# APPLICATIONS OF FFT[1]

Ioannis Z. Emiris, INRIA Sophia-Antipolis, B.P. 93, 06902 France. emiris@sophia.inria.fr.
Victor Y. Pan, Mathematics and Computer Science Department, Lehman College, City University of New York, Bronx, NY 10468, USA. vpan@lcvax.lehman.cuny.edu.

## 1 Introduction

The subject of this chapter lies in the area of theoretical computer science, though it borrows certain results from computational mathematics, and is fundamental to the theory and practice of signal and image processing and scientific and engineering computing.

A central theme is to bridge the gap between polynomial arithmetic on the one hand, and integer arithmetic and matrix computations on the other. This is the premise of applying the Fast Fourier Transform (FFT) on a wide range of problems, yielding the fastest known algorithms for performing the basic arithmetic operations on integers, polynomials, and dense structured matrices. For instance, this chapter shows different ways of multiplying two univariate polynomials fast. This means that, instead of the classical method that requires a quadratic number of multiplications and additions between constants, we consider algorithms with significantly lower complexity in terms of the polynomial degrees. In particular, we detail an FFT-based approach, which requires a quasi-linear number of operations. Another example is a popular and very efficient method for performing arithmetic on modern computers over rationals of arbitrary size.

We state the complexity bounds under the random access machine (RAM) model of computation [Aho et al., 1974]. Typically, a unit cost is assigned to addition, subtraction, multiplication and division between real numbers, as well as to reading or writing into a memory location; this is the arithmetic model. In estimating the complexity of integer operations, we shall assign different costs to these basic operations, depending on the bit size of the involved parameters; this is the Boolean or bit model. The distinction between the two models shall be explicit or obvious from the context.

Section 2 examines the fundamental transforms between vectors, in particular, the discrete Fourier transform, its inverse, vector convolution, its extensions to wrapped convolutions, and the

[1]This material is based on work supported in part by the European Union under ESPRIT FRISCO project LTR 21.024 (first author), by the National Science Foundation, under Grants Nos. CCR-9020690 and CCR-9625344 (second author), and by PSC CUNY Awards Nos. 666327 and 667340 (second author).

sine and cosine transforms. The basic algorithm for computing these transforms is the Fast Fourier Transform (FFT), which is discussed in some detail and then applied to these problems.

Section 3 applies these results to some fundamental operations on univariate polynomials and shows the correlations between the main vector transforms and polynomial arithmetic. These results are carried over to the integers, then extended to the polynomials in several variables.

Section 4 examines structured matrices defined by significantly fewer elements than the full matrix size, typically, a linear function of the matrix dimension, instead of quadratic. Furthermore, we perform all the fundamental computations with such structured matrices in time quasi linear in the matrix dimension, which is a dramatic improvement over the quadratic or higher complexity estimates for the complexity of the same computations with arbitrary matrices. Such an improvement relies on exploiting correlations among structured matrices, fast polynomial arithmetic and FFT. Lastly, we examine some transformations among different classes of structured matrices and between computations with such matrices and some other major algebraic computational problems. Although this is a mere glimpse at the numerous applications of structured matrices, we hope to illustrate the richness of the subject.

We practically omit the lower bound topics, referring the reader to [Borodin and Munro, 1975, Knuth, 1997, Bürgisser et al., 1997] on some nontrivial results. The reader always may apply the obvious information lower bounds: since each arithmetic and each Boolean operand has two operations and one output, there must always be at least $\max\{O, I/2\}$ operations, where $O$ and $I$ are the sizes of the output and input, respectively. We also omit certain generalizations of Fourier transforms that are of some interest in theoretical computer science; see, for instance, [Bürgisser et al., 1997]. There are several important applications of FFT in engineering, such as signal and image processing and solving PDEs (see the end of Section 2.1 and the introduction of Section 4), which could not be covered in a chapter of the present size.

We try to cite books, surveys or comprehensive articles, rather than the publications that first contained a certain result. There is a tacit understanding that the interested reader will look into the bibliography of the cited references, in which the historical development is outlined.

Hereafter, log stands for $\log_2$ unless specified otherwise.

