property that would guide us to at least one point on every loop.

The curve $D(u,v) = 0$, where $D(u,v)$ is the determinant of $M(u,v)$, is an algebraic plane curve in the complex projective plane defined by u and v. We are, however, interested only in finding the part that lies in the portion of the real plane defined by $(u,v) \in [U_1, U_2] \times [V_1, V_2]$. If we relax this restriction so that one of the variables, say v, can take complex values, the intersection curve is defined as a continuous set consisting of real and complex components. Using this idea, we characterize loops as: *if the curve in the real domain* $[U_1, U_2] \times [V_1, V_2]$ *consists of a closed component, then two arbitrary complex conjugate paths meet at one of the real points (corresponding to a turning point) on the loop.*

Our method, then, to locate one point on each loop component is to follow all the complex paths (using tracing algorithm), starting from the domain boundary, until they either meet the real plane or leave the domain.

4.2 Tracing

Given the start points, we evaluate the curve using our tracing algorithm. A number of algorithms for tracing based on local iterative methods have been used in homotopy methods, surface interrogations and solutions of differential equations [Hof89, Mor92]. Given a point on the curve, an approximate value of the next point is obtained by taking a small step size in a direction determined by the local geometry of the curve. Based on the approximate value, these algorithms use local iterative methods like Newton's method to trace back on to the curve. Given a start point, these algorithms are applicable to trace algebraic curves as well. The main issues concerning all tracing algorithms are *(i) convergence back on to the curve, (ii) component jumping, and (iii) singular points.* The convergence problems arising from the behavior of Newton's method are well known. *Component jumping* can occur when two components of the curve are relatively close to each other as shown in fig.2(a). In this case, the tracing algorithm can jump from point A on component $C1$ to point B on component $C2$. Singular points are points where the curve self-intersects or the tangent vector vanishes. They typically lead to multiple branches around the singular point. The tracing algorithm has to determine these singular points efficiently and trace all the branches.

In order to prevent component jumping, we subdivide the original domain (in u, v-space) into smaller regions such that each region has at most one curve component. Once all the starting points are determined (on open components and loops), a region has only one component when it has at most two starting points. Subdivision of the domain is performed by determining *isolines* (lines of constant u or v in the domain) based on the starting points previously computed. These *isolines* are then intersected with the algebraic curve using eigenvalue methods. The subdivision stops when each new region has at most one curve component. Fig.2(b) and (c) show the application of this algorithm to the curve in Fig.2(a). Sometimes, in the presence of singular points, it is not possible to separate out the various components (see Fig.3). If the algorithm is unable to isolate single curves in a domain after repeated levels of subdivision, then either the curve has a singularity, or some components of the curve lie very close to each other. Singular points are determined by local minimization of an energy function which is dependent on the curve equation and its partial derivatives. If the local minima is close to zero, then there is a singular point present. This method of identifying singularities is susceptible to numerical errors especially if they lie very close to each other. We, however, believe that such pathological cases are rare in practice.

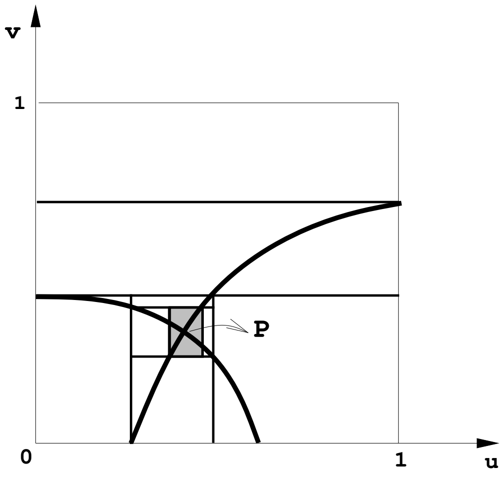


Figure 3: Curve with singular point

More details of the component splitting algorithm and singularity detection are presented in [KM95].

After performing component splitting, we have regions with at most one curve component and its starting points. Starting from these points, the *tracing* algorithm computes successive curve points using the local geometry of the curve. Let the component be C (see Fig.2(d)). Given a point $\mathbf{Q}_1 = (u_1, v_1)$ the skeleton of the tracing algorithm is given below. Furthermore, it is required that any point on the piecewise representation of the curve is not more than ϵ apart from the curve.

- Compute $D^u(u_1, v_1)$ and $D^v(u_1, v_1)$, the partial derivatives of the curve with respect to u and v, respectively [MC91]. This is the vector (on the plane) normal to the plane curve.
- Given the normal vector, find the unit vector corresponding to the tangent. Let this vector be (t_u, t_v).
- Find an approximate point $\mathbf{Q}_2 = (u_2, v_2)$, where $u_2 = u_1 + t_u * S$, and $v_2 = v_1 + t_v * S$, where S is the step size and $S \leq \epsilon$.
- Using (u_2, v_2), converge back to the curve at $\mathbf{Q}_3 = (u_2, v_3)$, if $\mid t_u \mid > \mid t_v \mid$, or to $\mathbf{Q}_3 = (u_3, v_2)$, if $\mid t_v \mid > \mid t_u \mid$ using *inverse power iterations*.

