Chapter 10

Algebraic Algorithms[1]

Angel Díaz
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180-3590;
Inter-Net: diaza@cs.rpi.edu

Erich Kaltofen
Department of Mathematics, North Carolina State University
Raleigh, NC 27695-8205
Inter-Net: kaltofen@eos.ncsu.edu

Victor Pan
Mathematics and Computer Science Department, Lehman College
City University of New York, Bronx, NY 10468
Inter-Net: vpan@lcvax.lehman.cuny.edu

[1] This material is based on work supported in part by the National Science Foundation under Grants No. CCR-9319776 (first and second author) and No. CCR-9020690 (third author), by GTE under a Graduate Computer Science Fellowship (first author), and by PSC CUNY Awards 665301 and 666327 (third author). Part of this work was done while the second author was at the Department of Computer Science at Rensselaer Polytechnic Insititute in Troy, New York.

Appears in The Computer Science and Engineering Handbook, A. B. Tucker editor, CRC Press, Boca Raton, Florida, pp. 226–248 (1997).

Contents

1 Introduction **3**

2 Matrix Computations and Approximation of Polynomial Zeros **3**
2.1 Products of Vectors and Matrices, Convolution of Vectors . . . 3
2.2 Some Computations Related to Matrix Multiplication . . . 4
2.3 Gaussian Elimination Algorithm . . . 5
2.4 Singular Linear Systems of Equations . . . 5
2.5 Sparse Linear Systems (Including Banded Systems), Direct and Iterative Solution Algorithms 6
2.6 Dense and Structured Matrices and Linear Systems . . . 6
2.7 Parallel Matrix Computations . . . 6
2.8 Rational Matrix Computations, Computations in Finite Fields and Semirings . . . 7
2.9 Matrix Eigenvalues and Singular Values Problems . . . 7
2.10 Approximating Polynomial Zeros . . . 9
2.11 FFT and Fast Polynomial Arithmetic . . . 10

3 Systems of Nonlinear Equations and Other Applications **10**
3.1 Resultant Methods . . . 11
3.2 Gröbner Bases . . . 12

4 Polynomial Factorization **14**
4.1 Polynomials in a single variable over a finite field . . . 15
4.2 Polynomials in a single variable over fields of characteristic zero . . . 16
4.3 Polynomials in two variables . . . 17
4.4 Polynomials in many variables . . . 18

5 Further Information **18**

1 Introduction

The title's subject is the algorithmic approach to algebra: arithmetic with numbers, polynomials, matrices, differential polynomials, such as $y'' + (1/2 + x^4/4)y$, truncated series, and algebraic sets, i.e., quantified expressions such as $\exists x \in \mathbb{R} : x^4 + p \cdot x + q = 0$, which describes a subset of the two-dimensional space with coordinates p and q for which the given quartic equation has a real root. Algorithms that manipulate such objects are the backbone of modern symbolic mathematics software such as the Maple and Mathematica systems, to name but two among many useful systems. This chapter restricts itself to algorithms in four areas: linear matrix algebra, root finding of univariate polynomials, solution of systems of nonlinear algebraic equations, and polynomial factorization.

2 Matrix Computations and Approximation of Polynomial Zeros

This section covers several major algebraic and numerical problems of scientific and engineering computing that are usually solved numerically, with rounding-off or chopping the input and computed values to a fixed number of bits that fit the computer precision (sections 2 and 3 are devoted to some fundamental infinite precision symbolic computations, and in section 1.8 we comment on the infinite precision techniques for some matrix computations). We also study approximation of polynomial zeros, which is an important, fundamental, and very popular subject, too. In our presentation, we will very briefly list the major subtopics of our huge subject and will give some pointers to the bibliography. We will include brief coverage of the topics of the algorithm design and analysis, regarding the complexity of matrix computation and of approximating polynomial zeros. The reader may find further material on these subjects in the survey articles [Pan84a], [Pan91], [Pan92a], [Pan95b] and in the book [BP94], [BPar].

2.1 Products of Vectors and Matrices, Convolution of Vectors

An $m \times n$ matrix $A = (\ a_{i,j}\ ,\ i = 0, 1, \ldots, m-1;\ j = 0, 1, \ldots, n-1\ )$ is a 2-dimensional array, whose (i, j)-entry is $(A)_{i,j} = a_{i,j}$. A is a column vector of dimension m if $n = 1$ and is a row vector of dimension n if $m = 1$. Transposition, hereafter, indicated by the superscript T, transforms a row vector $\vec{v}^{\ T} = [\ v_0, \cdots, v_{n-1}\ ]$ into a column vector $\vec{v} = [\ v_0, \cdots, v_{n-1}\ ]^T$.

For 2 vectors, $\vec{u}^{\ T} = (\ u_0, \cdots, u_{m-1}\ )$ and $\vec{v}^{\ T} = (\ v_0, \cdots, v_{n-1}\ )^T$, their *outer product* is an $m \times n$ matrix,

$$W = \vec{u}\ \vec{v}^{\ T} = [\ w_{i,j}\ ,\ i = 0, \cdots, m-1;\ j = 0, \cdots, n-1\ ]\ ,$$

where $w_{i,j} = u_i\ v_j$, for all i and j, and their *convolution* vector is said to equal

$$\vec{w} = \vec{u} \circ \vec{v} = (\ w_0, \cdots, w_{m+n-2}\ )^T, \quad w_k = \sum_{i=0}^{k} u_i v_{k-i}\ ,$$

where $u_i = v_j = 0$, for $i \geq m$, $j \geq n$; in fact, $\vec{w}$ is the coefficient vector of the product of 2 polynomials,

$$u(x) = \sum_{i=0}^{m-1} u_i x^i \quad \text{and} \quad v(x) = \sum_{i=0}^{n-1} v_i x^i\ ,$$

having coefficient vectors $\vec{u}$ and $\vec{v}$, respectively.

If $m = n$, the scalar value

$$\vec{v}^{\ T}\ \vec{u} = \vec{u}^{\ T}\ \vec{v} = u_0 v_0 + u_1 v_1 + \cdots + u_{n-1} v_{n-1} = \sum_{i=0}^{n-1} u_i v_i$$

is called the *inner* (*dot*, or *scalar*) *product* of $\vec{u}$ and $\vec{v}$.

The straightforward algorithms compute the inner and outer products of $\vec{u}$ and $\vec{v}$ and their convolution vector by using $2n - 1$, mn, and $mn + (m-1)(n-1) = 2mn - m - n + 1$ arithmetic operations (hereafter, referred to as ops), respectively.

These upper bounds on the numbers of ops for computing the inner and outer products are sharp, that is, cannot be decreased, for the general pair of the input vectors $\vec{u}$ and $\vec{v}$, whereas (see, e.g. [BP94]) one may apply the *Fast Fourier Transform* (FFT) in order to compute the convolution vector $\vec{u} \circ \vec{v}$ much faster, for larger m and n; namely, it suffices to use $4.5K \log K + 2K$ ops, for $K = 2^k$, $k = \lceil \log(m+n+1) \rceil$. (Here and hereafter, all logarithms are binary unless specified otherwise.)

If $A = (a_{i,j})$ and $B = (b_{j,k})$ are $m \times n$ and $n \times p$ matrices, respectively, and $\vec{v} = (v_k)$ is a p-dimensional vector, then the straightforward algorithms compute the vector

$$\vec{w} = B\vec{v} = (\ w_0, \cdots, w_{n-1}\ )^T, \qquad w_i = \sum_{j=0}^{p-1} b_{i,j} v_j, \quad i = 0, \cdots, n-1\ ,$$

by using $(2p-1)n$ ops (sharp bound), and compute the *matrix product*

$$AB = (\ w_{i,k}\ ,\ i = 0, \cdots, m-1;\ k = 0, \cdots, p-1\ )$$

by using $2mnp - mp$ ops, which is $2n^3 - n^2$ if $m = n = p$. The latter upper bound is not sharp: the subroutines for $n \times n$ matrix multiplication on some modern computers, such as CRAY and Connection Machines, rely on algorithms using $O(n^{2.81})$ ops, and some nonpractical algorithms involve $O(n^{2.376})$ ops [BP94], [GL89].

In the special case, where all the input entries and components are bounded integers having short binary representation, each of the above operations with vectors and matrices can be reduced to a single multiplication of 2 longer integers, by means of the techniques of *binary segmentation* (cf. [Pan84b], sect. 40, [Pan91], [Pan92b], or [BP94], Examples 3.9.1–3.9.3).

For an $n \times n$ matrix B and an n-dimensional vector $\vec{v}$, one may compute the vectors $B^i\ \vec{v}$, $\quad i = 1, 2, \cdots, k-1$, which define *Krylov sequence* or *Krylov matrix*

$$[\ B^i\ \vec{v},\ i = 0, 1, \cdots, k-1\ ]\ ,$$

used as a basis of several computations. The straightforward algorithm takes on $(2n-1)nk$ ops, which is order n^3 if k is of order n. An alternative algorithm first computes the matrix powers

$$B^2, B^4, B^8, \cdots, B^{2^s}\ , \qquad s = \lceil \log k\ \rceil - 1\ ,$$

and then the products of $n \times n$ matrices B^{2^i} by $n \times 2^i$ matrices, for $i = 0, 1, \cdots, s$:

$$\begin{array}{ll} B & v\ , \\ B^2 & (\ v,\ Bv\ ) = (\ B^2v,\ B^3v\ )\ , \\ B^4 & (\ v,\ Bv,\ B^2v,\ B^3v\ ) = (\ B^4v,\ B^5v,\ B^6v,\ B^7v\ )\ , \\ \vdots & \end{array}$$

The last step completes the evaluation of the Krylov sequence, which amounts to $2s$ matrix multiplications, for $k = n$, and, therefore, can be performed (in theory) in $O(n^{2.376} \log k)$ ops.

2.2 Some Computations Related to Matrix Multiplication

Several fundamental matrix computations can be ultimately reduced to relatively few [that is, to a constant number, or, say, to $O(\log n)$] $n \times n$ matrix multiplications. These computations include the evaluation of *det* A, the *determinant* of an $n \times n$ matrix A; of its *inverse* A^{-1} (where A is nonsingular, that is, where *det* $A \neq 0$); of the coefficients of its *characteristic polynomial*, $c_A(x) = det(\ xI - A\ )$, x denoting a scalar variable and I being the $n \times n$ identity matrix, which has ones on its diagonal and zeros elsewhere; of its *minimal polynomial*, $m_A(x)$; of its *rank*, rank A; of the solution vector $\vec{x} = A^{-1}\ \vec{v}$ to a nonsingular *linear system of equations*, $A\ \vec{x} = \vec{v}$; of various *orthogonal* and *triangular factorizations* of A, and of a submatrix of A having the maximal rank, as well as some fundamental computations with singular matrices. Consequently, all these operations can be performed by using (theoretically) $O(n^{2.376})$ ops (cf. [BP94], chapter 2). The idea is to represent the input matrix A as a block matrix and, operating with its blocks (rather than with

its entries), to apply fast matrix multiplication algorithms. In practice, due to various other considerations (accounting, in particular, for the overhead constants hidden in the "O" notation, for the memory space requirements, and particularly, for numerical stability problems), these computations are based either on the straightforward algorithm for matrix multiplication or on other methods allowing order n^3 arithmetic operations (cf. [GL89]). Many block matrix algorithms supporting the (nonpractical) estimate $O(n^{2.376})$, however, become practically important for parallel computations (see section 1.7).

In the next 6 sections, we will more closely consider the solution of a linear system of equations, $A\ \vec{v} = \vec{b}$, which is the most frequent operation in practice of scientific and engineering computing and is highly important theoretically. We will partition the known solution methods depending on whether the coefficient matrix A is *dense and unstructured*, *sparse*, or *dense and structured*.

2.3 Gaussian Elimination Algorithm

The solution of a nonsingular linear system $A\ \vec{x} = \vec{v}$ uses only about n^2 ops if the system is lower (or upper) triangular, that is, if all subdiagonal (or superdiagonal) entries of A vanish. For example (cf. [Pan92b]), let $n = 3$,

$$\begin{aligned} x_1 + 2x_2 - \ x_3 &= \ \ 3\ , \\ -2x_2 - 2x_3 &= -10\ , \\ -6x_3 &= -18\ . \end{aligned}$$

Compute $x_3 = 3$ from the last equation, substitute into the previous ones, and arrive at a triangular system of $n-1 = 2$ equations. In $n-1$ (in our case, in 2) such recursive substitution steps, we compute the solution.

The triangular case is itself important; furthermore, every nonsingular linear system is reduced to 2 triangular ones by means of *forward elimination* of the variables, which essentially amounts to computing the PLU-factorization of the input matrix A, that is, to computing 2 lower triangular matrices L and U^T (where L has unit values on its diagonal) and a permutation matrix P such that $A = PLU$. (A permutation matrix P is filled with zeros and ones and has exactly one nonzero entry in each row and in each column; in particular, this implies that $P^T = P^{-1}$. $P\ \vec{u}$ has the same components as $\vec{u}$ but written in a distinct (fixed) order, for any vector $\vec{u}$.) As soon as the latter factorization is available, we may compute $\vec{x} = A^{-1}\ \vec{v}$ by solving 2 triangular systems, that is, at first, $L\ \vec{y} = P^T\ \vec{v}$, in $\vec{y}$, and then $U\ \vec{x} = \vec{y}$, in $\vec{x}$. Computing the factorization (elimination stage) is more costly than the subsequent *back substitution stage*, the latter involving about $2n^2$ ops. The Gaussian classical algorithm for elimination requires about $2n^3/3$ ops, not counting some comparisons, generally required in order to ensure appropriate *pivoting*, also called *elimination ordering*. Pivoting enables us to avoid divisions by small values, which could have caused numerical stability problems. Theoretically, one may employ fast matrix multiplication and compute the matrices P, L, and U in $O(n^{2.376})$ ops [AHU74] (and then compute the vectors $\vec{y}$ and $\vec{x}$ in $O(n^2)$ ops). Pivoting can be dropped for some important classes of linear systems, notably, for *positive definite* and for *diagonally dominant* systems ([GL89], [Pan91], [Pan92b], or [BP94]).

We refer the reader to [GL89], pp.82–83, or [Pan92b], p.794, on sensitivity of the solution to the input and round-off errors in numerical computing. The output errors grow with the *condition number* of A, represented by $||A||\ ||A^{-1}||$ for an appropriate matrix norm or by the ratio of maximum and minimum singular values of A. Except for ill-conditioned linear systems $A\ \vec{x} = \vec{v}$, for which the condition number of A is very large, a rough initial approximation to the solution can be rapidly refined (cf. [GL89]) via the *iterative improvement algorithm*, as soon as we know P and rough approximations to the matrices L and U of the PLU factorization of A. Then b correct bits of each output value can be computed in $(b+n)n^2$ ops as $b \to \infty$.

2.4 Singular Linear Systems of Equations

If the matrix A is singular (in particular, if A is rectangular), then the linear system $A\ \vec{x} = \vec{v}$ is either overdetermined, that is, has no solution, or underdetermined, that is, has infinitely many solution vectors. All of them can be represented as $\{\ \vec{x_0} + \vec{y}\ \}$, where $\vec{x_0}$ is a fixed solution vector and $\vec{y}$ is a vector from the *null space* of A, $\{\ \vec{y}\ :\ A\ \vec{y} = \vec{0}\ \}$, that is, $\vec{y}$ is a solution of the homogeneous linear system $A\ \vec{y} = \vec{0}$. (The null space of an $n \times n$ matrix A is a linear space of the dimension n–rank A.) A vector $\vec{x_0}$ and a basis for

the null-space of A can be computed by using $O(n^{2.376})$ ops if A is an $n \times n$ matrix or by using $O(mn^{1.736})$ ops if A is an $m \times n$ or $n \times m$ matrix and if $m \geq n$ (cf. [BP94]).

For an overdetermined linear system $A\ \vec{x} = \vec{v}$, having no solution, one may compute a vector $\vec{x}$ minimizing the norm of the residual vector, $||\ \vec{v} - A\ \vec{x}\ ||$. It is most customary to minimize the Euclidean norm,

$$||\vec{u}|| = (\ \sum_i |u_i|^2\ )^{1/2}\ , \qquad \vec{u} = \vec{v} - A\ \vec{x} = (u_i)\ .$$

This defines a least-squares solution, which is relatively easy to compute both practically and theoretically ($O(n^{2.376})$ ops suffice in theory) (cf. [BP94], [GL89]).

2.5 Sparse Linear Systems (Including Banded Systems), Direct and Iterative Solution Algorithms

A matrix is sparse if it is filled mostly with zeros, say, if its all nonzero entries lie on 3 or 5 of its diagonals. In many important applications, in particular, to solving partial and ordinary differential equations (PDEs and ODEs), one has to solve sparse linear systems, whose matrix is sparse and where, moreover, the disposition of its nonzero entries has a certain structure. Then, memory space and computation time can be dramatically decreased (say, from order n^2 to order $n \log n$ words of memory and from n^3 to $n^{3/2}$ or $n \log n$ ops) by using some special data structures and special solution methods. The methods are either direct, that is, are modifications of Gaussian elimination with some special policies of elimination ordering that preserve sparsity during the computation (notably, *Markowitz rule* and *nested dissection*, [GL81], [GT87], [LRT79], [Pan93]), or various iterative algorithms. The latter algorithms rely either on computing Krylov sequences [Saa95] or on multilevel or multigrid techniques [McC87], [PR92], specialized for solving linear systems that arise from discretization of PDEs. An important particular class of sparse linear systems is formed by *banded linear systems* with $n \times n$ coefficient matrices $A = (a_{i,j})$ where $a_{i,j} = 0$ if $i - j > g$ or $j - i > h$, for $g + h$ being much less than n. For banded linear systems, the nested dissection methods are known under the name of *block cyclic reduction* methods and are highly effective, but [PSA95] gives some alternative algorithms , too. Some special techniques for computation of Krylov sequences for sparse and other special matrices A can be found in [Pan95a]; according to these techniques, Krylov sequence is recovered from the solution of the associated linear system $(I - A)\ \vec{x} = \vec{v}$, which is solved fast in the case of a special matrix A.

2.6 Dense and Structured Matrices and Linear Systems

Many dense $n \times n$ matrices are defined by $O(n)$, say, by less than $2n$, parameters and can be multiplied by a vector by using $O(n \log n)$ or $O(n \log^2 n)$ ops. Such matrices arise in numerous application (to signal and image processing, coding, algebraic computation, PDEs, integral equations, particle simulation, Markov chains, and many others). An important example is given by $n \times n$ *Toeplitz matrices* $T = (t_{i,j})$, $t_{i,j} = t_{i+1,j+1}$ for $i,\ j = 0, 1, \cdots, n-1$. Such a matrix can be represented by $2n - 1$ entries of its first row and first column or by $2n - 1$ entries of its first and last columns. The product $T\ \vec{v}$ is defined by vector convolution, and its computation uses $O(n \log n)$ ops. Other major examples are given by *Hankel matrices* (obtained by reflecting the row or column sets of Toeplitz matrices), *circulant* (which are a subclass of Toeplitz matrices), *Bezout*, *Sylvester*, *Vandermonde*, and *Cauchy* matrices. The known solution algorithms for linear systems with such dense structured coefficient matrices use from order $n \log n$ to order $n \log^2 n$ ops. These properties and algorithms are extended via associating some linear operators of displacement and scaling to some more general classes of matrices and linear systems. We refer the reader to [BP94] for many details and further bibliography.

2.7 Parallel Matrix Computations

Algorithms for matrix multiplication are particularly suitable for parallel implementation; one may exploit natural association of processors to rows and/ or columns of matrices or to their blocks, particularly, in the implementation of matrix multiplication on loosely coupled multiprocessors (cf. [GL89], [Qui94]). This motivated particular attention to and rapid progress in devising effective parallel algorithms for block matrix computations. The complexity of parallel computations is usually represented by the computational and

communication time and the number of processors involved; decreasing all these parameters, we face a trade-off; the product of time and processor bounds (called potential work of parallel algorithms) cannot usually be made substantially smaller than the sequential time bound for the solution. This follows because, according to a variant of *Brent's scheduling principle*, a single processor can simulate the work of s processors in time $O(s)$. The usual goal of designing a parallel algorithm is in decreasing its parallel time bound (ideally, to a constant, logarithmic or polylogarithmic level, relative to n) and keeping its work bound at the level of the record sequential time bound for the same computational problem (within constant, logarithmic, or at worst polylog factors). This goal has been easily achieved for matrix and vector multiplications, but turned out to be nontrivial for linear system solving, inversion, and some other related computational problems. The recent solution for general matrices [KP91, KP92] relies on computation of a Krylov sequence and the coefficients of the minimum polynomial of a matrix, by using randomization and auxiliary computations with structured matrices (see the details in [BP94]).

2.8 Rational Matrix Computations, Computations in Finite Fields and Semirings

Rational algebraic computations with matrices are performed for a rational input given with no errors, and the computations are also performed with no errors. The precision of computing can be bounded by reducing the computations modulo one or several fixed primes or prime powers. At the end, the exact output values $z = p/q$ are recovered from $z \ mod \ M$ (if M is sufficiently large relative to p and q) by using the continued fraction approximation algorithm, which is the Euclidean algorithm applied to integers (cf. [Pan91], [Pan92a], and [BP94], sect. 3 of ch. 3). If the output z is known to be an integer lying between $-m$ and m and if $M > 2m$, then z is recovered from $z \bmod M$ as follows:

$$z = \begin{cases} z \bmod M & \text{if} \quad z \bmod M < m \\ -M + z \bmod M & \text{otherwise} . \end{cases}$$

The reduction modulo a prime p may turn a nonsingular matrix A and a nonsingular linear system $A\vec{x} = \vec{v}$ into singular ones, but this is proved to occur only with a low probability for a random choice of the prime p in a fixed sufficiently large interval (see [BP94], sect. 9 of ch. 4). To compute the output values z modulo M for a large M, one may first compute them modulo several relatively prime integers $m_1, m_2, \cdots, m_k$ having no common divisors and such that $m_1 m_2 \cdots m_k > M$ and then easily recover $z \bmod M$ by means of the Chinese remainder algorithm. For matrix and polynomial computations, there is an effective alternative technique of *p-adic* (*Newton-Hensel*) *lifting* (cf. [BP94], sect. 3 of ch. 3), which is particularly powerful for computations with dense structured matrices, since it preserves the structure of a matrix. We refer the reader to [Bar68] and [GCL92] on some special techniques, which enable one to control the growth of all intermediate values computed in the process of performing rational Gaussian elimination, with no round-off and no reduction modulo an integer.

[GM84] and [Pan93] describe some applications of matrix computations on semirings(with no divisions and subtractions allowed) to graph and combinatorial computations.

2.9 Matrix Eigenvalues and Singular Values Problems

The matrix eigenvalue problem is one of the major problems of matrix computation: given an $n \times n$ matrix A, one seeks a $k \times k$ diagonal matrix Λ and an $n \times k$ matrix V of full rank k such that

$$A \ V = \Lambda V \ . \tag{9.1}$$

The diagonal entries of Λ are called the *eigenvalues* of A; the entry (i, i) of Λ is associated with the i-th column of V, called an *eigenvector* of A. The eigenvalues of an $n \times n$ matrix A coincide with the zeros of the characteristic polynomial

$$c_A(x) = \det(\ xI - A\ ) \ .$$

If this polynomial has n distinct zeros, then $k = n$, and V of (9.1) is a nonsingular $n \times n$ matrix. The matrix $A = I + Z$, $Z = (z_{i,j})$, $z_{i,j} = 0$ unless $j = i + 1$, $z_{i,i+1} = 1$, is an example of a matrix for which $k = 1$, so that the matrix V degenerates to a vector.

In principle, one may compute the coefficients of $c_A(x)$, the characteristic polynomial of A, and then approximate its zeros (see the next section) in order to approximate the eigenvalues of A. Given the eigenvalues, the corresponding eigenvectors can be recovered by means of the inverse power iteration [GL89], [Wil65]. Practically, the computation of the eigenvalues via the computation of the coefficients of $c_A(x)$ is not recommended, due to arising numerical stability problems [Wil65], and most frequently, the eigenvalues and eigenvectors of a general (unsymmetric) matrix are approximated by means of the QR *algorithm* [Wil65], [Wat82], [GL89]. Before application of this algorithm, the matrix A is simplified by transforming it into the more special (*Hessenberg*) *form*, H, by a *similarity transformation*,

$$H = UAU^H \ , \tag{9.2}$$

where $U = (u_{i,j})$ is a unitary matrix, $U^H U = I$, $U^H = (\overline{u}_{j,i})$ is the Hermitian transpose of U, $\overline{z}$ denoting the complex conjugate of z; $U^H = U^T$ if U is a real matrix [GL89]. Similarity transformation into Hessenberg form is one of examples of *rational transformations* of a matrix into special *canonical forms*, of which transformations into *Smith* and *Hermite forms* are two other most important representatives [KKS90], [GCL92], and [Gie95].

In practice, the eigenvalue problem is very frequently symmetric, that is, arises for a real symmetric matrix A, for which

$$A^T = (a_{j,i}) = A = (a_{i,j}) \ ,$$

or for complex Hermitian matrices A, for which

$$A^H = (\overline{a}_{j,i}) = A = (a_{i,j}) \ .$$

For real symmetric or Hermitian matrices A, the eigenvalue problem (called symmetric) is treated much more easily than in the unsymmetric case. In particular, in the symmetric case, we have $k = n$, that is, the matrix V of (10.1) is a nonsingular $n \times n$ matrix, and moreover, all the eigenvalues of A are real and little sensitive to small input perturbations of A (according to the Courant-Fisher minimization criterion [Par80], [GL89]).

Furthermore, similarity transformation of A to the Hessenberg form gives much stronger results in the symmetric case: the original problem is reduced to one for a symmetric tridiagonal matrix H of (9.2) (this can be achieved via Lanczos algorithm, cf. [GL89] or [BP94], sect. 3 of ch. 2). For such a matrix H, application of the QR algorithm is dramatically simplified; moreover, two competitive algorithms are also widely used, that is, the *bisection* [Par80] (a slightly slower but very robust algorithm) and the *divide-and-conquer* method [Cup81], [GL89]. The latter method has a modification [BP91] that only uses $O(n \log^2 n(\log n + \log^2 b))$ arithmetic operations in order to compute all the eigenvalues of an $n \times n$ symmetric tridiagonal matrix A within the output error bound $2^{-b}||A||$, where $||A|| \le n \max |a_{i,j}|$.

The eigenvalue problem has a generalization, where generalized eigenvalues and eigenvectors, for a pair A, B of matrices, are sought, such that

$$A \ V = B \ \Lambda \ V$$

(the solution algorithm should proceed without computing the matrix $B^{-1}A$, so as to avoid numerical stability problems).

In another highly important extension of the symmetric eigenvalue problem, one seeks a singular value decomposition (SVD) of a (generally unsymmetric and, possibly, rectangular) matrix A: $A = U\Sigma V^T$, where U and V are unitary matrices, $U^H U = V^H V = I$, and Σ is a diagonal (generally rectangular) matrix, filled with zeros, except for its diagonal, filled with (positive) singular values of A and, possibly, with zeros. The SVD is widely used in the study of numerical stability of matrix computations and in numerical treatment of singular and ill-conditioned (close to singular) matrices. An alternative tool is orthogonal (QR) factorization of a matrix, which is not as refined as SVD but is a little easier to compute [GL89]. The squares of the singular values of A equal the eigenvalues of the Hermitian (or real symmetric) matrix $A^H A$, and the SVD of A can be also easily recovered from the eigenvalue decomposition of the Hermitian matrix

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix} \ ,$$

but more popular are some effective direct methods for the computation of the SVD [GL89].

2.10 Approximating Polynomial Zeros

Solution of an n-th degree polynomial equation,

$$p(x) = \sum_{i=0}^{n} p_i \; x^i = 0 \; , \qquad p_n \neq 0$$

(where one may assume that $p_{n-1} = 0$; this can be ensured via shifting the variable x), is a classical problem that has greatly influenced the development of mathematics throughout the centuries [Pan95b]. The problem remains highly important for the theory and practice of present day computing, and dozens of new algorithms for its approximate solution appear every year. Among the existent implementations of such algorithms, the practical heuristic champions in efficiency (in terms of computer time and memory space used, according to the results of many experiments) are various modifications of *Newton's iteration*, $z(i+1) = z(i) - a(i)p(z(i))/p'(z(i))$, $a(i)$ being the step-size parameter [Mad73], *Laguerre's method* [HPR77, Fos81], and the randomized *Jenkins-Traub algorithm* [JT70] (all three for approximating a single zero z of $p(x)$), which can be extended to approximating other zeros by means of deflation of the input polynomial via its numerical division by $x - z$. For simultaneous approximation of all the zeros of $p(x)$ one may apply the Durand-Kerner algorithm, which is defined by the following recurrence:

$$z_j(i+1) = \frac{z_j(i) - p((z_j(i))}{z_j(i) - z_k(i)} \; , \quad j = 1, \cdots, n, \quad i = 1, 2, \cdots . \tag{10.1}$$

Here, the customary choice for the n initial approximations $z_j(0)$ to the n zeros of

$$p(x) = p_n \prod_{j=1}^{n} (x - z_j)$$

is given by $z_j(0) = Z \; exp(2\pi\sqrt{-1}/n) \; , \quad j = 1, \cdots, n$, Z exceeding (by some fixed factor $t > 1$) $\max_j |z_j|$; for instance, one may set

$$Z = 2 \; t \; \max_{i<n}(p_i/p_n) \; . \tag{10.2}$$

For a fixed i and for all j, the computation according to (10.1) is simple, only involving order n^2 ops, and according to the results of many experiments, the iteration (10.1) rapidly converges to the solution, though no theory confirms or explains these results. Similar is the situation with various modifications of this algorithm, which are now even more popular than the original algorithms and many of which are listed in [Pan92a] and [Pan95b] (also cf. [BPar, McN93]).

On the other hand, there are 2 groups of algorithms that, when implemented, promise to be competitive or even substantially superior to Newton's and Laguerre's iteration, the algorithm by Jenkins and Traub, and all the algorithms of the Durand-Kerner type. One such group is given by the modern modifications and improvements (due to [Pan87], [Ren89], [Pan94a], and [Pan94b]) of *Weyl's quadtree construction* of 1924. In this approach, an initial square S, containing all the zeros of $p(x)$ [say, $S = \{ \; x \; , \; | \; Im \; x \; | < Z, \; | \; Re \; x \; | < Z \; \}$ for Z of (10.2)], is recursively partitioned into 4 congruent subsquares. In the center of each of them, a proximity test is applied that estimates the distance from this center to the closest zero of $p(x)$. If such a distance exceeds one half of the diagonal length, then the subsquare contains no zeros of $p(x)$ and is discarded. When this process ensures a strong isolation from each other for the components formed by the remaining squares, then certain extensions of Newton's iteration [Ren89], [Pan94a], and [Pan94b] or some iterative techniques based on numerical integration [Pan87] are applied and very rapidly converge to the desired approximations to the zeros of $p(x)$, within the error bound $2^{-b}Z$ for Z of (10.2). As a result, the algorithms of [Pan87], [Pan94a], and [Pan94b] solve the entire problem of approximating (within $2^{-b}Z$) all the zeros of $p(x)$ at the overall cost of performing $O(\; (n^2 \log n) \; \log(bn) \;)$ ops (cf. [BPar]), versus order n^2 operations at each iteration of Durand-Kerner type.

The second group is given by the *divide-and-conquer algorithms*. They first compute a sufficiently wide annulus A, which is free of the zeros of $p(x)$ and contains comparable numbers of such zeros (that is, the same numbers up to a fixed constant factor) in its exterior and its interior. Then the 2 factors of $p(x)$ are numerically computed, that is, $F(x)$, having all its zeros in the interior of the annulus, and $G(x) = p(x)/F(x)$,

having no zeros there. The same process is recursively repeated for $F(x)$ and $G(x)$ until factorization of $p(x)$ into the product of linear factors is computed numerically. From this factorization, approximations to all the zeros of $p(x)$ are obtained. The algorithms of [Pan95a], [Panar] based on this approach only require $O(\, n \log(bn)\, (\log n)^2\,)$ ops in order to approximate all the n zeros of $p(x)$ within $2^{-b}Z$ for Z of (10.2). (Note that this is a quite sharp bound: at least n ops are necessary in order to output n distinct values.)

The computations for the polynomial zero problem are ill-conditioned, that is, they generally require a high precision for the worst case input polynomials in order to ensure a required output precision, no matter which algorithm is applied for the solution. Consider, for instance, the polynomial $(x - \frac{6}{7})^n$ and perturb its x-free coefficient by 2^{-bn}. Observe the resulting jumps of the zero $x = 6/7$ by 2^{-b}, and observe similar jumps if the coefficients p_i are perturbed by $2^{(i-n)b}$ for $i = 1, 2, \cdots, n-1$. Therefore, to ensure the output precision of b bits, we need an input precision of at least $(n-i)b$ bits for each coefficient p_i, $i = 0, 1, \cdots, n-1$. Consequently, for the worst case input polynomial $p(x)$, any solution algorithm needs at least about factor n increase of the precision of the input and of computing, versus the output precision.

Numerically unstable algorithms may require even a higher input and computation precision, but the inspection shows that this is not the case for the algorithms of [Pan87], [Ren89], [Pan94a], [Pan94b], [Pan95a], and [Panar] (cf. [BPar]).

2.11 FFT and Fast Polynomial Arithmetic

To yield the record complexity bounds for approximating polynomial zeros, one should exploit fast algorithms for basic operations with polynomials (their multiplication, division, and transformation under the shift of the variable), as well as fast Fourier transform (FFT), both directly and for supporting the fast polynomial arithmetic. The FFT and fast basic polynomial algorithms (including ones for multipoint polynomial evaluation and interpolation) are the basis for many other fast polynomial computations, performed both numerically and symbolically (compare the next sections). These basic algorithms, their impact on the field of algebraic computation, and their complexity estimates have been extensively studied in [AHU74], [BM75], [BP94].

3 Systems of Nonlinear Equations and Other Applications

Given a system $\{p_1(x_1, \ldots, x_n), p_2(x_1, \ldots, x_n), \ldots, p_r(x_1, \ldots, x_n)\}$ of nonlinear polynomials with rational coefficients (each $p_i(x_1, \ldots, x_n)$ is said to be an element of $\mathbb{Q}[x_1, \ldots, x_n]$, the ring of polynomials in $x_1, \ldots, x_n$ over the field $\mathbb{Q}$ of rational numbers), the n-tuple of complex numbers $(a_1, \ldots, a_n)$ is a common solution of the system, if $f_i(a_1, \ldots, a_n) = 0$ for each i with $1 \leq i \leq r$. In this section, we explore the problem of exactly solving a system of nonlinear equations over the field $\mathbb{Q}$. We provide an overview and cite references to different symbolic techniques used for solving systems of algebraic (polynomial) equations. In particular, we describe methods involving *resultant* and *Gröbner basis* computations.

The *Sylvester resultant method* is the technique most frequently utilized for determining a common zero of two polynomial equations in one variable [Knu81]. However, using the Sylvester method successively to solve a system of multivariate polynomials proves to be inefficient. Successive resultant techniques, in general, lack efficiency as a result of their sensitivity to the ordering of the variables[KN92]. It is more efficient to eliminate all variables together from a set of polynomials, thus leading to the notion of the *multivariate resultant*. The three most commonly used multivariate resultant formulations are the *Dixon* [Dix08, KS95b] *Macaulay* [Mac16, Can90, KL88], and *sparse resultant formulations* [CE93a, Stu91]

The theory of Gröbner bases provides powerful tools for performing computations in multivariate polynomial rings. Formulating the problem of solving systems of polynomial equations in terms of polynomial ideals, we will see that a Gröbner basis can be computed from the input polynomial set, thus allowing for a form of back substitution (cf. section 1.3) in order to compute the common roots.

Although not discussed, it should be noted that the *characteristic set algorithm* can be utilized for polynomial system solving. Ritt [Rit50] introduced the concept of a characteristic set as a tool for studying solutions of algebraic differential equations. In 1984 Wu [Wu84, Wu86], in search of an effective method for automatic theorem proving, converted Ritt's method to ordinary polynomial rings. Given the before mentioned system P, the characteristic set algorithm transforms P into a triangular form, such that the set of common zeros of P is equivalent to the set of roots of the triangular system [KN92].

Throughout this exposition we will also see that these techniques used to solve nonlinear equations can be applied to other problems as well, such as computer aided design and automatic geometric theorem proving.

3.1 Resultant Methods

The question of whether two polynomials $f(x), g(x) \in \mathbb{Q}[x]$,

$$\begin{aligned} f(x) &= f_n x^n + f_{n-1} x^{n-1} + \ldots + f_1 x + f_0, \\ g(x) &= g_m x^m + g_{m-1} x^{m-1} + \ldots + g_1 x + g_0, \end{aligned}$$

have a common root leads to a condition that has to be satisfied by the coefficients of both f and g. Using a derivation of this condition due to Euler, the *Sylvester matrix* of f and g (which is of order $m+n$) can be formulated. The vanishing of the determinant of the Sylvester matrix, known as the *Sylvester resultant*, is a necessary and sufficient condition for f and g to have common roots [Knu81].

As a running example let us consider the following system in two variables provided by Lazard [Laz81]:

$$\begin{aligned} f &= x^2 + xy + 2x + y - 1 = 0, \\ g &= x^2 + 3x - y^2 + 2y - 1 = 0. \end{aligned}$$

The Sylvester resultant can be used as a tool for eliminating several variables from a set of equations [KN92]. Without loss of generality, the roots of the Sylvester resultant of f and g treated as polynomials in y, whose coefficients are polynomials in x, are the x-coordinates of the common zeros of f and g. More specifically, the Sylvester resultant of the Lazard system with respect to y is given by the following determinant:

$$\det\left(\begin{bmatrix} x+1 & x^2+2\,x-1 & 0 \\ 0 & x+1 & x^2+2\,x-1 \\ -1 & 2 & x^2+3\,x-1 \end{bmatrix}\right) = -x^3 - 2\,x^2 + 3\,x.$$

The roots of the Sylvester resultant of f and g are $\{-3, 0, 1\}$. For each x value, one can substitute the x value back into the original polynomials yielding the solutions $(-3, 1), (0, 1), (1, -1)$.

The method just outlined can be extended recursively, using *polynomial GCD computations*, to a larger set of multivariate polynomials in $\mathbb{Q}[x_1, \ldots, x_n]$. This technique, however, is impractical for eliminating many variables, due to an explosive growth of the degrees of the polynomials generated in each elimination step.

The Sylvester formulations has led to a *subresultant theory*, developed simultaneously by G.E. Collins and W.S. Brown and J. Traub. The subresultant theory produced an efficient algorithm for computing polynomial GCDs and their resultants, while controlling intermediate expression swell [Bro71, BT71, Col67, Col71, Knu81].

It should be noted that by adopting an implicit representation for symbolic objects, the intermediate expression swell introduced in many symbolic computations can be palliated. Recently, polynomial GCD algorithms have been developed that use implicit representations and thus avoid the computationally costly content and primitive part computations needed in those GCD algorithms for polynomials in explicit representation [DK95, Kal88, KT90].

The solvability of a set of nonlinear multivariate polynomials over the field $\mathbb{Q}$ can be determined by the vanishing of a generalization of the Sylvester resultant of two polynomials in a single variable.

Due to the special structure of the Sylvester matrix, Bézout developed a method for computing the resultant as a determinant of order $\mathrm{Max}(m, n)$ during the eighteenth century. Cayley [Cay65] reformulated Bézout's method leading to Dixon's [Dix08] extension to the bivariate case. Dixon's method can be generalized to a set

$$\{p_1(x_1, \ldots, x_n), p_2(x_1, \ldots, x_n), \ldots, p_{n+1}(x_1, \ldots, x_n)\}$$

of $n+1$ generic n-degree polynomials in n variables [KSY94]. The vanishing of the Dixon resultant is a necessary and sufficient condition for the polynomials to have a non-trivial projective common zero, and also a necessary condition for the existence of an affine common zero. The Dixon formulation gives the resultant

up to a multiple, and hence in the affine case it may happen that the vanishing of the Dixon resultant does not necessarily indicate that the equations in question have a common root. A non-trivial multiple, known as the *projection operator*, can be extracted via a method based on so-called *rank subdeterminant computation* (RSC) [KSY94]. It should be noted that the RSC method can also be applied to the Macaulay and sparse resultant formulations as is detailed here.

In 1916, Macaulay[Mac16] constructed a resultant for n homogeneous polynomials in n variables, which simultaneously generalizes the Sylvester resultant and the determinant of a system of linear equations [CKL89, KN92]. As the Dixon formulation, the Macaulay resultant is a multiple of the resultant (except in the case of generic homogeneous polynomials, where it produces the exact resultant). For the Macaulay formulation, Canny [Can90] has invented a general method that perturbs any polynomial system and extracts a non-trivial projection operator.

Using recent results pertaining to sparse polynomial systems [GKZ94, Stu91, SZ92], the mixed sparse resultant of a system of $n+1$ sparse polynomials in n variables in its matrix form was given by Canny and Emiris [CE93a] and consequently improved in [CE93b, CE95]. Here, sparsity denotes that only certain monomials in each of the $n+1$ polynomials have non-zero coefficients. The determinant of the sparse resultant matrix, like the Macaulay and Dixion matrices, only yields a projection operation, not the exact resultant.

Suppose we are asked to find the common zeros of a set of n polynomials in n variables $\{p_1(x_1,\ldots,x_n)$, $p_2(x_1,\ldots,x_n)$, $\ldots$, $p_n(x_1,\ldots,x_n)\}$. By augmenting the polynomial set by a generic linear form [Can90, CM91, KN92], one can construct the *u-resultant* of a given system of polynomials. The u-resultant factors into linear factors over the complex numbers, providing the common zeros of the given polynomials equations. The u-resultant method takes advantage of the properties of the multivariate resultant, and hence can be constructed using either Dixon's, Macaulay's, or sparse formulations.

Consider the previous example augmented by a generic linear form:

$$\begin{aligned} f_1 &= x^2 + xy + 2x + y - 1 = 0, \\ f_2 &= x^2 + 3x - y^2 + 2y - 1 = 0, \\ f_l &= ux + vy + w = 0. \end{aligned}$$

As described in Canny, Kaltofen and Lakshman [CKL89], the following matrix M corresponds to the Macaulay u-resultant of the above system of polynomials, with z being the homogenizing variable:

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & u & 0 & 0 & 0 \\ 2 & 0 & 1 & 3 & 0 & 1 & 0 & u & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & v & 0 & 0 & 0 \\ 1 & 2 & 1 & 2 & 3 & 0 & w & v & u & 0 \\ -1 & 0 & 2 & -1 & 0 & 3 & 0 & w & 0 & u \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & -1 & 0 & 0 & v & 0 \\ 0 & -1 & 1 & 0 & -1 & 2 & 0 & 0 & w & v \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & w \end{bmatrix}.$$

It should be noted that

$$\det(M) = (u - v + w)(-3u + v + w)(v + w)(u - v)$$

corresponds to the affine solutions $(1,-1)$, $(-3,1)$, $(0,1)$, and one solution at infinity.

An empirical comparison of the detailed resultant formulations can be found in Kapur and Saxena [KS95b]. Recently, the multivariate resultant formulations are being used for other applications such as *algebraic and geometric reasoning* [KSY94], *computer-aided design* [SG86], and for *implicitization and finding base points* [Chi90].

3.2 Gröbner Bases

Solving systems of nonlinear equations can be formulated in terms of polynomial ideals [BW93, GCL92, Win96]. Let us first establish some terminology.

The ideal generated by a system of polynomial equations $p_1, \ldots, p_r$ over $\mathbb{Q}[x_1, \ldots, x_n]$ is the set of all linear combinations

$$(p_1, \ldots, p_r) = \{h_1 p_1 + \ldots + h_r p_r \mid h_1, \ldots, h_r \in \mathbb{Q}[x_1, \ldots, x_n]\}.$$

The algebraic variety of $p_1, \ldots, p_r \in \mathbb{Q}[x_1, \ldots, x_n]$ is the set of their common zeros,

$$V(p_1, \ldots, p_r) = \{(a_1, \ldots, a_n) \in \mathbb{C}^n \mid f_1(a_1, \ldots, a_n) = \ldots = f_r(a_1, \ldots, a_n) = 0\}.$$

A version of the *Hilbert Nullstellensatz* states that

$$V(p_1, \ldots, p_r) = \text{the empty set } \emptyset \Longleftrightarrow 1 \in (p_1, \ldots, p_r) \text{ over } \mathbb{Q}[x_1, \ldots, x_n],$$

which relates the solvability of polynomial systems to the ideal membership problem.

A term $t = x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n}$ of a polynomial is a product of powers with $\deg(t) = e_1 + e_2 + \ldots + e_n$. In order to add needed structure to the polynomial ring we will require that the terms in a polynomial be ordered in an admissible fashion [GCL92, KN92]. Two of the most common admissible orderings are the *lexicographic order* ($\prec_l$), where terms are ordered as in a dictionary, and the *degree order* ($\prec_d$), where terms are first compared by their degrees with equal degree terms compared lexicographically. A variation to the lexicographic order is the *reverse lexicographic order*, where the lexicographic order is reversed [DTS88], page 96.

It is this above mentioned structure that permits a type of simplification known as polynomial reduction. Much like a polynomial remainder process, the process of polynomial reduction involves subtracting a multiple of one polynomial from another to obtain a smaller degree result [BW93, GCL92, KN92, Win96].

A polynomial g is said to be reducible with respect to a set $P = \{p_1, \ldots, p_r\}$ of polynomials if it can be reduced by one or more polynomials in P. When g is no longer reducible by the polynomials in P, we say that g is *reduced* or is *a normal form* with respect to P.

For an arbitrary set of basis polynomials, it is possible that different reduction sequences applied to a given polynomial g could reduce to different normal forms. A basis $G \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is a *Gröbner basis* if and only if every polynomial in $\mathbb{Q}[x_1, \ldots, x_n]$ has a unique normal form with respect to G. Bruno Buchberger [Buc65, Buc76, Buc83, Buc85] showed that every basis for an ideal $(p_1, \ldots, p_r)$ in $\mathbb{Q}[x_1, \ldots, x_n]$ can be converted into a Gröbner basis $\{p_1^*, \ldots, p_s^*\} = GB(p_1, \ldots, p_r)$, concomitantly designing an algorithm that transforms an arbitrary ideal basis into a Gröbner basis. Another characteristic of Gröbner bases is that by using the above mentioned reduction process we have

$$g \in (p_1 \ldots, p_r) \Longleftrightarrow (g \bmod p_1^*, \ldots, p_s^*) = 0.$$

Further, by using the Nullstellensatz it can be shown that $p_1 \ldots, p_r$ viewed as a system of algebraic equations is solvable if and only if $1 \notin GB(p_1, \ldots, p_r)$.

Depending on which admissible term ordering is used in the Gröbner bases construction, an ideal can have different Gröbner bases. However, an ideal cannot have different (reduced) Gröbner bases for the same term ordering.

Any system of polynomial equations can be solved using a lexicographic Gröbner basis for the ideal generated by the given polynomials. It has been observed, however, that Gröbner bases, more specifically lexicographic Gröbner bases, are hard to compute [BW93, GCL92, Lak90, Win96]. In the case of zero-dimensional ideals, those whose varieties have only isolated points, Faugère, Gianni, Lazard and Mora [FGLM93] outlined a change of basis algorithm which can be utilized for solving zero-dimensional systems of equations. In the zero-dimensional case, one computes a Gröbner basis for the ideal generated by a system of polynomials under a degree ordering. The so-called *change of basis algorithm* can then be applied to the degree ordered Gröbner basis to obtain a Gröbner basis under a lexicographic ordering.

Turning to Lazard's example in form of a polynomial basis,

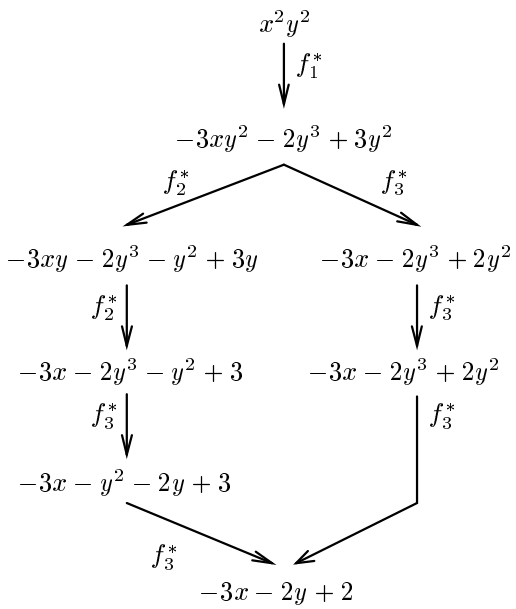
$$\begin{aligned} f_1 &= x^2 + xy + 2x \quad + \; y - 1, \\ f_2 &= x^2 \quad + 3x - y^2 + 2y - 1, \end{aligned}$$

one obtains (under lexicographical ordering with $x \prec_l y$) a Gröbner basis in which the variables are triangularized such that the finitely many solutions can be computed via back substitution:

$$\begin{aligned} {f_1}^* &= x^2 \quad + 3x \quad + 2y - 2, \\ {f_2}^* &= \quad xy - \quad x \quad - \quad y + 1, \\ {f_3}^* &= \qquad\quad y^2 \qquad - 1. \end{aligned}$$

It should be be noted that the final univariate polynomial is of minimal degree and the polynomials used in the back substitution will have degree no larger than the number of roots.

As an example of the process of polynomial reduction with respect to a Gröbner basis, the following demonstrates two possible reduction sequences to the same normal form. The polynomial x^2y^2 is reduced with respect to the previously computed Gröbner basis $\{f_1^*, f_2^*, f_3^*\} = GB(f_1, f_2)$ along the following two distinct reduction paths, both yielding $-3x - 2y + 2$ as the normal form.



There is a strong connection between lexicographic Gröbner bases and the previously mentioned resultant techniques. For some types of input polynomials, the computation of a reduced system via resultants might be much faster than the computation of a lexicographic Gröbner basis. A good comparison between the Gröbner computations and the different resultant formulations can be found in Kapur and Saxena [KS95b].

In a survey article, Buchberger [Buc85] detailed how Gröbner bases can be used as a tool for many polynomial ideal theoretic operations. Other applications of Gröbner basis computations include automatic geometric theorem proving [Kap86, Wu84, Wu86], multivariate polynomial factorization and GCD computations [GT85] , and polynomial interpolation [LS95, LS94].

4 Polynomial Factorization

The problem of factoring polynomials is a fundamental task in symbolic algebra. An example in one's early mathematical education is the factorization $x^2 - y^2 = (x + y) \cdot (x - y)$, which in algebraic terms is a factorization of a polynomial in two variables with integer coefficients. Technology has advanced to a state where most polynomial factorization problems are doable on a computer, in particular, with any of the popular mathematical software, such as the Mathematica or Maple systems. For instance, the factorization of the determinant of a 6×6 symmetric Toeplitz matrix over the integers is computed in Maple as

```
> readlib(showtime):
> showtime():
01 := T := linalg[toeplitz]([a,b,c,d,e,f]);
```

$$T := \begin{bmatrix} a & b & c & d & e & f \\ b & a & b & c & d & e \\ c & b & a & b & c & d \\ d & c & b & a & b & c \\ e & d & c & b & a & b \\ f & e & d & c & b & a \end{bmatrix}$$

```
time   0.03   words   7701
O2 := factor(linalg[det](T));
```

$$\begin{aligned} -(&2\,d\,c\,a - 2\,b\,c\,e + 2\,c^2\,a - a^3 - d\,a^2 + 2\,d^2\,c + d^2\,a + b^3 + 2\,a\,b\,c - 2\,c^2\,b \\ &+ d^3 + 2\,a\,b^2 - 2\,d\,c\,b - 2\,c\,b^2 - 2\,e\,c^2 + 2\,e\,b^2 + 2\,f\,c\,b + 2\,b\,a\,e \\ &+ b^2\,f + c^2\,f + b\,e^2 - b\,a^2 - f\,d\,b - f\,d\,a - f\,a^2 - f\,b\,a + e^2\,a - 2\,d\,b^2 \\ &+ d\,c^2 - 2\,d\,e\,b - 2\,d\,e\,c - d\,b\,a)(2\,d\,c\,a - 2\,b\,c\,e - 2\,c^2\,a + a^3 \\ &- d\,a^2 - 2\,d^2\,c - d^2\,a + b^3 + 2\,a\,b\,c - 2\,c^2\,b + d^3 - 2\,a\,b^2 + 2\,d\,c\,b \\ &+ 2\,c\,b^2 + 2\,e\,c^2 - 2\,e\,b^2 - 2\,f\,c\,b + 2\,b\,a\,e + b^2\,f + c^2\,f + b\,e^2 - b\,a^2 \\ &- f\,d\,b + f\,d\,a - f\,a^2 + f\,b\,a - e^2\,a - 2\,d\,b^2 + d\,c^2 + 2\,d\,e\,b - 2\,d\,e\,c \\ &+ d\,b\,a) \end{aligned}$$

```
time   27.30   words   857700
```

Clearly, the Toeplitz determinant factorization requires more than tricks from high school algebra. Indeed, the development of modern algorithms for the polynomial factorization problem is one of the great successes of the discipline of symbolic mathematical computation. Kaltofen has surveyed the algorithms until 1992 in [Kal82, Kal90, Kal92], mostly from a computer science perspective. In this article we shall focus on the applications of the known fast methods to problems in science and engineering. For a more extensive set of references, please refer to Kaltofen's survey articles.

4.1 Polynomials in a single variable over a finite field

At the first glance, the problem of factoring an integer polynomial modulo a prime number appears to be very similar to the problem of factoring an integer represented in a prime radix. That is simply not so. The factorization of the polynomial $x^{511} - 1$ can be done modulo 2 on a computer in a matter of milliseconds, while the factorization of the integer $2^{511} - 1$ into its integer factors is a computational challenge. For those interested: the largest prime factors of $2^{511} - 1$ have 57 and 67 decimals digits, respectively, which makes a tough but not undoable 123 digit product for the number field sieve factorizer [Ley95]. Irreducible factors of polynomials modulo 2 are needed to construct finite fields. For example, the factor $x^9 + x^4 + 1$ of $x^{511} - 1$ leads to a model of the finite field with 2^9 elements, GF(2^9), by simply computing with the polynomial remainders modulo $x^9 + x^4 + 1$ as the elements. Such irreducible polynomials are used for setting up error-correcting codes, for instance, the BCH codes [MS77]. Berlekamp's [Ber67, Ber70] pioneering work on factoring polynomials over a finite field by linear algebra is done with this motivation. The linear algebra tools that Berlekamp used seem to have been introduced to the subject as early as in 1937 by Petr (cf. [Sch56]).

Today, factoring algorithms for univariate polynomials over finite fields form the innermost subalgorithm to lifting-based algorithms for factoring polynomials in one [Zas69] and many [Mus75] variables over the integers. When Maple computed the factorization of the above Toeplitz determinant, it began with factoring a univariate polynomial modulo a prime integer. The case when the prime integer is very large has lead to a significant development in computer science itself. As it turns out, by selecting random residues the expected performance of the algorithms can be speeded up exponentially [Ber70, Rab80]. Randomization is now an important tool for designing efficient algorithms and has proliferated to many fields of computer science. Paradoxically, the random elements are produced by a congruential random number generator, and the actual computer implementations are quite deterministic, which leads some computer scientists to believe

that random bits can be eliminated in general at no exponential slow-down. Nonetheless, for the polynomial factoring problem modulo a large prime, no fast methods are known to-date that would work without this "probabilistic" approach.

One can measure the computing time of selected algorithms in terms of n, the degree of the input polynomial, and p, the cardinality of the field. When counting arithmetic operations modulo p (including reciprocals), the best known algorithms are quite recent. Berlekamp's 1970 method performs $O(n^\omega + n^{1+o(1)} \log p)$ residue operations. Here and below, ω denotes the exponent implied by the used linear system solver, i.e., $\omega = 3$ when classical methods are used, and $\omega = 2.376$ when asymptotically fast (though impractical) matrix multiplication is assumed. The correction term $o(1)$ accounts for the $\log n$ factors derived from the FFT-based fast polynomial multiplication and remaindering algorithms. An approach in the spirit of Berlekamp's but possibly more practical for $p = 2$ has recently been discovered by Niederreiter [Nie94]. A very different technique by Cantor and Zassenhaus [CZ81] first separates factors of different degrees and then splits the resulting polynomials of equal degree factors. It has $O(n^{2+o(1)} \log p)$ complexity and is the basis for the following two methods. Algorithms by von zur Gathen and Shoup [vzGS92] have running time $O(n^{2+o(1)} + n^{1+o(1)} \log p)$ and those by Kaltofen and Shoup [KS95a] have running time $O(n^{1.815} \log p)$, the latter with fast matrix multiplication.

For n and p simultaneously large, a variant of the method by Kaltofen and Shoup [KS95a] that uses classical linear algebra and runs in $O(n^{2.5} + n^{1+o(1)} \log p)$ residue operations is the current champion among the practical algorithms. With it Shoup, using his own fast polynomial arithmetic package [Sho96], has factored a random-like polynomial of degree 2048 modulo a 2048-bit prime number in about 12 days on a Sparc-10 computer using 68 Mbyte of main memory. For even larger n, but smaller p, parallelization helps, and Kaltofen and Lobo [KL94] could factor a polynomial of degree $n = 15\,001$ modulo $p = 127$ in about 6 days on 8 computers that are rated at 86.1 MIPS. To-date, the largest polynomial factored modulo 2 is $X^{216\,091} + X + 1$; this was accomplished by Peter Montogmery in 1991 by using Cantor's fast polynomial multiplication algorithm based on additive transforms [Can89].

4.2 Polynomials in a single variable over fields of characteristic zero

As mentioned before, generally usable methods for factoring univariate polynomials over the rational numbers begin with the Hensel lifting techniques introduced by Zassenhaus [Zas69]. The input polynomial is first factored modulo a suitable prime integer p, and then the factorization is lifted to one modulo p^k for an exponent k of sufficient size to accommodate all possible integer coefficients that any factors of the polynomial might have. The lifting approach is fast in practice, but there are hard-to-factor polynomials on which it runs an exponential time in the degree of the input. This slowdown is due to so-called parasitic modular factors. The polynomial $x^4 + 1$, for example, factors modulo all prime integers but is irreducible over the integers: it is the cyclotomic equation for 8-th roots of unity. The products of all subsets of modular factors are candidates for integer factors, and irreducible integer polynomials with exponentially many such subsets exist [KMS83].

The elimination of the exponential bottleneck by giving a polynomial-time solution to the integer polynomial factoring problem, due to A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász [LLL82], is considered a major result in computer science algorithm design. The key ingredient to their solution is the construction of integer relations to real or complex numbers. For the simple demonstration of this idea, consider the polynomial

$$x^4 + 2x^3 - 6x^2 - 4x + 8.$$

A root of this polynomial is $\alpha \approx 1.236067977$, and $\alpha^2 \approx 1.527864045$. We note that $2\alpha + \alpha^2 \approx 4.000000000$, hence $x^2 + 2x - 4$ is a factor. The main difficulty is to efficiently compute the integer linear relation with relatively small coefficients for the high precision big-float approximations of the powers of a root. Lenstra *et al.* solve this diophantine optimization problem by means of their now famous lattice reduction procedure, which is somewhat reminiscent of the ellipsoid method for linear programming.

The determination of linear integer relations among a set of real or complex numbers is a useful task in science in general. Very recently, some stunning identities could be produced by this method, including the

following formula for π [Fin95]:

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right).$$

Even more surprising, the lattice reduction algorithm can prove that no linear integer relation with integers smaller than a chosen parameter exists among the real or complex numbers. There is an efficient alternative to the lattice reduction algorithm, originally due to H. Ferguson and R. W. Forcade [FF82] and recently improved by Ferguson and D. Bailey.

The complexity of factoring an integer polynomial of degree n with coefficients of no more than l bits is thus a polynomial in n and l. From a theoretical point of view, an algorithm with a low estimate is by V. Miller [Mil92] and has a running time of $O(n^{5+o(1)}l^{1+o(1)} + n^{4+o(1)}l^{2+o(1)})$ bit-operations. It is expected that the relation-finding methods will become usable in practice on hard-to-factor polynomials in the near future. If the hard-to-factor input polynomial is irreducible, an alternate approach can be used to prove its irreducibility. One finds an integer evaluation point at which the integral value of the polynomial has a large prime factor, and the irreducibility follows by mathematical theorems. M. Monagan [Mon92] has proven large hard-to-factor polynomials irreducible in this way, which would be hopeless by the lifting algorithm.

Coefficient fields other than finite fields and the rational numbers are of interest. Computing the factorizations of univariate polynomials over the complex numbers is the root finding problem described in the section *Approximating Polynomial Zeros* before. When the coefficient field has an extra variable, such as the field of fractions of polynomials ("rational functions") the problem reduces, by an old theorem of C. F. Gauss, to factoring multivariate polynomials, which we discuss below. When the coefficient field is the field of Laurent series in t with a finite segment of negative powers,

$$\frac{c_{-k}}{t^k} + \frac{c_{-k+1}}{t^{k-1}} + \cdots + \frac{c_{-1}}{t} + c_0 + c_1 t + c_2 t^2 + \cdots, \quad \text{where } k \geq 0,$$

fast methods appeal to the theory of Puiseux series, which constitute the domain of algebraic functions [Wal93].

4.3 Polynomials in two variables

Factoring bivariate polynomials by reduction to univariate factorization via homomorphic projection and subsequent lifting can be done similarly to the univariate algorithm [Mus75]. The second variable y takes the role of the prime integer p and $f(x, y) \bmod y = f(x, 0)$. Lifting is possible only if $f(x, 0)$ had no multiple root. Provided that $f(x, y)$ has no multiple factor, which can be insured by a simple GCD computation, the squarefreeness of $f(x, 0)$ can be obtained by variable translation $\hat{y} = y + a$, where a is an easy-to-find constant in the coefficient field. For certain domains, such as the rational numbers, any irreducible multivariate polynomial $h(x, y)$ can be mapped to an irreducible univariate polynomial $h(x, b)$ for some constant b. This is the important *Hilbert irreducibility theorem*, whose consequence is that the combinatorial explosion observed in the univariate lifting algorithm is, in practice, unlikely. However, the magnitude and probabilistic distribution of good points b is not completely analyzed.

For so-called non-Hilbertian coefficient fields good reduction is not possible. An important such field are the complex numbers. Clearly, all $f(x, b)$ completely split into linear factors, while $f(x, y)$ may be irreducible over the complex numbers. An example for an irreducible polynomial is $f(x, y) = x^2 - y^3$. Polynomials that remain irreducible over the complex numbers are called absolutely irreducible. An additional problem is the determination of the algebraic extension of the ground field in which the absolutely irreducible factors can be expressed. In the example

$$x^6 - 2x^3y^2 + y^4 - 2x^3 = (x^3 - \sqrt{2}x - y^2) \cdot (x^3 + \sqrt{2}x - y^2),$$

the needed extension field is $\mathbb{Q}(\sqrt{2})$. The relation-finding approach proves successful for this problem. The root is computed as a Taylor series in y, and the integrality of the linear relation for the powers of the series means that the multipliers are polynomials in y of bounded degree. Several algorithms of polynomial-time complexity and pointers to the literature are found in [Kal95].

Bivariate polynomials constitute implicit representations of algebraic curves. It is an important operation in geometric modeling to convert from implicit to parametric representation. For example, the circle

$$x^2 + y^2 - 1 = 0$$

has the rational parameterization

$$x = \frac{2t}{1+t^2}, \quad y = \frac{1-t^2}{1+t^2}, \quad \text{where } -\infty \leq t \leq \infty.$$

Algorithms are known that can find such rational parameterizations provided that they exist [SW91]. It is crucial that the inputs to these algorithms are absolutely irreducible polynomials.

4.4 Polynomials in many variables

Polynomials in many variables, such as the symmetric Toeplitz determinant exhibited above, are rarely given explicitly, due to the fact that the number of possible terms grows exponentially in the number of variables: there can be as many as $\binom{n+v}{n} \geq 2^{\min\{n,v\}}$ terms in a polynomial of degree n with v variables. Even the factors may be dense in canonical representation, but could be sparse in another basis: for instance, the polynomial

$$(x_1 - 1)(x_2 - 2)\cdots(x_v - v) + 1$$

has only 2 terms in the "shifted basis," while it has 2^v terms in the power basis, i.e., in expanded format.

Randomized algorithms are available that can efficiently compute a factor of an implicitly given polynomial, say, a matrix determinant, and even can find a shifted basis with respect to which a factor would be sparse, provided, of course, that such a shift exists. The approach is by manipulating polynomials in so-called black box representations [KT90]: a black box is an object that takes as input a value for each variable, and then produces the value of the polynomial it represents at the specified point. In the Toeplitz example the representation of the determinant could be the Gaussian elimination program which computes it. We note that the size of the polynomial in this case would be nearly constant, only the variable names and the dimension need to be stored. The factorization algorithm then outputs procedures which will evaluate all irreducible factors at an arbitrary point (supplied as the input). These procedures make calls to the black box given as input to the factorization algorithm in order to evaluate them at certain points, which are derived from the point at which the procedures computing the values of the factors are probed. It is, of course, assumed that subsequent calls evaluate one and the same factor and not associates that are scalar multiples of one another. The algorithm by Kaltofen and Trager [KT90] finds procedures that with a controllably high probability evaluate the factors correctly. Randomization is needed to avoid parasitic factorizations of homomorphic images which provide some static data for the factor boxes and cannot be avoided without mathematical conjecture. The procedures that evaluate the individual factors are deterministic.

Factors constructed as black box programs are much more space efficient than those represented in other formats, for example, the straight-line program format [Kal89]. More importantly, once the black box representation for the factors is found, sparse representations can be rapidly computed by any of the new sparse interpolation algorithms. See [GL95] for the latest method allowing shifted bases and pointers to the literature of other methods, including ones for the standard power bases.

The black box representation of polynomials is normally not supported by commercial computer algebra systems such as Axiom, Maple, or Mathematica. Díaz is currently developing the FoxBox system in C++ that makes black box methodology available to users of such systems. It is anticipated that factorizations as those of large symmetric Toeplitz determinants will be possible on computers. Earlier implementations based on the straight-line program model [FIKL88] could factor 16×16 group determinants, which represent polynomials of over 300 million terms.

5 Further Information

The books [Knu81], [DTS88], [GCL92], and [Zip93] provide a much broader introduction to the general subject. There are well known libraries and packages of subroutines for the most popular numerical matrix computations, in particular, [D$^+$78] for solving linear systems of equations, [S$^+$70] and [G$^+$72] for

approximating matrix eigenvalues, and [A+92] for both of the 2 latter computational problems. There is a comprehensive treatment of numerical matrix computations [GL89], with extensive bibliography, and there are several more specialized books on them [GL81], [Wil65], [Par80], [Saa92], [Saa95], as well as many survey articles [HNP91], [Wat91], [OV85], [Pan92b] and thousands of research articles.

Special (more efficient) parallel algorithms have been devised for special classes of matrices, such as sparse [PR93], [Pan93], banded [PSA95], and dense structured (cf. [BP94]). We also refer to [PP95] on simple but surprisingly effective extension of Brent's principle for improving the processor and work efficiency of parallel matrix algorithms and to [GL89], [OV85] and [HNP91] on practical parallel algorithms for matrix computations.

Defining Terms

characteristic polynomial: a polynomial associated with a square matrix, the determinant of the matrix when a single variable is subtracted to its diagonal entries. The roots of the characteristic polynomial are the eigenvalues of the matrix.

condition number: a scalar derived from a matrix that measures its relative nearness to a singular matrix. Very close to singular means a large condition number, in which case numeric inversion becomes an unstable process.

degree order: an order on the terms in a multivariate polynomial; for two variables x and y with $x \prec y$ the ascending chain of terms is $1 \prec x \prec y \prec x^2 \prec xy \prec y^2 \cdots$.

determinant: a polynomial in the entries of a square matrix with the property that its value is non-zero if and only if the matrix is invertible

lexicographic order: an order on the terms in a multivariate polynomial; for two variables x and y with $x \prec y$ the ascending chain of terms is $1 \prec x \prec x^2 \prec \cdots \prec y \prec xy \prec x^2y \cdots \prec y^2 \prec xy^2 \cdots$.

ops: arithmetic operations, i.e., additions, subtractions, multiplications, or divisions; as in **flops**, i.e., floating point operations.

singularity: a square matrix is singular if there is a non-zero second matrix such the the product of the two is the zero matrix. Singular matrices do not have inverses.

sparse matrix: a matrix where many of the entries are zero.

structured matrix: a matrix where each entry can be derived by a formula depending on few parameters. For instance, the Hilbert matrix has $1/(i+j-1)$ as the entry in row i and column j.

References

[A+92] E. Anderson et al. *LAPACK Users' Guide.* SIAM Publications, Philadelphia, 1992.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Algorithms.* Addison and Wesley, Reading, MA, 1974.

[Bar68] E. H. Bareiss. Sylvester's identity and multistep integers preserving Gaussian elimination. *Math. Comp.*, 22:565–578, 1968.

[Ber67] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell Systems Tech. J.*, 46:1853–1859, 1967. Republished in revised form in: E. R. Berlekamp, *Algebraic Coding Theory*, Chapter 6, McGraw-Hill Publ., New York, 1968.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.

[BM75] A. Borodin and I. Munro. *Computational Complexity of Algebraic and Numeric Problems.* American Elsevier, New York, N.Y., 1975.

[BP91] D. Bini and V. Y. Pan. Parallel complexity of tridiagonal symmetric eigenvalue problem. In *Proc. 2nd Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 384–393. ACM Press, New York, and SIAM Publications, Philadelphia, 1991.

[BP94] D. Bini and V. Y. Pan. *Polynomial and Matrix Computations, V.1, Fundamental Algorithms.* Birkhäuser, Boston, 1994.

[BPar] D. Bini and V. Y. Pan. *Polynomial and Matrix Computations, V.2.* Birkhäuser, Boston, to appear.

[Bro71] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18:478–504, 1971.

[BT71] W. S. Brown and J. F. Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM*, 18:505–514, 1971.

[Buc65] B. Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.d. thesis, Univ. Innsbruck, Fall 1965.

[Buc76] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical form. *ACM SIGSAM Bulletin*, 10(3):19–29, 1976.

[Buc83] B. Buchberger. A note on the complexity of constructing Gröbner-bases. In J. A. van Hulzen, editor, *Proc. EUROCAL '83*, Springer Lec. Notes Comp. Sci., pages 137–145, 1983.

[Buc85] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Recent Trends in Multidimensional Systems Theory*, pages 184–232. D. Reidel Publ. Comp., Dordrecht (Holland), 1985.

[BW93] T. Becker and V. Weispfenning. *Gröbner bases A Computational Approach to Commutative Algebra.* Springer Verlag, New York, N.Y., 1993.

[Can89] D. G. Cantor. On arithmetical algorithms over finite fields. *J. Combinatorial Theory, Series A*, 50:285–300, 1989.

[Can90] J. Canny. Generalized characteristic polynomials. *J. Symbolic Comput.*, 9(3):241–250, 1990.

[Cay65] A. Cayley. On the theory of eliminaton. *Cambridge and Dublin Mathematical Journal*, 3:210–270, 1865.

[CE93a] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In G. Cohen, T. Mora, and O. Moreno, editors, *Proc. AAECC-10*, volume 673 of *Springer Lect. Notes Comput. Sci.*, pages 89–104, 1993.

[CE93b] J. Canny and I. Emiris. A practical method for the sparse resultant. In M. Bronstein, editor, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '93*, pages 183–192, New York, N. Y., 1993. ACM Press.

[CE95] J. Canny and I. Emiris. Efficient incremental algorithms for the sparse resultant and the mixed volume. *J. Symbolic Computation*, 20(2):117–149, August 1995.

[Chi90] E. Chionh. Base points, resultants and implicit representation of rational surfaces. Ph.d. thesis, Dept. Comput. Sci., Univ. Waterloo, 1990.

[CKL89] J. Canny, E. Kaltofen, and Yagati Lakshman. Solving systems of non-linear polynomial equations faster. *Proc. ACM-SIGSAM 1989 Internat. Symp. Symbolic Algebraic Comput.*, pages 121–128, 1989.

[CM91] J. Canny and D. Manocha. Efficient techniques for multipolynomial resultant algorithms. In S. M. Watt, editor, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '91*, pages 85–95, New York, N.Y., 1991. ACM Press.

[Col67] G. E. Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM*, 14:128–142, 1967.

[Col71] G. E. Collins. The calculation of multivariate polynomial resultants. *J. ACM*, 18:515–532, 1971.

[Cup81] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36:177–195, 1981.

[CZ81] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comp.*, 36:587–592, 1981.

[D+78] J. Dongarra et al. *LAPACK Users' Guide*. SIAM Publications, Philadelphia, 1978.

[Dix08] A. L. Dixon. The elimination of three quantics in two independent variables. In *Proc. London Mathematical Society*, volume 6, pages 468–478, 1908.

[DK95] A. Diaz and E. Kaltofen. On computing greatest common divisors with polynomials given by black boxes for their evaluation. In A. H. M. Levelt, editor, *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC '95*, pages 232–239, New York, N. Y., 1995. ACM Press.

[DTS88] J. H. Davenport, E. Tournier, and Y. Siret. *Computer Algebra Systems and Algorithms for Algebraic Computation*. Academic Press, London, 1988.

[FF82] H. R. P. Ferguson and R. W. Forcade. Multidimensional Euclidean algorithms. *J. reine angew. Math.*, 334:171–181, 1982.

[FGLM93] J. C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *J. Symbolic Comput.*, 16(4):329–344, 1993.

[FIKL88] T. S. Freeman, G. Imirzian, E. Kaltofen, and Yagati Lakshman. DAGWOOD: A system for manipulating polynomials given by straight-line programs. *ACM Trans. Math. Software*, 14(3):218–240, 1988.

[Fin95] S. Finch. The miraculous Bailey-Borwein-Plouffe pi algorithm. Internet document, Mathsoft Inc., `http://www.mathsoft.com/asolve/plouffe/plouffe.html`, October 1995.

[Fos81] L. V. Foster. Generalizations of Laguerre's method: higher order methods. *SIAM J. Numer. Anal.*, 18:1004–1018, 1981.

[G+72] B. S. Garbow et al. *Matrix Eigensystem Routines: EISPACK Guide Extension*. Springer, New York, 1972.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publ., 1992.

[Gie95] M. Giesbrecht. Nearly optimal algorithms for canonical matrix forms. *SIAM J. Comput.*, 24(5):948–969, 1995.

[GKZ94] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser Verlag, Boston, 1994.

[GL81] A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Linear Systems*. Prentice Hall, Englewood Cliffs, N.J., 1981.

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

[GL95] D. Yu. Grigoriev and Y. N. Lakshman. Algorithms for computing sparse shifts for multivariate polynomials. In A. H. M. Levelt, editor, *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC '95*, pages 96–103, New York, N. Y., 1995. ACM Press.

[GM84] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley–Interscience, New York, 1984.

[GT85] P. Gianni and B. Trager. GCD's and factoring polynomials using Gröbner bases. *Proc. EUROCAL '85, Vol. 2, Springer Lec. Notes Comp. Sci.*, 204:409–410, 1985.

[GT87] J. R. Gilbert and R. E. Tarjan. The analysis of a nested dissection algorithm. *Numer. Math.*, 50:377–404, 1987.

[HNP91] M. T. Heath, E. Ng, and B. W. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33:420–460, 1991.

[HPR77] E. Hansen, M. Patrick, and J. Rusnack. Some modifications of Laguerre's method. *BIT*, 17:409–417, 1977.

[JT70] M. A. Jenkins and J. F. Traub. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. *Numer. Math.*, 14:252–263, 1970.

[Kal82] E. Kaltofen. Polynomial factorization. In B. Buchberger, G. Collins, and R. Loos, editors, *Computer Algebra, 2nd ed.*, pages 95–113. Springer Verlag, Vienna, 1982.

[Kal88] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *J. ACM*, 35(1):231–264, 1988.

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press, Greenwhich, Connecticut, 1989.

[Kal90] E. Kaltofen. Polynomial factorization 1982-1986. In D. V. Chudnovsky and R. D. Jenks, editors, *Computers in Mathematics*, volume 125 of *Lecture Notes in Pure and Applied Mathematics*, pages 285–309. Marcel Dekker, Inc., New York, N. Y., 1990.

[Kal92] E. Kaltofen. Polynomial factorization 1987-1991. In I. Simon, editor, *Proc. LATIN '92*, volume 583 of *Springer Lect. Notes Comput. Sci.*, pages 294–313, 1992.

[Kal95] E. Kaltofen. Effective Noether irreducibility forms and applications. *J. Comput. System Sci.*, 50(2):274–295, 1995.

[Kap86] D. Kapur. Geometry theorem proving using Hilbert's nullstellensatz. *J. Symbolic Comp.*, 2:399–408, 1986.

[KKS90] E. Kaltofen, M. S. Krishnamoorthy, and B. D. Saunders. Parallel algorithms for matrix normal forms. *Linear Algebra and Applications*, 136:189–208, 1990.

[KL88] E. Kaltofen and Yagati Lakshman. Improved sparse multivariate polynomial interpolation algorithms. *Proc. ISSAC '88, Springer Lect. Notes Comput. Sci.*, 358:467–474, 1988.

[KL94] E. Kaltofen and A. Lobo. Factoring high-degree polynomials by the black box Berlekamp algorithm. In J. von zur Gathen and M. Giesbrecht, editors, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94*, pages 90–98, New York, N. Y., 1994. ACM Press.

[KMS83] E. Kaltofen, D. R. Musser, and B. D. Saunders. A generalized class of polynomials that are hard to factor. *SIAM J. Comp.*, 12(3):473–485, 1983.

[KN92] D. Kapur and Lakshman Y. N. Elimination methods an introduction. In B. Donald, D. Kapur, and J. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*. Academic Press, 1992.

[Knu81] D. E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Ed. 2.* Addison Wesley, Reading, MA, 1981.

[KP91] E. Kaltofen and V. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. 3rd Ann. ACM Symp. Parallel Algor. Architecture*, pages 180–191, New York, N.Y., 1991. ACM Press. Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

[KP92] E. Kaltofen and V. Pan. Processor-efficient parallel solution of linear systems II: the positive characteristic and singular cases. In *Proc. 33rd Annual Symp. Foundations of Comp. Sci.*, pages 714–723, Los Alamitos, California, 1992. IEEE Computer Society Press. Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

[KS95a] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. In *Proc. 27th Annual ACM Symp. Theory Comp.*, pages 398–406, New York, N.Y., 1995. ACM Press.

[KS95b] D. Kapur and T. Saxena. Comparison of various multivariate resultant formulations. In A. H. M. Levelt, editor, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '95*, pages 187–195, New York, N.Y., 1995. ACM Press.

[KSY94] D. Kapur, T. Saxena, and L. Yang. Algebraic and geometric reasoning using Dixon resultants. In J. von zur Gathen and M. Giesbrecht, editors, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94*, pages 99–107, New York, N. Y., 1994. ACM Press.

[KT90] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput.*, 9(3):301–320, 1990.

[Lak90] Y. N. Lakshman. On the complexity of computing Gröbner bases for zero dimensional polynomia. Ph.d. thesis, Dept. Comput. Sci., Rensselaer Polytechnic Institute, Troy, NY, December 1990.

[Laz81] D. Lazard. Resolution des systemes d'equation algebriques. *Theoretical Comput. Sci.*, 15:77–110, 1981. In French.

[Ley95] P. Leyland. Cunningham project data. Internet document, Oxford Univ., `ftp://sable.ox.ac.uk/pub/math/cunningham/`, November 1995.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[LRT79] R. J. Lipton, D. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. on Numer. Analysis*, 16(2):346–358, 1979.

[LS94] Y. N. Lakshman and B. D. Saunders. On computing sparse shifts for univariate polynomials. In J. von zur Gathen and M. Giesbrecht, editors, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94*, pages 108–113, New York, N. Y., 1994. ACM Press.

[LS95] Y. N. Lakshman and B. D. Saunders. Sparse polynomial interpolation in non-standard bases. *SIAM J. Comput.*, 24(2):387–397, 1995.

[Mac16] F. S. Macaulay. Algebraic theory of modular systems. Cambridge Tracts 19, Cambridge, 1916.

[Mad73] K. Madsen. A root-finding algorithm based on Newton's method. *BIT*, 13:71–75, 1973.

[McC87] S. McCormick, editor. *Multigrid Methods.* SIAM Publications, Phiadelphia, 1987.

[McN93] J. M. McNamee. A bibliography on roots of polynomials. *J. Comput. Applied Math.*, 47(3):391–394, 1993.

[Mil92] V. Miller. Factoring polynomials via relation-finding. In D. Dolev, Z. Galil, and M. Rodeh, editors, *Proc. ISTCS '92*, volume 601 of *Springer Lect. Notes Comput. Sci.*, pages 115–121, 1992.

[Mon92] M. B. Monagan. A heuristic irreducibility test for univariate polynomials. *J. Symbolic Comput.*, 13(1):47–57, 1992.

[MS77] F. J. MacWilliams and N. J. A. Sloan. *The Theory of Error-Correcting Codes.* North-Holland Publ. Comp., New York, 1977.

[Mus75] D. R. Musser. Multivariate polynomial factorization. *J. ACM*, 22:291–308, 1975.

[Nie94] H. Niederreiter. New deterministic factorization algorithms for polynomials over finite fields. In L. Mullen and P. J.-S. Shiue, editors, *Finite Fields: Theory, Applications and Algorithms*, volume 168 of *Contemporary Mathematics*, pages 251–268, Providence, Rhode Island, 1994. Amer. Math. Soc.

[OV85] J. M. Ortega and R. G. Voight. Solution of partial differential equations on vector and parallel computers. *SIAM Review*, 27(2):149–240, 1985.

[Pan84a] V. Y. Pan. How can we speed up matrix multiplication? *SIAM Rev.*, 26(3):393–415, 1984.

[Pan84b] V. Y. Pan. How to multiply matrices faster. *Lecture Notes in Computer Science*, 179, 1984.

[Pan87] V. Y. Pan. Sequential and parallel complexity of approximate evaluation of polynomial zeros. *Computers in Mathematics (with Applications)*, 14(8):591–622, 1987.

[Pan91] V. Y. Pan. Complexity of algorithms for linear systems of equations,. In E. Spedicato, editor, *Computer Algorithms for Solving Linear Algebraic Equations (State of the Art)*, volume 77 of *NATO ASI Series,* Series F: Computer and Systems Sciences, pages 27–56, Berlin, 1991. Springer.

[Pan92a] V. Y. Pan. Complexity of computations with matrices and polynomials. *SIAM Review*, 34(2):225–262, 1992.

[Pan92b] V. Y. Pan. Linear systems of algebraic equations. In Marvin Yelles, editor, *Encyclopedia of Physical Sciences and Technology (2nd edition)*, volume 8, pages 779–804. 1992. Volume 7, pages 304–329, 1987 (first edition).

[Pan93] V. Y. Pan. Parallel solution of sparse linear and path systems. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, chapter 14, pages 621–678. Morgan Kaufmann Publ., San Mateo, CA, 1993.

[Pan94a] V. Y. Pan. Improved parallel solution of a triangular linear system. *Computers and Mathematics (with Applications)*, 27(11):41–43, 1994.

[Pan94b] V. Y. Pan. On approximating polynomial zeros: Modified quadtree construction and improved Newton's iteration. Manuscript, Lehman College, 1994.

[Pan95a] V. Y. Pan. Parallel computation of a Krylov matrix for a sparse and structured input. *Mathematical and Computer Modelling*, 21(11):97–99, 1995.

[Pan95b] V. Y. Pan. Solving a polynomial equation: Some history and recent progress. Manuscript, Lehman College, 1995.

[Panar] V. Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Computers in Mathematics (with Applications)*, to appear.

[Par80] B. Parlett. *Symmetric Eigenvalue Problem.* Prentice Hall, Englewood Cliffs, N.J., 1980.

[PP95] V. Y. Pan and F. P. Preparata. Work-preserving speed-up of parallel matrix computations. *SIAM J. Comput.*, 24(4):811–821, 1995.

[PR92] V. Y. Pan and J. H. Reif. Compact multigrid. *SIAM J. on Scientific and Statistical Computing*, 13(1):119–127, 1992.

[PR93] V. Y. Pan and J. H. Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM J. Comp.*, 22(6):1227–1250, 1993.

[PSA95] V. Y. Pan, I. Sobze, and A. Atinkpahoun. On parallel computations with band matrices. *Information and Computation*, 120(2):227–250, 1995.

[Qui94] M. J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, Inc., New York, N.Y., 1994.

[Rab80] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comp.*, 9:273–280, 1980.

[Ren89] J. Renegar. On the worst case arithmetic complexity of approximating zeros of systems of polynomials. *SIAM J. Comput.*, 18(2):350–370, 1989.

[Rit50] J. F. Ritt. *Differential Algebra.* AMS, New York, 1950.

[S+70] B. T. Smith et al. *Matrix Eigensystem Routines: EISPACK Guide, 2nd ed.* Springer, New York, 1970.

[Saa92] Y. Saad. *Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms.* Manchester University Press, United Kingdom, 1992. John Wiley and Sons, New York, 1992.

[Saa95] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Kent Publishing Company, Boston, 1995.

[Sch56] Št. Schwarz. On the reducibility of polynomials over a finite field. *Quart. J. Math. Oxford Ser. (2)*, 7:110–124, 1956.

[SG86] T. Stederberg and R. Goldman. Algebraic geometry for computer-aided design. *IEEE Computer Graphics and Applications*, 6(6):52–59, 1986.

[Sho96] V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, page to appear, 1996.

[Stu91] B. Sturmfels. Sparse elimination theory. In D. Eisenbud and L. Robbiano, editors, *Proc. Computat. Algebraic Geom. and Commut. Algebra*, Cortona, Italy, June 1991.

[SW91] J. R. Sendra and F. Winkler. Symbolic parameterization of curves. *J. Symbolic Comput.*, 12(6):607–631, 1991.

[SZ92] B. Sturmfels and A. Zelevinsky. Multigraded resultants of the sylvester type. *Journal of Algebra*, 1992.

[vzGS92] J. von zur Gathen and V. Shoup. Computing frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.

[Wal93] P. G. Walsh. The computation of puiseux expansions and a quantitative version of runge's theorem on diophantine equations. Ph.d. thesis, Univ. Waterloo, Waterloo, Candad, 1993.

[Wat82] D. S. Watkins. Understanding the QR algorithm. *SIAM Review*, 24:427–440, 1982.

[Wat91] D. S. Watkins. Some perspectives on the eigenvalue problem. *SIAM Review*, 35(3):430–471, 1991.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, England, 1965.

[Win96] F. Winkler. *Introduction to Computer Algebra.* Springer Verlag, Heidelberg, 1996.

[Wu84] W. Wu. Basis principles of mechanical theorem proving in elementary geometries. *J. Syst. Sci. and Math Sci.*, 4(3):207–235, 1984.

[Wu86] W. Wu. Basis principles of mechanical theorem proving in elementary geometries. *J. of Automated Reasoning*, 2:219–252, 1986.

[Zas69] H. Zassenhaus. On Hensel factorization I. *J. Number Theory*, 1:291–311, 1969.

[Zip93] R. Zippel. *Effective Polynomial Computations*, page 384. Kluwer Academic Publ., Boston, Massachusetts, 1993.