## 2 Some Fundamental Transforms

The transforms discussed here can be thought of as mappings that transform a given vector to another vector. They are fundamental because a variety of interesting and general problems can be solved by means of these transforms. More importantly, polynomial and integer arithmetic can be reduced to application of such transforms, and this reduction yields algorithms supporting the record estimates for the asymptotic computational complexity (see Section 3).

Section 2.1 defines the discrete Fourier transform (DFT) and some closely related transforms, and outlines the main algorithm for solving DFT, namely, the fast Fourier transform (FFT). The record complexity bounds for polynomial arithmetic, including multiplication, division, transformation under a shift of the variable, evaluation, interpolation, and approximating polynomial zeros, are based on FFT. The FFT and fast polynomial algorithms are the basis for many other fast polynomial computations, performed both numerically and symbolically. Section 2.2 studies vector convolution and shows its equivalence to DFT and also to generalized DFT. Section 2.3 recalls the sine, cosine and some other transforms.

Abundant further material and bibliography on transforms and convolution can be found in [Brigham, 1974, Elliott and Rao, 1982, Blahut, 1984, Clausen, 1989, Duhamel and Vetterli, 1990, Press et al., 1992, Van Loan, 1992, Bini and Favati, 1993, Bini and Pan, 1994].

### 2.1 The Discrete Fourier Transform and Its Inverse

The **discrete Fourier transform (DFT)** of the coefficient vector $\mathbf{p} = [p_0, \ldots, p_n]^T$ of a polynomial

$$p(x) = p_0 + p_1 x + \cdots + p_n x^n \tag{1}$$

is the vector $[p(1), \ldots, p(\omega^{K-1})]^T$, where $\omega$ is a primitive $K$-th root of 1, so that $\{1, \omega, \omega^2, \ldots, \omega^{K-1}\}$ is the set of all the $K$-th roots of 1 and $K = 2^k = n + 1$, for a natural $k$. For simplicity, the reader may assume the study of DFT in the complex field but it can extended to other fields, rings, and even groups, where the $K$-th roots of 1 are defined [Clausen, 1989, Cantor and Kaltofen, 1991, Farach et al., 1995, Cole and Hariharan, 1996]. The problem of computing the DFT is solved by applying the **fast Fourier transform (FFT)** algorithm, which is an important example of recursive algorithms based on the **divide-and-conquer method**. The FFT algorithm can be traced back at least to [Runge and König, 1924] and, to some extent, even to a work of K. F. Gauss of 1805, though its introduction in modern times has been credited to [Cooley and Tukey, 1965]

(see [Aho et al., 1974, Borodin and Munro, 1975, Zippel, 1993, Bini and Pan, 1994, Knuth, 1997] for details). To derive FFT, write

$$p(x) = q(x^2) + xs(x^2) = q(y) + xs(y) \ ,$$

where $y = x^2$, $q(y) = p_0 + p_2 y + \cdots + p_{n-1} y^{(n-1)/2}$, $s(y) = p_1 + p_3 y + \cdots + p_n y^{(n-1)/2}$, and the polynomials $q(y)$ and $s(y)$ have degree at most $(n-1)/2$. To evaluate $p(x)$ at $x = \omega^h$ for $h = 0, 1, \ldots, n$, we first compute $q(y)$ and $s(y)$ at $y = (\omega^2)^h$ and then $q(\omega^{2h}) + xs(\omega^{2h})$ at $x = \omega^h$. There exist only $K/2$ distinct values among all the integer powers of $\omega^2$ since $\omega^2$ is a $(K/2)$-nd root of 1. Half of the multiplications of $\omega^h$ by $s(\omega^{2h})$ can be saved because $\omega^i = -\omega^{i+K/2}$ for all $i$. By applying this method recursively, we arrive at the overall complexity bound of $1.5K \log K$ arithmetic operations, hereafter referred to as **ops**. Note that the old classical algorithm uses $2n^2$ ops to compute DFT, not including the cost of computing the roots of 1.

For demonstration, we describe FFT over the field of complex numbers for polynomial

$$p(x) = 3x^3 + 2x^2 - x + 5.$$

Here, $K = 4$, and the 4-th roots of unity are $1, \omega = \sqrt{-1}, \omega^2 = -1$, and $\omega^3 = -\omega$. We write

$$p(x) = (5 + 2x^2) + x(-1 + 3x^2) = q(y) + xs(y), \qquad y = x^2.$$

At the cost of performing 2 multiplications (by 1 and $\omega$) and 4 additions/subtractions, this reduces DFT for $p(x)$ to the evaluation of $q(y)$ and $s(y)$ at the points 1 and $\omega^2 = -1$. We compute $q(1), q(\omega^2), s(1)$, and $s(\omega^2)$, by using 2 multiplications (of the values 2 and 3 by 1) and 4 additions/subtractions: $5+2,\ 5-2,\ -1+3,\ -1-3$. Overall, 4 multiplications and 8 additions were required, which is indeed bounded by $1.5\,K\,\log K = 12$.

In actual computations, we may use complex approximations to $\omega^i = \exp(2\pi i\sqrt{-1}/K),\ i = 0, \ldots, K-1$ [Knuth, 1997]. In computations where all the $p_i$ are integers, we may perform the computations over the ring $Z_m$ of integers modulo an appropriate natural $m$ or use other special techniques (cf. [Winograd, 1980] and [Bini and Pan, 1994, sect. 1.7]). For appropriate choices of $m$, the desired $K$-th roots of 1 are readily available in $Z_m$ (see [Aho et al., 1974, pp. 265-269], [Borodin and Munro, 1975, pp. 86-87], or [Bini and Pan, 1994, sect. 3.3]). Performing FFT over finite fields or rings is also required in several applications, for instance, to integer multiplication (see [Bini and Pan, 1994, ch. 1]).

The **Inverse Discrete Fourier Transform (IDFT)** of a vector $\mathbf{r}$ of polynomial values $r_h = p(\omega^h),\ h = 0, 1, \cdots, K-1$, on a set of all the $K$-th roots of 1 is the coefficient vector $\mathbf{p} = [p_0, \cdots, p_n]^T$

of $p(x)$. Computing DFT and IDFT is a special case of the more general problems of multipoint polynomial **evaluation** and **interpolation** (Section 3.2), whose equivalence to structured matrix computations is analyzed in Section 4.1.

The IDFT can be computed in at most $K+1.5K \log K$ ops by means of applying the inverse FFT algorithm, which is again a divide-and-conquer procedure, reminiscent of FFT (see [Bini and Pan, 1994, p. 12]). Alternatively, we may reduce DFT and IDFT to each other and to matrix computations, based on the following vector equation: $\Omega \mathbf{p} = \mathbf{r}/\sqrt{K}$. Here, $\Omega = [\omega^{ij}/\sqrt{K}]$ is the Fourier matrix, $\mathbf{p} = [p_j]$ is the input vector of DFT, and $\mathbf{r} = [r_i]$, $i, j = 0, 1, \ldots, n$, where $\mathbf{r}/\sqrt{K}$ is the output vector $[p(\omega^h)]$ of DFT. It follows that $\Omega^{-1} = [\omega^{-ij}/\sqrt{K}]$ and $\mathbf{p} = \Omega^{-1}\mathbf{r}/\sqrt{K} = [\omega^{-ij}]\mathbf{r}/K$. Since $\omega^{-1}$ is a $K$-th root of 1, the latter matrix-by-vector product can be computed by means of an FFT; it will remain to divide the resulting vector by $K$ to complete the computation of the IDFT.

The problem of computing the generalized DFT at any number $K$ of points is defined as follows: Given a value $w$, having the reciprocal $w^{-1}$ but not necessarily being any root of 1, and a vector, $\mathbf{p} = (p_0, p_1, \ldots, p_{K-1})$, compute the vector $\mathbf{r}$ of the generalized DFT, $\mathbf{r} = (r_h)$, $r_h = \sum_{j=0}^{K-1} p_j w^{hj}$ for $h = 0, 1, \ldots, K-1$.

In Section 2.2, we will obtain the complexity bound of $O(K \log K)$ ops for computing generalized DFT for any $K$, by reducing the problem to computing vector convolution, defined in Section 2.2 and shown to be equivalent to the generalized DFT.

By using a variable shift (Section 3.1) and scaling of the variable, we may extend the solution to the generalized DFT so as to evaluate $p(x)$ on the set $\{ah^{2i} + fh^i + g,\ i = 0, 1, \ldots, n-1\}$ for any fixed 4-tuple of constants $(a, f, g, h)$, by using $O(n \log n)$ ops [Aho et al., 1975].

Then again, the bound $O(n \log n)$ is a dramatic improvement of the classical algorithms for IDFT and generalized DFT, which require order of $n^2$ ops. In practice of computations, DFT and IDFT are frequently computed on $K \geq 10000$ or even $K \geq 100000$ points, so the practical impact of FFT has been immense. There are several issues related to efficient implementation of FFT, as well as further techniques for reducing the complexity of computing the DFT and IDFT for specific smaller $K$, even though these algorithms do not decrease the asymptotic complexity estimates [Winograd, 1980, Van Loan, 1992, Bini and Bozzo, 1993]. There exist public domain codes implementing FFT freely accessible via netlib [Swarztrauber, 1984, Bailey, 1993, Bailey, 1993b, Frigo and Johnson], and certain libraries of arbitrary-precision integer arithmetic [Bailey, 1993b, Biehl et al., 1995, GNU, 1996] use FFT.

Some comments are in order on conditioning and numerical stability. It is fortunate that the DFT is a well-conditioned problem and that FFT is a numerically stable algorithm if we consider both input and output as vectors and measure the errors in terms of vector norms. More formally, let $\mathbf{x}$ and $\mathbf{y}$ be a pair of $K$-dimensional complex vectors, for $K = 2^k$, such that $\mathbf{y} = DFT(\mathbf{x})$ is the DFT of $\mathbf{x}$, let $FFT(\mathbf{x})$ denote the vector computed by applying the matrix version of the FFT algorithm described above and by using floating point arithmetic with $d$ bits, $d \geq 10$, and let $e_K(x)$ express the error vector of dimension $K$. Then, according to [Bini and Pan, 1994, prop. 3.4.1], we have

$$FFT(x) = DFT(x + e_K(x));$$
$$||e_K(x)|| \leq ((1 + \rho\, 2^{-d})^k - 1)||x||,\ 0 < \rho < 4.83,$$

where $||\cdot|| = ||\cdot||_2$ denotes the Euclidean norm. Moreover,

$$||e_K(x)|| \leq (5k)\ 2^{-d}||x|| \quad \text{for any} \quad K \leq 2^{2^{d-6}}.$$

It immediately follows ([Bini and Pan, 1994, cor. 3.4.1]) that

$$||FFT(x) - DFT(x)|| \leq 5\sqrt{K}(\log K)2^{-d}||x||,$$

if $K < 2^{2^{d-6}}$.

Similar results hold for IDFT and certain other computational problems reducible to computing DFT. In particular, we may apply FFT with a relatively low precision when we compute the product of two polynomials with integer coefficients (see Section 2.2), and then rounding off still gives us the output with no error. On the other hand, polynomial division and computing the greatest common divisor (GCD) and the zeros of polynomials (see Sections 3.1, 3.3 and Chapter 12) are generally ill-conditioned problems [Householder, 1970, Bini and Pan, 1994, Knuth, 1997].

The application of FFT to the computation of the continuous Fourier transform is central to several engineering and numerical applications [Geddes et al., 1992]. More specifically, there are important applications to digital filters, image restoration, and numerical solution of PDEs, which are not developed here; see, for instance, [Chan, 1996].

## 2.2 Vector Convolution

Another fundamental problem equivalent to computing the DFT, as well as the generalized DFT, is the computation of the **convolution**, as well as the positive and/or negative wrapped convolution,

of 2 vectors $u = (u_i)$ and $v = (v_i)$. Given the values $u_0, v_0, u_1, v_1, \ldots, u_n, v_n$, we seek the coefficients $w_i$, $w_i^+$ and/or $w_i^-$ of the polynomials

$$w(x) = \sum_{i=0}^{2n} w_i x^i, \quad w^+(x) = \sum_{i=0}^{n} w_i^+ x^i, \quad w_i^-(x) = \sum_{i=0}^{n} w_i^- x^i,$$

$$w_i^+ = w_i + \hat{w}_i, \quad w_i^- = w_i - \hat{w}_i,$$
$$w_i = \sum_{j=0}^{i} u_j v_{i-j}, \quad \hat{w}_i = w_{h+1+i} = \sum_{j=i+1}^{n} u_j v_{n+1+i-j}, \quad i = 0, 1, \ldots, n, \quad \hat{w}_n = 0.$$

In fact, $w(x) = u(x)v(x)$, *so that the coefficient vector of the product of 2 polynomials is the convolution* of their coefficient vectors, $w^+(x) = w(x) \bmod (x^{n+1}-1)$, $w^-(x) = w(x) \bmod (x^{n+1}+1)$, $u(x) = \sum_{j=0}^{n} u_j x^j$, $v(x) = \sum_{j=0}^{n} v_j x^j$.

We will reduce this problem essentially to the solution of a few DFT problems. Let $n+1 = K = 2^k$ for a natural $k$, for otherwise, we may pad $u(x)$ and $v(x)$ with $l$ zero leading coefficients each, for $l < n$, so as to bring the degree values to the form $K - 1 = 2^k - 1$, $k = \lceil \log_2 n \rceil$. By applying Toom's evaluation–interpolation techniques [Toom, 1963], we may evaluate at first $u(x)$ and $v(x)$ at the $(2K)$-th roots of 1 (2 FFTs), then multiply the $2K$ computed values pairwise, which gives us the values of $w(x)$ at the $(2K)$-th roots of 1, and then obtain the coefficients of $w(x)$ (via IDFT) at the overall cost of $9K \log K + 2K$ ops. By reducing $w(x)$ modulo $(x^{n+1} \pm 1)$, we may also obtain $w^+(x)$ and $w^-(x)$.

Let us compute $w^+$ by performing two DFTs and an IDFT at $K$-th roots of 1. Consider the values of $w^+(x)$ at $\omega^h$, where $\omega$ is a primitive $(n+1)$-st root of unity and $h = 0, \ldots, n$. Applying $\hat{w}_n = 0$, we have

$$w^+(\omega^h) = \sum_{i=0}^{n} (w_i + \hat{w}_i)\omega^{ih} = \sum_{i=0}^{n} \omega^{ih} \sum_{j=0}^{i} u_j v_{i-j} + \sum_{i=0}^{n-1} \omega^{ih} \sum_{j=i+1}^{n} u_j v_{n+1+i-j}.$$

Since $\omega^{n+1} = 1$, the second summand can be written as follows:

$$\left(\omega^{n+1}\right)^h \sum_{i=0}^{n-1} \omega^{ih} \sum_{j=i+1}^{n} u_j v_{n+1+i-j} = \sum_{i=n+1}^{2n} \omega^{ih} \sum_{j=0}^{i} u_j v_{i-j},$$

where the second summand is simplified by the fact that $u_j = v_j = 0$ for $j < 0$. Let $w(x)$ be the product polynomial $u(x)v(x)$. Then $w^+(\omega^h) = w(\omega^h)$, $h = 0, \ldots, n$, and we may recover $\omega^+(x)$, by means of Toom's technique, at the cost of 3 FFTs, each on the set of $(n+1)$-st roots of 1. Therefore, computing the positive wrapped convolution has complexity of $4.5K \log K + K$ ops.

A similar argument applies to the negative wrapped convolution. Let $\psi$ be a $(2n+2)$-nd primitive root of unity and let $\omega$ and $h$ be as above. Then $w^-(x) = u(x)v(x)$, for $x = \psi\omega^h$, since $x^{n+1} = -1$, and it suffices to use 3 FFTs, each on $n+1$ points $\psi\omega^h$, at the overall cost $4.5K \log K + K$.

Conversely, let us reduce the generalized DFT problem to convolution. We seek

$$r_h = \sum_{j=0}^{K-1} p_j w^{hj} = w^{-h^2/2} \sum_{j=0}^{K-1} p_j w^{(j+h)^2/2} w^{-j^2/2}, \quad \text{for} \quad h = 0, \ldots, K-1.$$

Let $w_i = r_h w^{h^2/2}$, $v_{i-j} = w^{(j+h)^2/2}$ and $u_j = p_j w^{-j^2/2}$, for $i = K-1, \ldots, 2K-2$ and $j = 0, \ldots, K-1$. Essentially, we change indices by setting $i = 2K-2-h$. The undefined values of $u_j$ are zero; in particular, $u_j = 0$ for $j \geq K$. Then,

$$w_i = \sum_{j=0}^{K-1} u_j v_{i-j} + \sum_{j=K}^{i} u_j v_{i-j}, \qquad i = K-1, \ldots, 2K-2,$$

which is a part of the convolution problem $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, where $i = 0, \ldots, 2K-2$ and $u_s = v_s = 0$ for $s \geq K$. Hence, the generalized DFT can be reduced to two wrapped convolutions, followed by $2K$ multiplications for computing the $u_j$ and recovering $r_h$ from $w_i$. Therefore, the *asymptotic complexity of generalized DFT* is $O(K \log K)$, *the same as for DFT and (wrapped) convolutions.*

More generally, $4.5K \log K + 2K$ ops suffice for computing the convolution of two vectors of lengths $m+1$ and $n+1$, respectively, which is the coefficient vector of the product of two polynomials of degrees at most $m$ and $n$ with given coefficient vectors, where $K = 2^k$ and $k = \lceil \log(m+n+1) \rceil$. The problem and the solution are immediately extended to multiplication of several polynomials [Bini and Pan, 1994].

For smaller $m$ and $n$, the convolution and the associated polynomial product can be alternatively computed by means of the straightforward (classical) algorithm that uses $(m+1)(n+1)$ multiplications and $(m+1)(n+1) - m - n - 1$ additions.

For moderate $m$ and $n$, $m \leq n$, one may prefer the alternative algorithm of [Karatsuba and Ofman, 1963], where we assume for simplicity that $n$ is a power of 2 and rely on the recursive application of the following equations:

$$\begin{aligned} &u(x)v(x) = \left(u_0(x) + x^{n/2}u_1(x)\right)\left(v_0(x) + x^{n/2}v_1(x)\right) = \\ &u_0(x)v_0(x)\left(1 - x^{n/2}\right) + \left(u_1(x) + u_0(x)\right)\left(v_1(x) + v_0(x)\right)x^{n/2} + u_1(x)v_1(x)\left(x^n - x^{n/2}\right). \end{aligned} \tag{2}$$

The algorithm uses $O(n^{\log 3})$ ops, where $\log 3 = 1.5849\ldots$, and has a small overhead constant.

## 2.3 Sine and Cosine Transforms

Besides FFT, other related transforms widely used in signal processing include sine, cosine, Hartley and wavelet transforms. Given a vector $\mathbf{y} = [y_1, \ldots, y_n]$, its sine transform can be defined as the vector $\mathbf{x} = [x_1, \ldots, x_n]$, where

$$x_i = \sum_{j=1}^{n} y_j \sin \frac{\pi ij}{n+1}, \quad i = 1, \ldots, n,$$

or, equivalently, $\mathbf{x} = \sqrt{\frac{n+1}{2}} Sy$, where $S = [\sqrt{\frac{2}{n+1}} \sin \frac{\pi ij}{n+1}]$, $i, j = 1, \ldots, n$, $S^T = S^{-1} = S$. The cosine transform of vector $[y_j]$ can be defined analogously by substituting sine by cosine. For further variants of sine and cosine transforms see [Press et al., 1992, Kailath and Olshevsky, 1996].

The sine and cosine transforms can be performed in $O(n \log n)$ ops by means of FFT [Kailath and Olshevsky, 1996].

An important application of the sine transform is in computations with the matrix algebra $\tau$, introduced in [Bini and Capovani, 1983]; further applications can be found in [Huckle, 1996]. This algebra consists of all the $n \times n$ matrices $A = [a_{ij}]$, such that

$$a_{i,j-1} + a_{i,j+1} = a_{i-1,j} + a_{i+1,j}, \quad i, j = 0, 1, \ldots, n-1;$$
$$a_{i,j} = 0 \text{ if } i \in \{-1, n\}, \text{ or } j \in \{-1, n\}.$$

Then $A$ is also denoted $\tau(a)$, where $a$ is vector $[a_{0,0}, a_{1,0}, \ldots, a_{n-1,0}]$, i.e., the first column of $A$. Several properties of this matrix algebra, including its connection to Chebyshev-like polynomials, are discussed in [Bini and Pan, 1994, ch. 2].

For any matrices $A \in \tau$ and $S$ defined above, $SAS = D$ is a diagonal matrix with nonzero entries $d_1, \ldots, d_n$, given by $d_i = (Sa)_i / (\sqrt{2/(n+1)} \sin \frac{\pi i}{n+1})$, where $a$ denotes the first column of matrix $A$. Furthermore, given two vectors $u$, $v$, each having $n$ components, the following vectors can be computed in $O(n \log n)$ ops, by means of a constant number of sine transforms: $\tau(u)v$, the first column of $\tau(u)\tau(v)$, and $\tau(u)^{-1}v$, if the matrix $\tau(u)$ is nonsingular. The matrix algebra $\tau$ is important in the analysis of the spectral properties of band Toeplitz matrices and the computation of their tensor rank, as well as in solving Toeplitz and block Toeplitz linear systems (see Section 4.2).

The reader is referred to [Strang, 1993, Bini and Favati, 1993] on wavelet and Hartley transforms.

# 3 Fast Polynomial and Integer Arithmetic

Computations with integers and polynomials, in one or more variables, is of fundamental importance in computational mathematics and computer science. Such operations lie at the core of every computer algebra system. In this section, we focus on univariate polynomials and show some extensions to integer arithmetic and multivariate polynomials. The connection between vector manipulation, on the one hand, and univariate polynomial and integer arithmetic, on the other hand, relies on representing a polynomial or an integer by a vector of coefficients or digits, respectively. A large number of problems in science and engineering can be solely expressed in terms of polynomials. In addition, polynomials serve as the basis for the study of more complex structures, such as rational functions, algebraic functions, power series and transcendental functions.

The first theme of this section is to apply the transforms seen so far in order to yield the record asymptotic upper bounds on the complexity of several fundamental operations with univariate polynomials. In exploiting the power of FFT, we are implicitly assuming that the polynomials are dense, in other words, most of their coefficients are nonzero and, therefore, it makes sense to represent them by the vector of their coefficients. Different representations of sparse polynomials (for instance, by the straight-line programs for their evaluation) and different complexity measures (for instance, in terms of the output size) can be found in [Aho et al., 1974, Buchberger et al., 1982, Kaltofen, 1989, Zippel, 1993].

In particular, we cover polynomial multiplication, division, evaluation and interpolation, as well as further extensions of these basic operations. Our study shows that all of these problems have the same asymptotic complexity within a logarithmic factor, which is due to their correlation to DFT, IDFT, and vector convolution.

Some algorithms for manipulating univariate polynomials can be adapted to performing the analogous operation over the integers and vice versa (see Section 3.4).

Lastly, Section 3.5 examines the case of polynomials in more than one indeterminates.

## 3.1 Multiplication, Division, and Variable Shift

Multiplication of univariate polynomials is an important and fundamental operation, to which all other important operations can be ultimately reduced. This section also studies division of polynomials with remainder and computing the reciprocal of a polynomial. We show that both of these operations, as well as the transformation of a polynomial under a shift of the variable, have the

same asymptotic complexity as multiplication. The "linear" operations of addition, subtraction, and multiplication by a constant for degree $n$ polynomials can be performed in $n+1$ ops.

Polynomial multiplication is the problem of computing the coefficients $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i = 0, 1, \ldots, m+n$, of the polynomial $w(x) = u(x)v(x)$ provided that we are given the coefficients of a pair of polynomials $u(x)$ and $v(x)$,

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

and that $u_j = 0$, for $j > m$, $v_k = 0$, for $k > n$. We have already solved this problem, in $O(K \log K)$ ops, for $K = m+n$, in Section 2.2 as the problem of computing the convolution of 2 vectors.

Given polynomials $u(x)$ and $v(x)$ of degrees $m$ and $n$, respectively, polynomial division with a remainder is the computation of the coefficients of the unique pair of polynomials $q(x) = \sum_{g=0}^{m-n} q_g x^g$ (quotient) and $r(x) = \sum_{h=0}^{n-1} r_h x^h$ (remainder) such that

$$u(x) = q(x)v(x) + r(x), \quad \deg r(x) < n.$$

$r(x)$ is also called the **residue** of $u(x)$ modulo $v(x)$ and is denoted $u(x) \bmod v(x)$.

A related problem is the computation of the reciprocal of a polynomial. Given a natural $n$ and a polynomial $u(x)$, $u(0) \neq 0$, compute the first $n$ coefficients of the formal power series $w(x)$ such that $w(x)u(x) = 1$ or, equivalently, compute $w(x) \bmod x^n$. Both problems of polynomial division and computation of the reciprocal can be reduced to polynomial multiplication [Aho et al., 1974, Borodin and Munro, 1975, Bini and Pan, 1994] and solved in $O(n \log n)$ ops.

Another reduction of polynomial division to FFT is direct, by means of Toom's techniques of evaluation-interpolation with no reduction to polynomial multiplication. This is easily done where it is known that $r(x) \equiv 0$, but in [Pan et al., 1992] such a reduction to FFT has been elaborated for parallel approximate division of any pair of polynomials.

So far, we have assumed computations in the fields that support $O(n \log n)$ reduction to FFT. For an arbitrary ring of constants with unity, only the reduction to FFT in $O(n \log n \log \log n)$ is supported, so that the cited complexity estimates for these operations with polynomials are multiplied by an $O(\log \log n)$ factor, thus yielding an $O(n \log n \log \log n)$ overall bound [Cantor and Kaltofen, 1991]. Over any ring of constants, however, multiplication, division and the reciprocal computation have complexities related by constant factors, since each of them can be solved by means of a constant number of applications of any one of the others [Bini and Pan, 1994]. Hereafter, we will write $M(n)$ to de-

note the asymptotic arithmetic complexity of solving these problems, where $n$ is the degree of the input polynomials.

Shifting the variable, also known as a **Taylor shift**, is the problem of computing the coefficients $q_0(\Delta)$, $q_1(\Delta), \ldots, q_n(\Delta)$ of the polynomial

$$q(y) = p(y + \Delta) = \sum_{h=0}^{n} p_h (y + \Delta)^h = \sum_{g=0}^{n} q_g(\Delta) y^g,$$

given a scalar $\Delta$ and the coefficients $p_0, p_1, \ldots, p_n$ of a polynomial $p(x)$. This problem is solved in $O(M(n))$ ops [Aho et al., 1975, Bini and Pan, 1994].

The practical impact of using FFT and the fast convolution algorithm in computer algebra, however, is more limited than one may suspect, for several reasons. In many cases, the application of the fast convolution algorithm requires special care in order to contract the resulting growth of the precision needed in order to represent the auxiliary parameters [Fateman, 1974]. In addition to the classical algorithms, supporting the bound $M(n) = O(n^2)$, and one of [Karatsuba and Ofman, 1963], supporting the bound $M(n) = O(n^{\log 3})$, strong competitors to FFT and fast convolution are the algorithms based on the techniques of binary segmentation (see Chapter 12), which is effective for computations in finite fields and in other cases where the input parameters are represented by short binary integers.

## 3.2 Evaluation, Interpolation and Chinese Remainder Computations

Historically, the first topic in the algebraic computational complexity theory is the evaluation of a polynomial at a single point. Given the coefficients of a polynomial

$$p(x) = p_0 + p_1 x + \cdots + p_n x^n,$$

compute its value at a point $x_0$. Newton's optimal solution, known as Horner's rule, is based on the decomposition

$$p(x_0) = (\cdots((p_n x_0 + p_{n-1})x_0 + p_{n-2})x_0 + \cdots + p_1)x_0 + p_0$$

and uses $n$ multiplications and $n$ additions [Aho et al., 1974, Borodin and Munro, 1975, