Algorithms for Intersecting Parametric and Algebraic Curves

Dinesh Manocha
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720, USA

James Demmel
Computer Science Division
and Mathematics Department
University of California at Berkeley
Berkeley, CA 94720, USA

Abstract

The problem of computing the intersection of parametric and algebraic curves arises in many applications of computer graphics, geometric and solid modeling. Earlier algorithms are based on techniques from Elimination theory or subdivision and iteration. The former is however, restricted to low degree curves. This is mainly due to issues of efficiency and numerical stability. In this paper we use Elimination theory and express the resultant of the equations of intersection as a matrix determinant. The matrix itself rather than its symbolic determinant, a polynomial, is used as the representation. The algorithm for intersection corresponds to substituting the other equation to construct an equivalent matrix such that the intersection points can be extracted from the eigenvalues and eigenvectors of the latter. Moreover, the algebraic and geometric multiplicities of the eigenvalues give us information about the intersection (multiplicity, tangential intersection etc.). As a result we are able to accurately compute higher order intersections. The main advantage of this approach lies in its *efficiency and robustness*. Moreover, the numerical accuracy of these operations is well understood. For almost all cases we are able to compute accurate answers in 64 bit IEEE floating point arithmetic.

Keywords: Intersection, Curves, Eigenvalues, Accuracy, Robustness.

1. Introduction

The problems of computing the intersection of parametric and algebraic curves are fundamental to geometric and solid modeling. Parametric curves, like B-splines and Bézier curves, are extensively used in the modeling systems and algebraic plane curves are becoming popular as well [Hof89, MM89, SP86, Sed89]. Intersection is a primitive operation in the computation of a boundary representation from a CSG (constructive solid geometry) model in a CAD system. Other applications of intersection include hidden curve removal for free form surfaces, [EC90]. Algorithms for computing the intersection of these curves have been extensively studied in the literature.

As far as computing the intersection of rational parametric curves is concerned, algorithms based on implicitization [Sed83], Bézier subdivision [LR80] and interval arithmetic [KM83] are well known. The implicitization approach computes the implicit form of one of the curve using resultants [Sed83]. Given the implicit representation of one curve, substitute the second parametrization and obtain a univariate polynomial in its parameter. The problem of intersection corresponds to computing the roots of the resulting polynomial. The Bézier subdivision relies on the convex hull property of Bézier curves and de Casteljau algorithm for subdividing Bézier curves. The resulting algorithm performs a linearly converging binary search. It has been improved by more effective use of the convex hull property [SWZ89]. The interval arithmetic uses an idea similar to subdivision. Each curve is preprocessed to determine its vertical and horizontal tangents, and the curve is divided into 'intervals' which have vertical or horizontal tangents only at the endpoints.

The relative performance and accuracy of these algorithms is highlighted in [SP86]. In particular, implicitization based approaches are faster as compared to other methods for curves of degree up to four. However, their relative performance degrades for higher degree curves. This is mainly due to issues of numerical stability and their effect on the choice of representation and algorithms for root finding. For curves of degree greater than three, the resulting univariate polynomial has degree 16 or higher. The problem of computing real roots of such high degree polynomials can be ill–conditioned [Wil59]. Using exact arithmetic or symbolic techniques to overcome the numerical problems have a considerable effect on the efficiency of the problem and therefore, subdivision based algorithms perform better.

The algorithms for algebraic curve intersection are analogous to those of intersecting parametric curves. Re-

sultants can be used to eliminate one variable from the two equations corresponding to the curves. The problem of intersection corresponds to computing roots of the resulting univariate polynomial. This approach causes numerical problems for higher degree curves (greater than four). A robust algorithm based on subdivision has been presented in [Sed89]. However, resultant based algorithms are considered to be the fastest for lower degree curves.

In many applications, the intersection may be of higher order involving tangencies and singular points. Such instance are rather common in industrial applications [MM89]. Most algorithms require special handling for tangencies and thereby requiring additional computation for detecting them. In fact algorithms based on subdivision and Newton–type techniques often fail to accurately compute the intersections in such cases. Special techniques for computing first order tangential contacts of parametric curves are given in [MM89]. [Sed89] presents modification of his algorithm for computing all double points of an algebraic curve in a triangular domain. However, no efficient and accurate techniques are known for computing higher order intersections for general cases.

In this paper we present efficient and robust algorithms for intersecting parametric and algebraic curves. For parametric curves we implicitize one of the curves and represent the implicit form as a *matrix determinant.* However, we do not compute the symbolic determinant and express the implicit formulation as a matrix. The idea of matrix determinant has been used in [MC91] to represent and evaluate the intersection of rational parametric surfaces. Given the implicit form, we substitute the other parametrization into the matrix formulation and use the resulting matrix to construct a numerical matrix such that the intersection points can be computed from its eigendecompositon. This is in contrast with expanding the symbolic determinant and finding the roots of the resulting polynomial. The advantages of this technique lie in *efficiency, robustness and numerical accuracy.* The algorithms for computing eigenvalues and eigenvectors of a matrix are *backward stable* and fast implementations are available as part of packages like EISPACK and LAPACK [GL89, ABB+90]. Furthermore, we effectively use the algebraic and geometric multiplicities of the eigenvalues to determine the exact multiplicity of the intersection. The algorithm for intersecting algebraic curves is rather similar, except the relationship between algebraic and geometric multiplicities of the eigenvalue and the multiplicity of intersection is different.

The rest of the paper is organized in the following manner. In section 2 we present our notation and review techniques from Elimination theory for implicitizing parametric curves. Furthermore, we show that the problems of intersecting parametric and algebraic curves can be reduced to computing roots of polynomials expressed as matrix determinants. In section 3, we present results from linear algebra and numerical analysis being used in the algorithm. Section 4 deals with reducing the problem of root finding to computing the eigendecomposition. Given the eigenvalues and eigenvectors, we compute the intersection points of parametric curves in the domain of interest. We also discuss the performance and robustness of the resulting algorithm. Section 5 deals with higher order intersections and illustrates the technique with examples.