A single tracing step is shown in fig.2(d). The main part of the tracing algorithm is the use of inverse power iterations to converge back to the curve. We will explain this in detail. For the rest of the analysis let us assume that $\mathbf{Q}_3 = (u_2, v_3)$.

Inverse Power Iterations: In a single step of the tracing algorithm, we need to compute the eigenvalue of $\mathbf{M}(u_2, v)$ which is closest to v_2 (fig.2(d)). As a result, we compute the companion matrix $\mathbf{C}$ from $\mathbf{M}(u_2, v)$ (see eq. (3.5)) and set $s = v_2$. Therefore, we need to compute the smallest eigenvalue of the matrix $\mathbf{C} - s\mathbf{I}$. The smallest eigenvalue of $\mathbf{C} - s\mathbf{I}$ corresponds to the largest eigenvalue of $(\mathbf{C} - s\mathbf{I})^{-1}$. Instead of computing the inverse explicitly (which is numerically unstable), we use inverse power iterations [GL89].

To solve the matrix system efficiently, we use LU decomposition of the matrix $(\mathbf{C} - s\mathbf{I})$ using Gaussian elimination. We also make use of the structure of the matrix to reduce its complexity. Given s, let $\mathbf{B} = \mathbf{C} - s\mathbf{I}$. $\mathbf{B}$ is of the form:

$$\mathbf{B} = \begin{pmatrix} \alpha_1\mathbf{I}_n & \mathbf{I}_n & \mathbf{0} & \dots & \mathbf{0} \\ & \ddots & \dots & & \\ & \mathbf{0} & \mathbf{0} & \dots & \alpha_1\mathbf{I}_n & \mathbf{I}_n \\ & \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3 & \dots & \mathbf{P}_m \end{pmatrix}$$

where α_1 is a function of s. The LU decomposition of $\mathbf{B}$ has the form:

$$\mathbf{B} = \begin{pmatrix} \alpha_1\mathbf{I}_n & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \alpha_1\mathbf{I}_n & \dots & \mathbf{0} \\ \ddots & \dots & & \\ \mathbf{R}_1 & \mathbf{R}_2 & \dots & \mathbf{L}_m \end{pmatrix} \begin{pmatrix} \mathbf{I}_n & \frac{1}{\alpha_1}\mathbf{I}_n & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n & \dots & \mathbf{0} \\ \ddots & \dots & & \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{U}_m \end{pmatrix}$$

where $\mathbf{L}_m$ and $\mathbf{U}_m$ correspond to the LU decomposition of $\mathbf{R}_m$. $\mathbf{R}_i$'s can be easily computed from the $\mathbf{P}_i$'s. LU decomposition can sometimes face numerical problems if the matrix B is ill-conditioned. In such cases, LQ factorization can be used, where Q and L are orthogonal and lower triangular matrices respectively.

A key property of inverse power iteration is that it converges to the eigenvalue closest to s. If the closest eigenvalue to s is a complex conjugate pair, the power method does not converge to any real value. In such cases, the tracing algorithm chooses a smaller step size for computation.

5 Implementation and Applications

We have implemented the algorithms using linear algebra libraries. The application of the algorithm to zero and one dimensional sets involves:

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 rational function of the coefficients.
2. **Zero-Dimensional Sets:** Reduce the problem to an eigenvalue problem and extract the solutions from the

eigenvalues and eigenvectors. 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 a simple eigenvalue or a generalized eigenvalue problem. 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.

3. **One-Dimensional Sets:** Compute an algebraic plane curve birationally equivalent to the space curve. Compute a start point on all the component of the lower dimensional algebraic curve. This involves finding a point on the open components using the zero-dimensional solver. The real solutions in the domain correspond to a start point on an open component. The complex solutions of the zero dimensional solver are used as start points for tracing paths in the complex space. After finding a start point on each component, the algorithm performs component splitting to decompose the domain into regions containing at most one component. Components with singularities and multiple branches are isolated using component splitting and local methods. Finally, we use inverse power iterations to trace the components.

All the parts mentioned above are relatively simple to implement, given the linear algebra routines (BLAS). A major feature of the algorithm is that at each stage the numerical accuracy of the operations involved is well understood. As a result, it is possible to come up with tight bounds on the accuracy of the resulting solution. In the rest of this section, we describe the application of the algorithm to compute configurations of robot chains, molecular conformations and boundary representation of solid models.

5.1 Inverse Kinematics of Robot and Molecular Chains

The inverse kinematics problem for general serial mechanisms is a fundamental problem in robotics and molecular modeling. Robot manipulators are modeled as a rigid serial chain consisting of revolute and prismatic joints. The position and orientation of the end effector is a direct function of the joint variables. In most robotics applications, we are given the pose of the end effector and the problem of inverse kinematics corresponds to computing the joint displacements for that pose. Inverse kinematics has been a fundamental problem in robotics literature for more than three decades [SV89].

Over the last few decades, the use of computer tools has gained a lot of importance in molecular modeling and drug design. One of the fundamental problem in drug design is that of finding the three dimensional structures of complex molecules. This is important as the activity of a molecule depends on how well its three dimensional structure fits the three dimensional structure of a target receptor. These three dimensional configurations are called *conformers*. Finding conformers corresponding to minimal energy configuration involves three dimensional manipulation, given the position and orientation of the molecular chain [GoS70]. This is also useful for *protein folding* applications. We model the molecular chain as a rigid structure [BA82].

Robot Kinematics: We initially use the terminology from the robotics and mechanics literature to formulate the problem. The geometric problem in molecular modeling is similar, except it is described using chemical bonds, bond angles and dihedral angles (as opposed to links, length of the links, joint angles etc.). Each link is represented by the line along its joint axis and the common normal to the next joint axis. In the case of parallel joints, any of the common normals can be chosen. The configuration of the end-effector of the robot is described using six parameters. As a result, we consider robots with six degrees of freedom. The links of the manipulator are numbered from 1 to 6, as shown in Fig. 4. The base link is 1, and the outermost link or hand is n. A coordinate system is attached to each link for describing the relative arrangements among the various links. The coordinate system attached to the ith link is numbered i. The 4×4 transformation matrix relating $i+1$ coordinate system to i coordinate system is [SV89]:

$$\mathbf{A}_i = \begin{pmatrix} c_i & -s_i\lambda_i & s_i\mu_i & a_ic_i \\ s_i & c_i\lambda_i & -c_i\mu_i & a_is_i \\ 0 & \mu_i & \lambda_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad (5.6)$$

where

$$s_i = \sin\theta_i, \quad c_i = \cos\theta_i, \quad \theta_i \text{ is the } i\text{th joint rotation angle,}$$

$$\mu_i = \sin\alpha_i, \quad \lambda_i = \cos\alpha_i,$$

$$\alpha_i \text{ is the twist angle between the axes of joints } i \text{ and } i+1,$$

$$a_i \text{ is the length of link } i+1,$$

$$d_i \text{ is the offset distance at joint } i.$$

For a given robot with revolute joints we are given the a_i's, d_i's, μ_i's and λ_i's. For the inverse kinematics problem we are also given the pose of the end-effector, attached to link 6. This pose is described with respect to the base link or

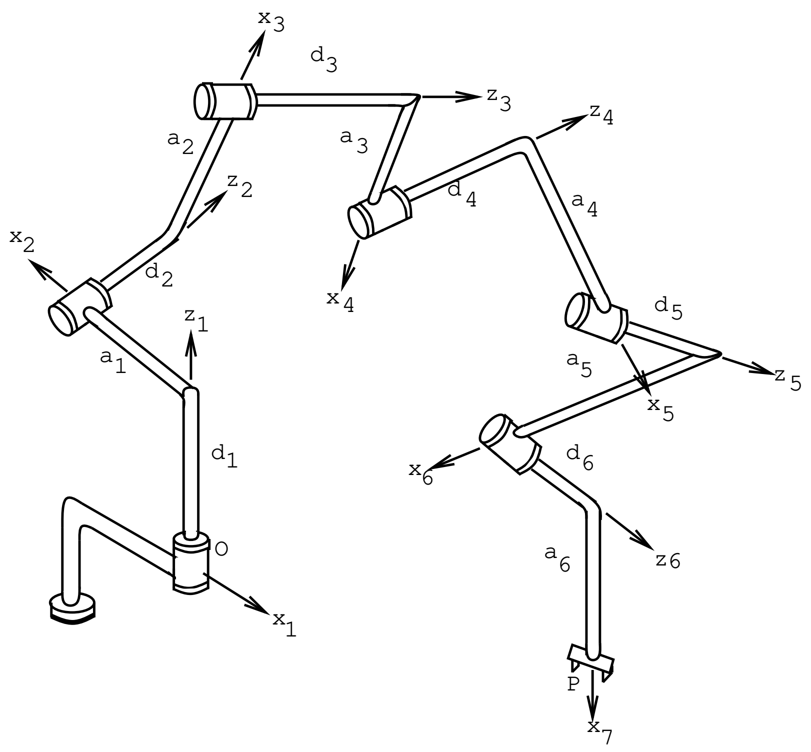


Figure 4: A general 6R robot manipulator

link 1. We represent this pose as:

$$\mathbf{A}_{hand} = \begin{pmatrix} l_x & m_x & n_x & q_x \\ l_y & m_y & n_y & q_y \\ l_z & m_z & n_z & q_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The problem of inverse kinematics corresponds to computing the joint angles, $\theta_1, \theta_2, \ldots \theta_n$ such that

$$\mathbf{A}_1\mathbf{A}_2 \ldots \mathbf{A}_6 = \mathbf{A}_{hand}. \tag{5.7}$$

The left hand side entries of the matrix equation given above are functions of the sines and cosines of the joint angles. Furthermore, this matrix equation corresponds to 12 scalar equations. Since the matrix formed by the first 3 rows and 3 columns of $\mathbf{A}_{hand}$ is orthonormal, only 6 of the 12 equations are independent. Thus, the problem of inverse kinematics of general manipulators with 6 joints corresponds to solving 6 equations for 6 unknowns.

These problem has been extensively studied in the robotics literature. In particular, a sparse resultant for these particular equations was derived by [LL88, RR89] and it was shown there can be at most 16 different configurations of the robot chain for a given pose of the end effector. We made use of the sparse resultant formulation and combined it with matrix computations to reduce to an eigenvalue problem. In particular, we obtain a 24×24 matrix and applied different eigenvalue algorithms to compute the solutions [Man92]. The performance of different algorithms for inverse kinematics, along with the machine platform are highlighted in Table 1.

Molecule Conformations: A molecular chain is classified using bond lengths, bond angles and dihedral angles. We initially model them as rigid chains using fixed bond lengths and bond angles. It has been shown that the minimal energy conformations computed by fixing these parameters is a good guess to the actual minima and can be used along with a perturbation treatment for the energy minimization [GoS70]. Given fixed bond-lengths and bond-angles, we present algorithms for two fundamental problems in conformational energy calculations: *ring closure* and *local conformational deformations* [GoS70]. These problems reduce to solving a system of algebraic equations. The problem of ring closure arises when we deal with cyclic molecules, i.e. the calculation of dihedral angles which correspond to exact ring closure. In conformational energy minimization procedures, one selects a starting conformation and alters it step by step such that the corresponding conformational energy decreases monotonically. However, a small change in a dihedral angle located near the middle of the long chain causes a drastic change in the overall conformation of the molecule. It is therefore, desired to

Algorithm	Reference	Machine	Average Time
Continuation	(Wampler and Morgan 1991 [WM91])	IBM 370-3090	10 sec.
Resultant - QR algorithm	(Manocha 1992 [Man92])	IBM RS/6000	0.011 sec.
Resultant - Selected Solutions	(Manocha 1994 [Man94a])	IBM RS/6000	0.0043 sec.

Table 1: Relative performance of various algorithms for inverse kinematics

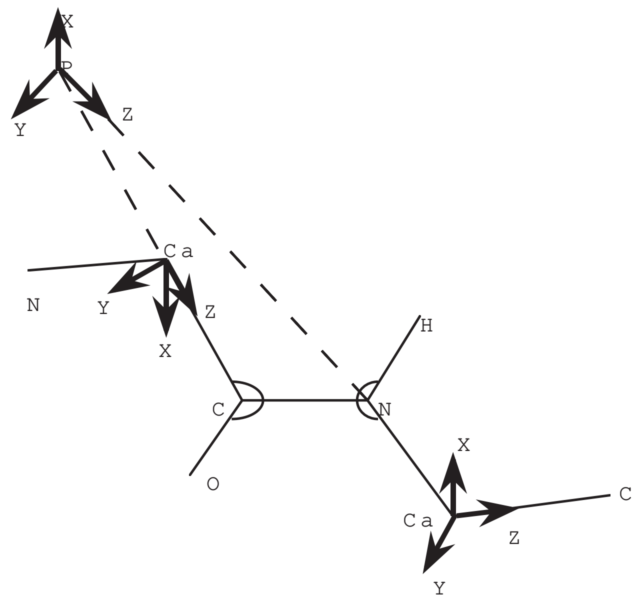


Figure 5: Coordinate systems on a polypeptide unit based on DH formalism

have a cooperative variation of angles which confines the conformational changes to a *local* section of the chain. As a result we are interested in calculation of changes in dihedral angles, which cause local deformations of conformations in long polymer chains [GoS70, BA82].

We use the Denavit-Hartenberg (DH) notation to model a molecular chain. The algorithm proceeds by assigning coordinate systems and computing the DH parameters of a local chain. In Fig. 5.1, we highlight the DH formulation for a *peptide* unit and the corresponding parameters are shown in Fig. 6. Given this formulation, the problems of ring closure for cyclic chains and local deformations reduce to inverse kinematics 5.7 [MZW95]. Depending on the geometry of the molecular chain, we reduce it to a 24×24 or 32×32 eigenvalue problem. The resulting algorithm takes anywhere from $10-20$ milliseconds to compute all the solu-

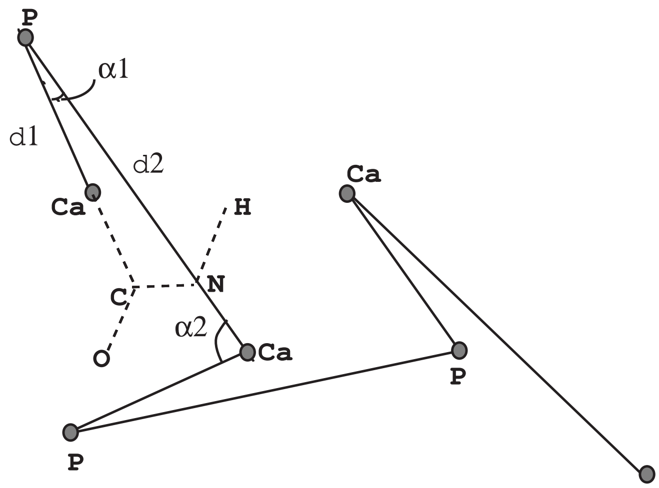
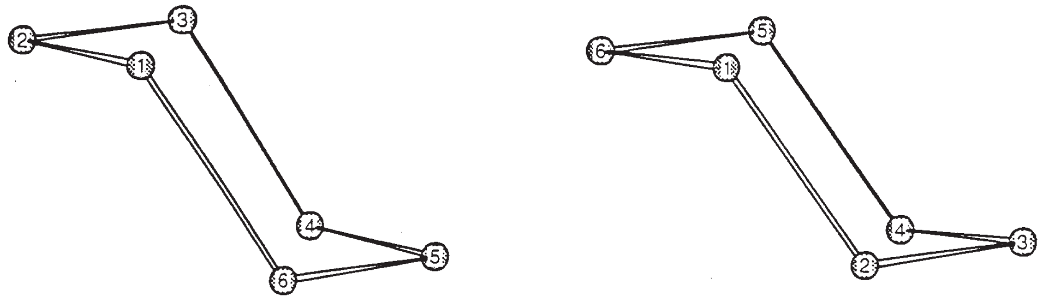


Figure 6: DH parameters for a polypeptide unit

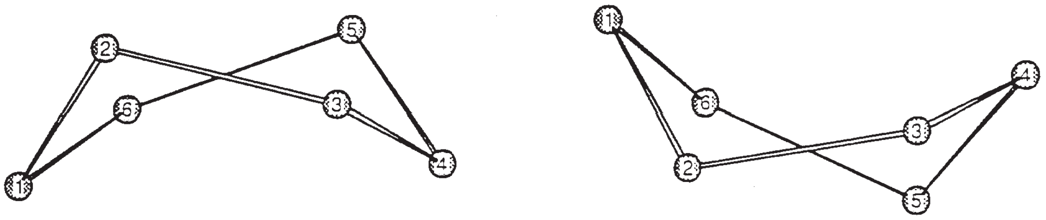
tions. Besides peptide chains, we used this formulation to study the conformations of cyclohexanes. For a cyclohexane, all $C-C$ bond lengths are fixed at 1.526 Angstrom, and $C-C-C$ bond angles are fixed at 110.4 degree. Since we are only concerned with main chain conformation, the bond lengths and bond angles of $C-H$ are irrelevant to our analysis. It was proved in [GoS73] that cyclohexane has infinite geometrically possible conformation due to its structural symmetry. In the following example, we slightly increase the length of the last bond. Since the symmetry no longer holds, we obtain a finite set of solutions. The DH parameters for this set of bond geometry are listed in Table 2. The end effector is an identity matrix because of the ring structure.

Four sets of solution computed by the algorithm are listed in Table 3. The first two sets correspond to the dihedral angles of the *chair* conformation in Fig. 7(a). The last two sets correspond to those of the *twisted boat* conformation in Fig. 7(b).

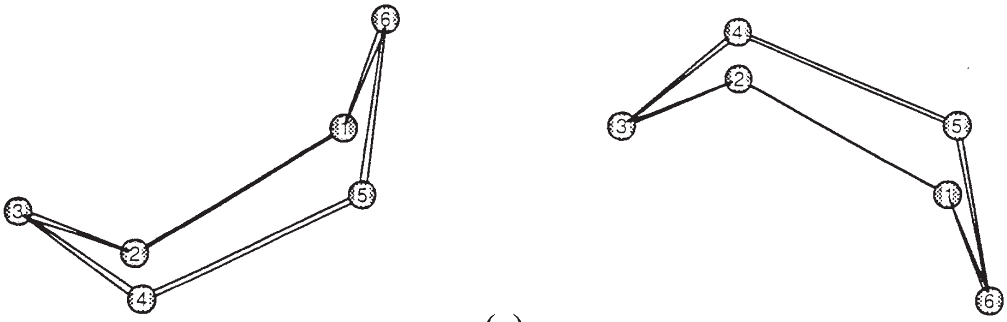
We should point out here that the four solutions are



(a)



(b)



(c)

Figure 7: Cyclohexane conformations: (a). (top) *chair* conformation, (b). (middle) *twisted boat* conformation, (c). (bottom) *boat* conformation

Number	Link Length	Offset Distance	Twist Angle
i	a_i	d_i	α_i
1	0.0	1.525	69.6
2	0.0	1.526	69.6
3	0.0	1.526	69.6
4	0.0	1.526	69.6
5	0.0	1.526	69.6
6	0.0	1.526	69.6

Table 2: Denavit-Hartenberg Parameters of Cyclohexane Model

i	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	-57.69	57.67	-57.62	57.60	-57.62	57.67
2	57.69	-57.67	57.62	-57.60	57.62	-57.67
3	67.78	-32.04	-31.98	67.70	-31.98	-32.04
4	-67.78	32.04	31.98	-67.70	31.98	32.04

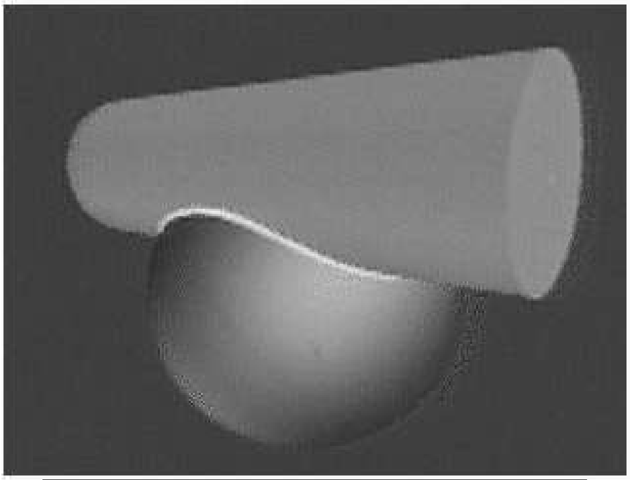
Table 3: The dihedral angles corresponding to the Cyclohexane with standard bond length and bond angles

obtained based on the assumption that bond lengths and bond angles are fixed as listed in Table 2. If variation of bond length or bond angle is allowed, other possible conformations may exist. Through a series of small perturbations of the bond geometry, we noticed that the *chair* conformation is more stable than the *boat* and *twisted boat* conformation. In all cases of small perturbation we tried, there are no more than four solutions for the ring closure problem. Two of these four solutions constantly correspond to the *chair* conformation, while the other two correspond to either the *boat* or the *twisted boat* conformation. Furthermore, the dihedral angles of the *chair* conformations stay almost constant, while those of either the *boat* or the *twisted boat* vary a lot for small perturbation s on bond geometry. Our result is in accord with previous experimental and theoretical results that the *chair* conformation is the favored conformation of cyclohexane.

5.2 Application to Solid Modeling

The field of solid modeling deals with the design and representation of physical objects. The two major representation schemata used in solid modeling are constructive solid geometry (CSG) and boundary representations (B-rep). Every solid in CSG is represented as a tree, where the leaf nodes are primitive solids and each intermediate node correspond to a boolean operation (an intersection, union or difference). The primitive solids may correspond to polyhedra, quadrics or solids whose surfaces are represented using piecewise rational parametric or piecewise

[redacted]



Fi[redacted]ere

algebraic surfaces [Hof89]. Fig. 8 demonstrates two difference operation and the final solid obtained after the difference operations. Most modelers use the CSG representation for model construction and compute the B-rep's for geometric operations.

A fundamental operation in the computation of the B-rep from a CSG operation is to evaluate their curve of intersection. If the solid boundaries are represented by appropriate algebraic equations, the intersection curve is nothing but the common solutions of a one-dimensional algebraic system of equations. Most of the current modelers use rational parametric surfaces. Given two parametric surfaces, $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$,

$$\begin{aligned}\mathbf{F}(s,t) &= (X(s,t), Y(s,t), Z(s,t), W(s,t)) \\ \mathbf{G}(u,v) &= (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))\end{aligned}$$

represented in homogeneous coordinates, their intersection curve is defined as the set of common points in 3-space and is given by the vector equation $\mathbf{F}(s,t) = \mathbf{G}(u,v)$. This results in the following set of three equations in four unknowns:

$$\begin{aligned}F_1(s,t,u,v) = X(s,t)\overline{W}(u,v) - \overline{X}(u,v)W(s,t) &= 0 \\ F_2(s,t,u,v) = Y(s,t)\overline{W}(u,v) - \overline{Y}(u,v)W(s,t) &= 0(5.8) \\ F_3(s,t,u,v) = Z(s,t)\overline{W}(u,v) - \overline{Z}(u,v)W(s,t) &= 0,\end{aligned}$$

and the domain of the intersection curve is $(s,t,u,v) \in [0,1] \times [0,1] \times [0,1] \times [0,1]$. This forms a system of three equations in four unknowns and generically results in a one-dimensional set (the intersection curve). Fig. 9 shows the intersection curve between a sphere and a cylinder.

We have applied the algorithms for evaluating one-dimensional algebraic sets to computation of B-rep's from CSG models. The one dimensional curves correspond to

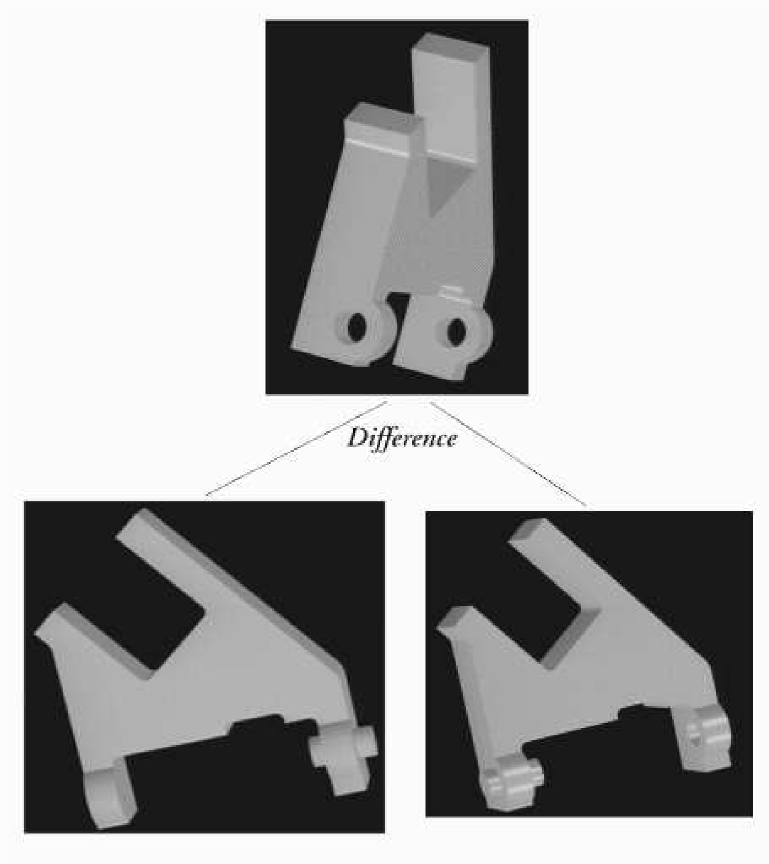


Figure 8: A one-level CSG tree

the boundary of the new solid. The overall algorithm involves ray-shooting and performing in/out tests besides curve evaluation. We have developed a solid modeling system applied to compute the boundary representation of parts of a submarine storage and handling system. The CSG model was provided to us courtesy of Electric Boat division of General Dynamics. Fig. 10 shows the pivot model of the submarine storage and handling room. It consists of 168 solids and the boundary is described using 4524 trimmed rational spline surfaces. The boundary of the trimmed surfaces corresponds to the algebraic intersection curve.

6 Future Work

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 frequently and at the moment good resultant formulations are not known for these systems. The performance of algorithms based on Macaulay formulation and eigendecomposition may be 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 algebraic complexity. Further research is needed to develop efficient and robust serial and parallel algorithms to compute the eigendecomposition of such matrices.

7 Acknowledgements

We would like to acknowledge our collaborators: Subodh Kumar, William Wright, Yunshan Zhu and members of UNC Walkthrough group. The CSG model of a submarine torpedo room was provided to us by Greg Angelini, Jim

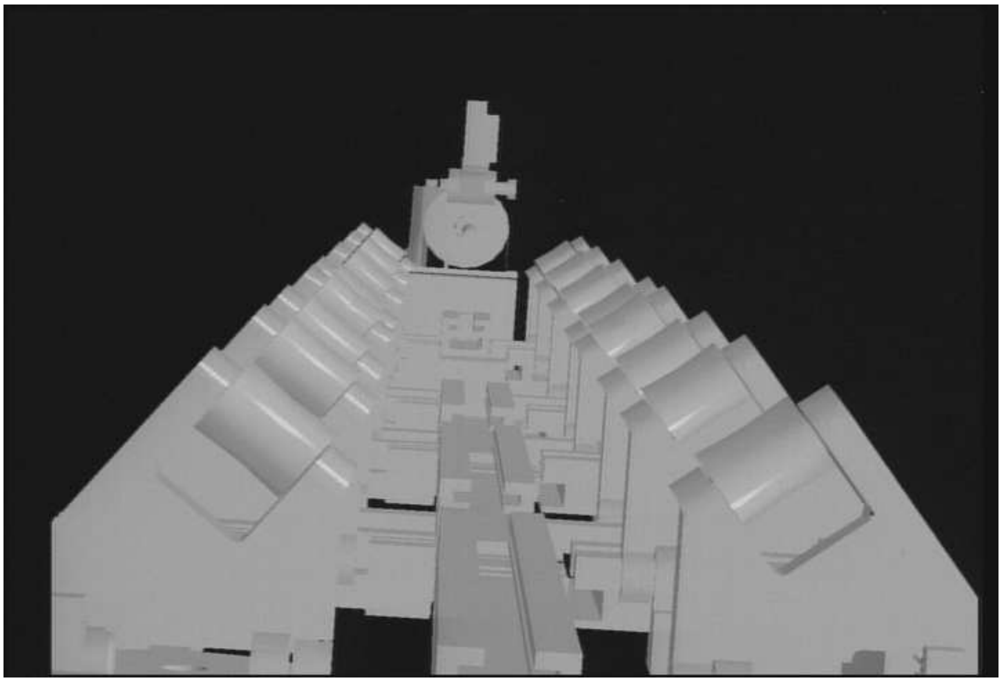


Figure 10: Pivot model of submarine storage and handling room

Bourdeaux and Ken Fast at Electric Boat, a subsidiary of General Dynamics.

References

[AB88] S.S. Abhyankar and C. Bajaj. Computations with algebraic curves. In *Lecture Notes in Computer Science*, volume 358, pages 279–284. Springer Verlag, 1988.

[ABB+92] 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.

[Abh90] S. S. Abhyankar. *Algebraic Geometry for Scientists and Engineers.* American Mathematical Society, Providence, R. I., 1990.

[AF88] S. Arnborg and H. Feng. Algebraic decomposition of regular curves. *Journal of Symbolic Computation*, 5:131–140, 1988.

[Arn83] D. S. Arnon. Topologically reliable display of algebraic curves. *Computer Graphics*, 17:219–227, 1983.

[AS86] 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.

[BA82] Ulrich Burkert and Norman Allinger. *Molecular Mechanics.* ACS Monographs. Americal Chemical Society, Washington, D.C., 1982.

[BDM89] Z. Bai, J. Demmel, and A. McKenney. On the conditioning of the nonsymmetric eigenproblem: Theory and software. Computer Science Dept. Technical Report 469, Courant Institute, New York, NY, October 1989. (LAPACK Working Note #13).

[BDMS79] J. Bunch, J. Dongarra, C. Moler, and G. W. Stewart. *LINPACK User's Guide.* SIAM, Philadelphia, 1979.

[Ber75] D. N. Bernshtein. The number of roots of a system of equations. *Funktsional'nyi Analiz i Ego Prilozheniya*, 9(3):1–4, 1975.

[BGW88] 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.

[Can88] J.F. Canny. *The Complexity of Robot Motion Planning.* ACM Doctoral Dissertation Award. MIT Press, 1988.

[CE93] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAECC*, pages 89–104. Springer-Verlag, 1993.

[Col75] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture Notes in Computer Science*, number 33, Springer-Verlag, 1975.

[Dix08] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[FM90] O.D. Faugeras and S. Maybank. Motion from point matches: Multiplicity of solutions. *International Journal of Computer Vision*, 4:225–246, 1990.

[GBDM77] B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51. Springer-Verlag, Berlin, 1977.

[GKZ88] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. Equations of hypergeometric type and newton polyhedra. *Doklady AN SSSR*, 300:529–534, 1988.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins Press, Baltimore, 1989.

[GoS70] N. Go and H.A. Scherga. Ring closure and local conformational deformations of chain molecules. *Macromolecules*, 3(2):178–187, 1970.

[GoS73] N. Go and H.A. Scherga. Ring closure in chain molecules with c_n, i or s_{2n} symmetry. *Macromolecules*, 6(2):273–281, 1973.

[GZ79] C.B. Garcia and W.I. Zangwill. Finding all solutions to polynomial systems and other systems of equations. *Math. Prog.*, 16:159–176, 1979.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.

[Hof90] C.M. Hoffmann. Algebraic and numeric techniques for offsets and blends. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computations of Curves and Surfaces*, pages 499–528. Kluwer Academic Publishers, 1990.

[Hor91] B.K.P. Horn. Relative orientation revisited. *Journal of Optical Society of America*, 8(10):1630–1638, 1991.

[Joh87] J.K. Johnstone. *The Sorting of points along an algebraic curve*. PhD thesis, Cornell University, Department of Computer Science, 1987.

[Jou91] Jean-Pierre Jouanolou. *Le Formalisme du Résultant*, volume 90 of *Advances in Mathematics*. 1991.

[KM95] S. Krishnan and D. Manocha. Numeric-symbolic algorithms for evaluating one-dimensional algebraic sets. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, 1995.

[LL88] H.Y. Lee and C.G. Liang. A new vector theory for the analysis of spatial mechanisms. *Mechanisms and Machine Theory*, 23(3):209–217, 1988.

[Mac02] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, 1(33):3–27, May 1902.

[Man92] 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.

[Man94a] D. Manocha. Computing selected solutions of polynomial equations. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 1–8, Oxford, England, 1994. ACM Press.

[Man94b] D. Manocha. Solving systems of polynomial equations. *IEEE Computer Graphics and Applications*, pages 46–55, March 1994. Special Issue on Solid Modeling.

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

[MC91] 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.

[MC93] D. Manocha and J.F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.

[Mer92] J-P. Merlet. Direct kinematics and assembly modes of parallel manipulators. *International Journal of Robotics Research*, 11(2):150–162, 1992.

[Moo79] R.E. Moore. *Methods and applications of interval analysis*. SIAM studies in applied mathematics. Siam, 1979.

[Mor25] F. Morley. The eliminant of a net of curves. *American Journal of Mathematics*, 47:91–97, 1925.

[Mor87] A.P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice–Hall, Englewood Cliffs, New Jersey, 1987.

[Mor92] 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.

[MZW95] D. Manocha, Y. Zhu, and W. Wright. Conformational analysis of molecular chains using nano-kinematics. *Computer Application of Biological Sciences (CABIOS)*, pages 71–86, 1995.

[PFTV90] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1990.

[PK92] J. Ponce and D.J. Kriegman. Elimination theory and computer vision: Recognition and positioning of curved 3d objects from range, intensity, or contours. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 123–146, 1992.

[RR89] M. Raghavan and B. Roth. Kinematic analysis of the 6r manipulator of general geometry. In *International Symposium on Robotics Research*, pages 314–320, Tokyo, 1989.

[Sal85] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York, 1885.

[SN91] T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.

[SS83] J. T. Schwartz and M. Sharir. On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351, 1983.

[Ste76] G.W. Stewart. Simultaneous iteration for computing invariant subspaces of non-hermitian matrices. *Numerische Mathematik*, 25:123–136, 1976.

[Stu91] B. Sturmfels. Sparse elimination theory. In D. Eisenbud and L. Robbiano, editors, *Computational Algebraic Geometry and Commutative Algebra*. Cambridge University Press, 1991.

[SV89] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. John Wiley and Sons, 1989.

[SZ94] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 1994. To appear.

[Wae50] B.L. Van Der Waerden. *Modern Algebra (third edition)*. F. Ungar Publishing Co., New York, 1950.

[Wal50] R.J. Walker. *Algebraic Curves*. Princeton University Press, New Jersey, 1950.

[Wil59] 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.

[Wil65] J.H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, Oxford, 1965.

[WM91] 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.