2. Parametric and Algebraic Curves

A rational Bézier curve is of the form [BBB87]:

$$\mathbf{P}(t) = (X(t), Y(t)) = \frac{\Sigma_{i=0}^{n} w_i \mathbf{P}_i B_{i,n}(t)}{\Sigma_{i=0}^{n} w_i B_{i,n}(t)}, \qquad 0 \le t \le 1$$

where $\mathbf{P}_i = (X_i, Y_i)$ are the coordinates of a control point, w_i is the weight of the control point and $B_{i,n}(t)$ corresponds to the Bernstein polynomial, $B_{i,n} = \binom{n}{i}(1-t)^{n-i}t^i$. Rational curves like B-splines can be converted into a series of Bézier curves by knot insertion algorithms [BBB87]. Thus, the problem of intersecting rational curves can be reduced to intersecting Bézier curves. Each of these curves is described by its corresponding control polygon and the curve is always contained in the convex hull of the control points. Therefore, the intersection of the convex hull of two such curves is a necessary condition for the intersection of curves. One such instance has been highlighted in Fig. I.

Algebraic plane curves are generally expressed in standard power basis:

$$F(x, y) = \Sigma_{i+j \le n} c_{ij} x^i y^j = 0.$$

They can also be represented in Bernstein basis. The problem of intersection corresponds to computing the common points on such curves in a particular domain.

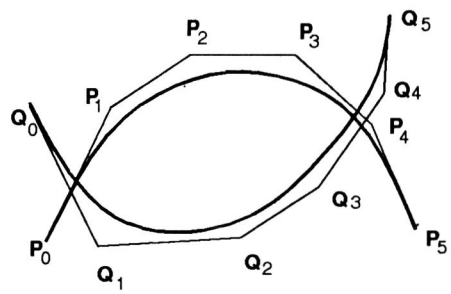

Fig. I
Intersection of Bézier curves

2.1. Resultants

Given two polynomials in one unknown, their resultant is a polynomial in their coefficients. Moreover, the vanishing of the resultant is a necessary and sufficient condition for the two polynomials to have a common root. Three methods are known in the literature for computing the resultant, owing to Sylvester, Bezout and Cayley [Sal85].

Each of them expresses the resultant as determinant of a matrix. The order of the matrix is different for different methods. We use Cayley's formulation as it results in a matrix of minimum order.

Given two polynomials, $F(x)$ and $G(x)$ of degree m and n, respectively. Without loss of generality we assume that $m \geq n$. Let's consider the bivariate polynomial

$$P(x,\alpha) = \frac{F(x)G(\alpha) - F(\alpha)G(x)}{x - \alpha}.$$

$P(x,\alpha)$ is a polynomial of degree $m-1$ in x and also in α. Let us represent it as

$$P(x,\alpha) = P_0(x) + P_1(x)\alpha + P_2(x)\alpha^2 + \ldots + P_{m-1}\alpha^{m-1},$$

where $P_i(x)$ is a polynomial of degree $m-1$ in x. The polynomials $P_i(x)$ can be written as follows:

$$\begin{pmatrix} P_0(x) \\ P_1(x) \\ \vdots \\ P_{m-1}(x) \end{pmatrix} = \begin{pmatrix} P_{0,0} & P_{0,1} & \ldots & P_{0,m-1} \\ P_{1,0} & P_{1,1} & \ldots & P_{1,m-1} \\ \vdots & \vdots & \vdots & \vdots \\ P_{m-1,0} & P_{m-1,1} & \ldots & P_{m-1,m-1} \end{pmatrix} \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{m-1} \end{pmatrix}$$

Let us denote the $m \times m$ matrix by M. The determinant of M is the resultant of $F(x)$ and $G(x)$ [Sal85]. Let us assume that $x = x_0$ is a common root of the two polynomials. Therefore, $P(x_0,\alpha) = 0$ for all α. As a result $P_i(x_0) = 0$ for $0 \leq i < m$. This condition corresponds to the fact that M is singular and $[1 \; x_0 \; x_0^2 \; \ldots \; x_0^{m-1}]^T$ is a vector in the kernel of M.

We use Cayley's resultant formulation for implicitizing parametric curves and eliminating a variable from a pair of algebraic equations representing algebraic plane curves. More details on the properties of implicit representation, computation and accuracy are given in [MD92].

2.2. Implicitizing Parametric Curves

Given a rational Bézier curve, $\mathbf{P}(t)$:

$$\mathbf{P}(t) = \left(\frac{x(t)}{w(t)}, \frac{y(t)}{w(t)}\right) = \left(\frac{\Sigma_{i=0}^n w_i X_i B_{i,n}(t)}{\Sigma_{i=0}^n w_i B_{i,n}(t)}, \frac{\Sigma_{i=0}^n w_i Y_i B_{i,n}(t)}{\Sigma_{i=0}^n w_i B_{i,n}(t)}\right).$$

To implicitize the curve we consider the following system of equations

$$\begin{aligned} Xw(t) - Wx(t) &= 0 \\ Yw(t) - Wy(t) &= 0. \end{aligned} \tag{1}$$

Consider them as polynomials in t and X, Y, W are treated as symbolic coefficients. The implicit representation corresponds to the resultant of (1) [Sed83].

We express the resultant as a matrix determinant. In this case the matrix has order n. Each entry of the matrix is of the form $\alpha_0 X + \alpha_1 Y + \alpha_2 W$, where $\alpha_0, \alpha_1, \alpha_2$ are scalars and functions of the control points and weights used to represent the Bézier curve. The algorithm for computing the entries of the matrix assumes that the polynomial are expressed in power basis. However, converting from Bézier to power basis can introduce numerical errors [FR87]. To circumvent this problem we perform a reparametrization. Given

$$\mathbf{P}(t) = \left(\frac{\Sigma_{i=0}^n w_i X_i B_{i,n}(t)}{\Sigma_{i=0}^n w_i B_{i,n}(t)}, \frac{\Sigma_{i=0}^n w_i Y_i B_{i,n}(t)}{\Sigma_{i=0}^n w_i B_{i,n}(t)}\right).$$

Dividing by $(1-t)^n$ and substituting $s = \frac{t}{(1-t)}$ results in

$$\overline{\mathbf{P}}(s) = \left(\frac{\Sigma_{i=0}^n w_i X_i \binom{n}{i} s^i}{\Sigma_{i=0}^n w_i \binom{n}{i} s^i}, \frac{\Sigma_{i=0}^n w_i Y_i \binom{n}{i} s^i}{\Sigma_{i=0}^n w_i \binom{n}{i} s^i}\right)$$

The rest of the algorithm proceeds by computing the implicit representation of $\overline{\mathbf{P}}(s)$ and computing a matrix formulation by Cayley's method as

$$M \begin{pmatrix} 1 \\ s \\ s^2 \\ \vdots \\ s^{m-1} \end{pmatrix} = M \begin{pmatrix} (1-t)^{m-1} \\ t(1-t)^{m-2} \\ t^2(1-t)^{m-3} \\ \vdots \\ t^{m-1} \end{pmatrix} \tag{2}$$

The right hand side is obtained by substituting $s = \frac{t}{(1-t)}$ and multiplying vector by $(1-t)^{m-1}$. Later on, this relationship will be used to compute the inverse coordinates of the intersection points.

2.3. Intersecting Parametric Curves

Given two rational Bézier curves, $\mathbf{P}(t)$ and $\mathbf{Q}(u)$ of degree m and n respectively, the intersection algorithm proceeds by implicitizing $\mathbf{P}(t)$ and obtaining a $m \times m$ matrix M, whose entries are linear combinations of symbolic coefficients X, Y, W. The second parametrization $\mathbf{Q}(u) = (\overline{x}(u), \overline{y}(u), \overline{w}(u))$ is substituted into the matrix formulation. It results in a matrix polynomial $M(u)$ such that each of its entry is a linear combination of $\overline{x}(u), \overline{y}(u)$ and $\overline{w}(u)$. The intersection points correspond to the roots of

$$\text{Determinant}(M(u)) = 0. \tag{3}$$

2.4. Intersecting Algebraic Curves

In this section we consider the intersection of two algebraic plane curves. They are represented as zeros of $F(x,y)$ and $G(x,y)$, polynomials of degree m and n, respectively. The polynomials may be represented in power basis or Bernstein basis. Let the points of intersection in the domain of interest be $(x_1,y_1),\ldots,(x_k,y_k)$. To simplify the problem we compute the projection of these points on the x-axis. Algebraically projection corresponds to computing the resultant of $F(x,y)$ and $G(x,y)$ by treating them as polynomials in y and expressing the coefficients as polynomials in x.

We compute the resultant using Cayley's formulation. In case, the curves are expressed in Bernstein basis, we use the reparametrization highlighted in the previous section for implicitization. The resultant is expressed as a matrix determinant and each entry of the matrix is a polynomial in x. Let us denote the matrix by $M(x)$. The problem of intersection corresponds to computing roots of Determinant$(M(x)) = 0$.

3. Matrix Computations

In this section we present techniques from linear algebra and numerical analysis. We also highlight the numerical accuracy of the problems and the algorithm used to solve these problems in terms of their *condition numbers.*

3.1. Eigenvalues and Eigenvectors

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

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

where λ is the eigenvalue and $\mathbf{x}$ is the eigenvector. The eigenvalues of a matrix are the roots of its characteristic polynomial determinant$(A - \lambda I) = 0$. An eigenvalue, λ_i, of multiplicity k corresponds to a root of multiplicity k of the characteristic polynomial. This multiplicity is also defined as the *algebraic multiplicity* of the eigenvalue. Moreover, the dimension of the kernel of $(A - \lambda_i I)$ is the *geometric multiplicity* of λ_i. The geometric multiplicity is bounded by the algebraic multiplicity.

Most eigendecomposition algorithms make use of the symmetric orthogonal transformations of the form $A' = QAQ^{-1}$, where Q is any non-singular orthogonal $n \times n$ matrix. This transformation has the characteristic that the eigenvalues of A and A' are identical. Furthermore, if $\mathbf{y}$ is an eigenvector of A', $Q^{-1}\mathbf{y}$ is an eigenvector of A. The running time of the eigendecomposition algorithms is $O(n^3)$. However, the constant in front of n^3 can be as high as 25 for computing all the eigenvalues and eigenvectors.

3.2. Generalized Eigenvalue Problem

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

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

We represent this problem as eigenvalues of $A - \lambda B$. The vectors $\mathbf{x}$ correspond to the eigenvectors of this equation. If 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 B^{-1} and thereby obtaining:

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

However, B may have a high condition number and such a reduction can cause numerical problems. Algorithms for the generalized eigenvalue problems apply orthogonal transformations to A and B. In particular, we use the QZ algorithm for computing the eigenvalues and eigenvectors for this problem [GL89]. Its running time is $O(n^3)$. However, the constant can be as high as 75. Generally, it is slower by a factor of 2.5 to 3 as compared to QR algorithm for computing eigenvalues and eigenvectors of a matrix.

3.3. Condition Numbers

The condition number of a problem measures the sensitivity of a solution to small changes in the input. A problem is *ill-conditioned* if its condition number is large, and *ill-posed* if its condition number is infinite. These condition numbers are used to bound errors in computed solutions of numerical problems. More details on condition numbers are given in [GL89, Wil65]. The implementations of these condition number computations are available as part of LAPACK [BDM89].

In our algorithm, we perform computations like matrix inverse and computing eigenvalues and eigenvectors of a matrix. Therefore, we are concerned with the numerical accuracy of these operations.

4. Reduction to Eigenvalue Problem

In this section we consider the problem of intersecting parametric curves and reduce it computing an eigendecomposition of a matrix. The same reduction is applicable to the intersection of algebraic plane curves as explained in [MD92].

In section 2 we had reduced the problem of intersecting parametric curves, $\mathbf{P}(t)$ and $\mathbf{Q}(u)$ of degree m and n, respectively, to finding roots of a matrix determinant as shown in (3). Each entry of the $m \times m$ matrix, $M(u)$, is a linear combination of Bernstein polynomials of degree n in u. Let us represent it as a matrix polynomial

$$M(u) = M_n u^n + M_{n-1}u^{n-1}(1-u) + \ldots + M_0(1-u)^n,$$

where M_i is a matrix with numeric entries. On dividing the equation by $(1-u)^n$ we obtain a polynomial of the form

$$M_n(\frac{u}{1-u})^n + M_{n-1}(\frac{u}{1-u})^{n-1} + \ldots + M_0.$$

Substitute $s = \frac{u}{1-u}$ and the new polynomial is of the form

$$\overline{M}(s) = M_n s^n + M_{n-1}s^{n-1} + \ldots + M_0. \qquad (4)$$

In the original problem we were interested in the roots of Determinant$(M(u)) = 0$ in the range $[0,1]$. However, after reparametrizing we want to compute the roots of Determinant$(\overline{M}(s)) = 0$ in the range $[0,\infty]$. This can result in overflow problems if the original system has a root $u \approx 1$. In such cases M_n is nearly singular or ill–conditioned. Our algorithm takes care of such cases by linear transformations or using projective coordinates.

Let us consider the case when M_n is a well–conditioned matrix. Given $\overline{M}(s)$, we multiply the matrix polynomial by M_n^{-1}. As a result we obtain

$$\overline{M}'(s) = I_m s^n + \overline{M}_{n-1} s^{n-1} + \ldots + \overline{M}_0,$$

where I_m is an $m \times m$ identity matrix and $\overline{M}_i = M_n^{-1} M_i$ for all $i < n$. Given the matrix polynomial $\overline{M}'(s)$ we compute a $nm \times nm$ matrix of the form

$$C = \begin{pmatrix} 0 & I_m & 0 & \ldots & 0 \\ 0 & 0 & I_m & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & I_m \\ -\overline{M}_0 & -\overline{M}_1 & -\overline{M}_2 & \ldots & -\overline{M}_{n-1} \end{pmatrix}, \quad (5)$$

such that the eigenvalues of C correspond exactly to the roots of $\mathrm{Det}(\overline{M}(s)) = 0$. Furthermore C is a numeric matrix of order mn. The proof of this property is given as part of Theorem 1.1 [GLR82]. The relationship between C and $\overline{M}'(s)$ is identical to that between the characteristic polynomial of a matrix and its equivalent companion matrix [Wil65].

Let λ be an eigenvalue of C. As a result $\overline{M}'(\lambda)$ is a singular matrix. Let $\mathbf{v} = [v_1\ v_2\ \ldots\ v_m]^T$ be the vector in the kernel of $\overline{M}'(\lambda)$ such that

$$\overline{M}'(\lambda)\,(v_1\ v_2\ \ldots\ v_m)^T = (0\ 0\ \ldots\ 0)^T. \quad (6)$$

Corollary I: *The eigenvector of C corresponding to the eigenvalue λ has the form $\mathbf{V} = [\mathbf{v}\ \lambda\mathbf{v}\ \lambda^2\mathbf{v}\ \ldots\ \lambda^{n-1}\mathbf{v}]^T$, where $\mathbf{V}$ is a $mn \times 1$ vector.*
Proof: [MD92].

It follows that the eigenvalues of C correspond exactly to the preimages of intersection points on $\mathbf{Q}(u)$. However, we are only interested in the eigenvalues in the range $s_0 \in [0,\infty]$ and the preimages on the curve are obtained by substituting $u_0 = \frac{s_0}{1+s_0}$. This gives us a list of all the intersection points on $\mathbf{Q}(u_0)$ such that $u_0 \in [0,1]$. However, these points on $\mathbf{P}(t)$ may not lie in the range $t \in [0,1]$. As a result it is important for us to compute the preimage of the intersection point $(x_0, y_0, w_0) = \mathbf{Q}(u_0)$ with respect to $\mathbf{P}(t)$. We use the property of the linear system of equations (2) and Corollary I.

Let us assume that (x_0, y_0, w_0) is a simple point on $\mathbf{P}(t)$. Points of higher multiplicity are accounted for in the next section. Substitute for $(X, Y, W) = (x_0, y_0, w_0)$ in the matrix, M as shown in (2), corresponding to the implicit representation of $P(t)$. The resulting matrix is singular and let us assume that its kernel has dimension one. Kernels of higher dimension correspond to higher order intersection. The vector in the kernel corresponds to $\mathbf{v}$ shown in (6). Given the eigenvector of C corresponding to the eigenvalue s_0, use Corollary I to compute the eigenvector $\mathbf{v}$. Given $\mathbf{v}$ we use the structure of the linear system to compute the preimage of the point (x_0, y_0, w_0) by using the relation

$$\left((1-t_0)^{m-1}\ t_0(1-t_0)^{m-2}\ \ldots\ t_0^{m-1}\right)^T = k\,(v_1\ v_2\ \ldots\ v_m)^T,$$

where $k \neq 0$ is a constant. Thus, $t_0 = \frac{v_2}{v_1+v_2}$. The relationship between the eigenvalue s_0 of C, elements v_1, v_2 of the eigenvector $\mathbf{V}$ corresponding to s_0 and the point of intersection (x_0, y_0, w_0) can be expressed as

$$(x_0, y_0, w_0) = \mathbf{Q}(\frac{s_0}{1+s_0}) = \mathbf{P}(\frac{v_2}{v_1+v_2}). \quad (7)$$

As a result we are able to compute all the points of intersection in the domain of interest by computing the eigendecomposition of C.

Let us consider the case, when the matrix M_n in (4) is ill–conditioned. One such case occurs when the preimage of the point of intersection on $\mathbf{P}(t)$ is $t_0 \approx 1$. As a result, computation of M_n^{-1} and the corresponding reduction to the eigenvalue problem can introduce numerical problems. The general approach for solving such cases is the reduction to generalized eigenvalue problem. In this case we construct companion matrices of the form [GLR82, Section 7.2]

$$C_1 = \begin{pmatrix} I_n & \ldots & 0 & 0 \\ 0 & I_n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & I_n & 0 \\ 0 & \ldots & 0 & M_n \end{pmatrix}, C_2 = \begin{pmatrix} 0 & -I_n & 0 & \ldots & 0 \\ 0 & 0 & -I_n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & -I_n \\ M_0 & M_1 & \ldots & M_{n-2} & M_{n-1} \end{pmatrix}$$

such that the eigenvalues of $C_1 s + C_2$ correspond exactly to the roots of $\mathrm{Det}(M(s)) = 0$. Furthermore, the eigenvectors of this generalized system have a structure similar to $\mathbf{V}$ highlighted in Corollary I and is used for computing the inverse image of the intersection point with respect to $\mathbf{P}(t)$.

Solving a generalized eigenvalue system is more expensive than the normal eigenvalue system (almost a factor of 3). In many cases, we can perform a linear transformation on the coordinate of the matrix polynomial and reduce the resulting problem to an eigenvalue problem. The basic idea involves transforming $s = \frac{a\bar{s}+b}{c\bar{s}+d}$, where a, b, c and d are random numbers. The matrix polynomial $\overline{M}(s)$ in (4) is transformed into

$$P(\bar{s}) = (c\bar{s}+d)^n M(\frac{a\bar{s}+b}{c\bar{s}+d})$$

$$\Rightarrow P(\bar{s}) = P_n\bar{s}^n + P_{n-1}\bar{s}^{n-1} + \ldots + P_1\bar{s} + P_0,$$

where P_i's are computed from the M_j's. If P_n is a well-conditioned matrix then the problem of intersection is reduced to an eigenvalue problem, otherwise use a different transformation (by a different choice of a,b,c and d). The linear transformation is performed up to four or five times. If all the resulting leading matrices, P_n, are ill-conditioned, the intersection problem is reduced to a generalized eigenvalue problem. There are cases when any linear transformation can result in an ill-conditioned leading matrix. Furthermore, the domain of the eigenvalue system obtained after transformation is $[s_1, s_2]$ or $[s_2, s_1]$ depending upon the signs of a, b, c and d, where $s_1 = -\frac{b}{a}$ and $s_2 = -\frac{d}{c}$.

4.1. Implementation and Performance

The reduction to an eigenvalue or a generalized eigenvalue system involves estimating the condition number of a matrix, computing the matrix inverse and finding the eigenvalues of a matrix. For eigenvalues lying in the domain of interest, we compute the corresponding eigenvectors. Furthermore, we also compute the condition number of each eigenvalue in the domain of interest. The condition number is a function of the left and right eigenvectors of the matrix.

We used LAPACK implementation of eigendecomposition algorithms. Some of the routines were modified to compute the eigenvalues in the domain of interest. Furthermore, the domain was specified as $\alpha + j\beta$, where $\alpha > -\epsilon$, $|\beta| < \epsilon$ and $j = \sqrt{-1}$. ϵ is a small positive constant used to account for the numerical errors. In particular, we make ϵ a function of the condition number of M_n or P_n for eigenvalue problems. To compute the inverse coordinate of the intersection point, the right eigenvector $\mathbf{V}$ corresponding to the eigenvalue s_0 is computed. Let

$$\mathbf{V} = [v_{1,1}\ v_{1,2}\ \ldots\ v_{1,m}\ v_{2,1}\ \ldots\ v_{2,m}\ \ldots\ v_{n,1}\ \ldots\ v_{n,m}]^T.$$

Analysis of the accuracy of eigenvector computation indicates that each term of the eigenvector has a similar bound on the absolute error of the computation. As a result we tend to use terms of maximum magnitude to minimize the error in the computation [MD92]. In this case we compute the entries of $\mathbf{v} = [v_1\ v_2\ \ldots\ v_m]^T$ as:

$$[v_1\ v_2\ \ldots\ v_m]^T = \begin{cases} [v_{1,1}\ v_{1,2}\ \ldots\ v_{1,m}]^T & s_0 \leq 1 \\ \frac{1}{(s_0)^n}[v_{n,1}\ v_{n,2}\ \ldots\ v_{n,m}]^T & s_0 > 1 \end{cases}$$

Given $\mathbf{v}$ the inverse coordinate, t_0, is computed using v_1, v_2 or v_{m-1}, v_m by making use of similar numerical properties.

The performance of the algorithm is largely governed by the eigendecomposition routines. Roughly $80 - 85\%$ of the time is spent in these routines. The eigenvalue algorithms compute all the eigenvalues of the given matrix. It is difficult to restrict them to computing eigenvalues in the domain of interest. The order of the matrix, say p, corresponds to the product of the degree of the two curves and the number of eigenvalues is equal to the order. The running time of the algorithm is a cubic function of p. The eigenvalue algorithms are iterative and have good convergence properties. It is a long observed fact that the algorithm requires two iterations per eigenvalue. As a result it is possible to bound the actual running time of the eigenvalue computation by $10p^3$ for most cases. Furthermore the eigendecomposition algorithms are backward stable. We have been to able accurately compute the intersections of curves of degree up to ten. In practice it is possible to obtain accurate solutions for matrices of order 100 or more. This is in contrast with computing roots of high degree univariate polynomials (which is an ill-conditioned problems) or using symbolic computation for determinant computation and finding the roots of the resulting polynomial expressed in Bernstein basis using subdivision and iteration (which is relatively expensive and has slow convergence).

4.2. Example of Curve Intersection

In this section we illustrate the algorithm by considering the intersection of two rational cubic Bézier curves. The example is taken from [Sed83]. The control points of two Bézier curves (as shown in Fig. II), expressed in homogeneous coordinates, are $(4,1,1), (5,6,2), (5,0,2), (6,4,1)$ and $(7,4,1), (1,2,2), (9,2,2), (3,4,1)$. Thus,

$$\mathbf{P}(t) = (\frac{x(t)}{w(t)}, \frac{y(t)}{w(t)}),$$

where

$$x(t) = 4(1-t)^3 + 30(1-t)^2t + 30(1-t)t^2 + 6t^3$$

$$y(t) = (1-t)^3 + 36(1-t)^2t + 4t^3$$

$$w(t) = (1-t)^3 + 6(1-t)^2t + 6(1-t)t^2 + t^3.$$

The implicit representation has a matrix determinant formulation given as

$$M = \begin{pmatrix} -114w + 30x - 6y & 30w - 6x - 6y & -10w + 3x - 2y \\ 30w - 6x - 6y & 1070w - 213x - 2y & 96w - 12x - 6y \\ -10w + 3x - 2y & 96w - 12x - 6y & -120w + 24x - 6y \end{pmatrix}.$$

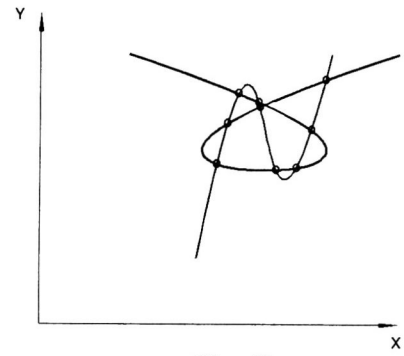

Fig. II
Intersection of rational cubic Bézier curves

The second parametrization, $\mathbf{Q}(u)$ is substituted into the matrix formulation (after a reparametrization of the form $s = \frac{u}{1-u}$). The resulting matrix polynomial has the form

$$\overline{M}(s) = \begin{pmatrix} -48 & -12 & -9 \\ -12 & 423 & 36 \\ -9 & 36 & -72 \end{pmatrix} s^3 + \begin{pmatrix} 864 & -216 & 78 \\ -216 & -5106 & -144 \\ 78 & -144 & 504 \end{pmatrix} s^2 + \begin{pmatrix} -576 & 72 & -66 \\ 72 & 5118 & 432 \\ -66 & 432 & -648 \end{pmatrix} s + \begin{pmatrix} 72 & -36 & 3 \\ -36 & -429 & -12 \\ 3 & -12 & 24 \end{pmatrix}.$$

The condition number of the leading matrix is 9.525.

Multiplying $M(s)$ with the inverse of the leading matrix and constructing the equivalent companion matrix results in

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1.48 & -1.07 & 0 & -11.92 & 4.58 & 0.40 & 17.8 & -8.24 & 0.40 \\ 0.13 & 0.94 & 0 & -0.53 & -11.92 & -.22 & 1.06 & 11.4 & -0.23 \\ -.07 & .44 & .33 & .30 & -.53 & -9.16 & -.61 & 4.74 & 6.83 \end{pmatrix}.$$

The eigendecomposition of C results in 9 points of intersection as shown in the following table.

eigenvectors we choose the elements v_1 or v_8 depending upon their relative magnitudes. The error bounds in the third column are obtained by using the condition number of the eigenvalues (as explained in [MD92]) and matrix norm as

$$E_i = \epsilon \parallel C \parallel \text{cond}_i, \tag{8}$$

where $\epsilon = 2.2204e - 16$ is the machine precision for 64 bit IEEE floating point arithmetic and cond_i is the condition number of the ith eigenvalue. As a result, the eigendecomposition algorithms compute the eigenvalues of C up to 12 digits of accuracy. The other sources of error arise from the computation of the entries of M, the matrix corresponding to the implicit representation, and inverting the leading matrix of the matrix polynomial $\overline{M}(s)$. In our case, this can account for inaccuracy of one digit (due to condition number of the matrix to be inverted). As a result, the intersection points are computed up to 11 digits of accuracy. Further accuracy can be obtained by using one or two iterations of Newton's method on the equations used for representing the problem of intersection. The output of the algorithm are the starting points for Newton's method. As a result the algorithm can be used to compute intersection points with high accuracy.

5. Higher Order Intersection

In the previous section, we presented an algorithm to compute the simple intersections of parametric curves. In this section we extend the analysis to higher order intersections.

According to Bezout's theorem two rational curves of degree m and n intersect in mn points (counted properly) [Wal50]. In our case, the preimages of these points correspond to the mn eigenvalues

Intersection Number	Eigenvalue s_0	Max. Error E_i	Parameter $u_0 = \frac{s_0}{1+s_0}$	Eigenvector's $\alpha = v_1 \vert v_8$	Eigenvector's $\beta = v_2 \vert v_9$	Parameter $t_0 = \frac{\beta}{\alpha+\beta}$	Point (X, Y)
1.	15.369	2.32e-14	0.9389	0.2173	0.0472	0.1785	(4.619, 3.412)
2.	11.802	2.85e-14	0.9219	0.6657	0.4432	0.3997	(4.911, 3.289)
3.	5.507	2.71e-14	0.8463	0.0703	1.000	0.9343	(5.688, 2.877)
4.	1.4654	1.27e-13	0.5944	0.1614	1.000	0.8610	(5.467, 2.321)
5.	0.5361	2.32e-14	0.3490	1.00	0.066	0.0622	(4.298, 2.378)
6.	0.1534	2.98e-14	0.133	1.00	0.1233	0.1099	(4.455, 2.971)
7.	0.0974	2.38e-14	0.0888	1.00	0.7277	0.4212	(4.931, 3.218)
8.	1.145	1.18e-13	0.534	1.00	0.4644	0.683	(4.174, 2.290)
9.	0.0382	1.14e-14	0.0369	0179	0.00032	0.9823	(5.901, 3.615)

Eigendecomposition and intersection points

The intersections points are computed using the relationship highlighted in (7). They are: In the columns corresponding to the components of the of C (5). Higher order intersections are the points having more than one preimage. In other words, eigenvalues of multiplicity greater than one corre-

spond to higher order intersection points. The *intersection multiplicity* of these points corresponds to the algebraic multiplicity of the corresponding eigenvalue. These intersections arise due to the tangential intersection of the two curves at the point of contact or the intersection point is a singular point on at least one of the curves. Some higher order intersections are highlighted in Fig. III. They are:

(a) Tangential intersection of two ellipses. The intersection multiplicity is 2.

(b) Second order intersection of a parabola with a curve having a loop.

(c) Intersection of an ellipse and a curve with a cusp. The intersection multiplicity is 2.

(d) Tangential intersection of a parabola with a loop. The intersection multiplicity is 3.

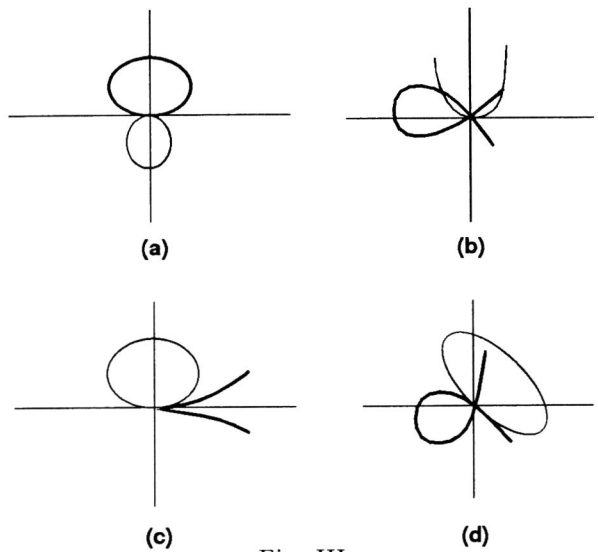

Fig. III
Higher order intersections

An intersection point, P, has multiplicity k, if a small perturbation in the coefficients of the curve (or the coefficients of the control polygon) results in k distinct intersection points in the neighborhood of P. Given two curves that intersect with multiplicity k at P, our algorithm computes the implicit representation and reduces the problem to an eigenvalue problem. The latter computation involves matrix inversion and multiplications using floating point operations. As a result, C in (5), corresponds to a slightly perturbed problem and k of its eigenvalues, say $\lambda_1, \lambda_2, \ldots, \lambda_k$ are very close to λ, the eigenvalue corresponding to P (assuming exact arithmetic). The eigendecomposition algorithms use floating point arithmetic and the computed eigenvalues correspond to $\lambda_1^{'}, \lambda_2^{'}, \ldots, \lambda_k^{'}$. However the problem of computing eigenvalues of multiplicity greater than one can be ill–conditioned [GL89, Wil65]. As a result $\lambda_i^{'}$ may agree with λ up to a few digits of accuracy. This can be determined from the condition number of $\lambda_i^{'}$. In such cases the mean of $\lambda_i^{'}$'s given by

$$\lambda^{'} = \frac{\lambda_1^{'} + \lambda_2^{'} + \ldots + \lambda_k^{'}}{k}$$

is much better conditioned. This can be verified by computing the condition number of a cluster of eigenvalues [BDM89]. As a result $\lambda^{'}$ is very close to λ, the eigenvalue of multiplicity k. The number k corresponds to the number of ill-conditioned eigenvalues, $\lambda_i^{'}$, located in the small neighborhood of each other. We illustrate this technique using the following example.

Example: Consider the intersection of cubic curve, $\mathbf{P}(t) = (x(t), y(t)) = (t^2 - 1, t^3 - t)$ with the parabola $\mathbf{Q}(u) = (\overline{x}(u), \overline{y}(u)) = (u^2 + u, u^2 - u)$ (as shown in Fig. IV). The cubic curve has a loop at the origin. We are interesting in computing all the intersection points corresponding to the domain $t \times u = [-2, 2] \times [-1, 1]$.

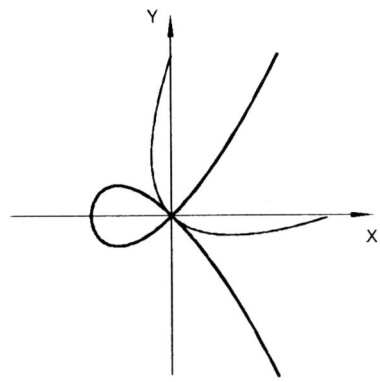

Fig. V
Higher order intersection of a cubic curve and a parabola

The implicit representation of $\mathbf{P}(t)$ is a matrix determinant of the form

$$M = \begin{pmatrix} -w - x & -y & w + x \\ -y & x & 0 \\ w + x & 0 & -w \end{pmatrix}.$$

After substituting and reducing the problem to an eigenvalue problem we obtain a 6×6 matrix

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & -2 & -1 & 0 \\ -1 & 0 & 1 & -2 & -2 & -1 \end{pmatrix}.$$

The relevant eigenvalues of this matrix and their condition numbers are:

Intersection Number	Eigenvalue $\lambda_i^{'}$	Cond. Number $Cond_i$
1.	0.0	1.2171
2.	-0.00000001296346	4.86e09
3.	0.00000001296346	4.86e09

Eigenvalues corresponding to higher order intersections

Thus, the second and third eigenvalues have a high condition number. Taking their average we obtain $\lambda^{'} = 0.0$, and it has a low condition number. As a result, we conclude that $\lambda^{'} = 0.0$ is an eigenvalue of multiplicity 3 and the curves have a triple intersection at $\mathbf{Q}(0.0) = (0.0, 0.0)$.

Q.E.D.

The procedure highlighted above is used for computing the intersection multiplicity of the point. However, the intersection can arise from tangential intersection, singular points or their combinations. The rest of the analysis deals with determining the nature of intersection.

Given an eigenvalue, λ, of algebraic multiplicity k, its geometric multiplicity corresponds to the dimension of the kernel of $(C - \lambda I)$. After accurately computing the algebraic multiplicity of the eigenvalue, we compute its geometric multiplicity. Furthermore, the algorithm computes a basis of the vectors spanning the kernel. Depending upon the nature of intersection the geometric multiplicity is less than or equal to algebraic multiplicity. The exact relationship between the multiplicities and nature of intersection has been described in [MD92]. Here we highlight the relationship for the examples in Fig. III and IV. We assume that the curves drawn in dark font are being implicitized. The curves highlighted in light font are substituted into the implicit representation. The choice of curve for implicitization has an impact on the geometric multiplicities of the matrix, although the algebraic multiplicities are unaffected [MD92].

Example Figure	Algebraic Multiplicity	Geometric Multiplicity
III(a)	2	1
III(b)	2	2
III(c)	2	2
III(d)	3	2
IV	3	2

Algebraic and geometric multiplicities of eigenvalues corresponding to Figs. III and IV

In case the geometric multiplicity is 1, the computation of the preimage corresponds exactly to the procedure described in the previous section. If the geometric multiplicity is greater than 1, the algorithm for preimage computation involves equation solving. Let us illustrate for Fig. V.

In the example corresponding to Fig. V, the intersection multiplicity is 3. If we implicitize the parabola and substitute the cubic curve into the implicit form, the eigenvalue corresponding to the origin has algebraic multiplicity 3 and geometric multiplicity 1. This is due to the fact that the parabola is intersecting the cubic curve tangentially at the loop, corresponding to the origin. The fact that the intersection point is a loop implies that $\mathbf{P}(t)$ has two distinct preimages $t_1 = 1$ and $t_2 = -1$. As a result both the vectors $\mathbf{v}_1 = [1\ t_1\ t_1^2]^T$ and $\mathbf{v}_2 = [1\ t_2\ t_2^2]^T$ lie in the kernel of M after we substitute $x = 0, y = 0, w = 1$. Since these vectors are linearly independent they constitute the basis of the eigenvectors corresponding to $\lambda = 0$. However, the eigendecomposition algorithm can return any two linearly independent vectors of the form $\mathbf{V}_1 = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$ and $\mathbf{V}_2 = \beta_1\mathbf{v}_1 + \beta_2\mathbf{v}_2$, where α_i's and β_j's are scalars. The rest of the algorithm involves computing $\mathbf{v}_1$ and $\mathbf{v}_2$ from $\mathbf{V}_1$ and $\mathbf{V}_2$. This corresponds to computing the scalars and can be achieved by equation solving [MD92].

6. Conclusion

In this paper we have highlighted a new technique for computing the intersection of parametric and algebraic curves. The algorithm involves use of resultants to represent the implicit representation of a parametric curve as a matrix determinant. The intersection problem is being reduced to an eigenvalue problem. The algorithm is very robust and can accurately compute the intersection points. There is a nice relationship between the algebraic and geometric multiplicities of the eigenvalues and the order of intersection. We used this relationship in accurately computing such intersections. Efficient implementations of eigenvalue routines are available as part of linear algebra packages and we used them in our implementations.

The approach highlighted here is also useful for intersecting curves and surfaces. In particular, the implicit representation of a parametric surface can be represented as a matrix determinant [MC91]. The parametric space curve is substituted into the implicit formulation and the problem can therefore, be reduced to an eigenvalue problem. This can be directly used for *ray tracing* parametric patches, as a ray corresponds to degree one parametric curve.

Acknowledgement: Dinesh Manocha is supported in part by IBM Graduate Fellowship, David and Lu-

cile Packard Fellowship and National Science Foundation grant ASC-9005933. James Demmel is supported in part by National Science Foundation grants DCR-8552474, ASC-8715728 and ASC-9005933.

References

[ABB+90] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. Lapack: A portable linear algebra library. Computer Science Technical Report CS-90-105, University of Tennessee, 1990.

[BBB87] R.H. Bartels, J.C. Beatty, and B.A. Barsky. *An Introduction to Splines for use in Computer Graphics & Geometric Modeling.* Morgan Kaufmann, San Mateo, California, 1987.

[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).

[EC90] G. Elber and E. Cohen. Hidden curve removal for free form surfaces. *Computer Graphics*, 24(4):95–104, 1990.

[FR87] R.T. Farouki and V.T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.

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

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

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

[KM83] P.A. Koparkar and S. P. Mudur. A new class of algorithms for the processing of parametric curves. *Computer-Aided Design*, 15(1):41–45, 1983.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):150–159, 1980.

[MC91] D. Manocha and J.F. Canny. A new approach for surface intersection. In *First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 209–220, 1991. Revised version to appear in *International Journal of Computational Geometry and Applications.*

[MD92] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves. Technical report, Computer Science Division, UC Berkeley, 1992.

[MM89] R.P. Markot and R. L. Magedson. Solutions of tangential surface and curve intersections. *Computer-Aided Design*, 21(7):421–427, 1989.

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

[Sed83] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces.* PhD thesis, Purdue University, 1983.

[Sed89] T.W. Sederberg. Algorithms for algebraic curve intersection. *Computer-Aided Design*, 21(9):547–555, 1989.

[SP86] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

[SWZ89] T.W. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[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.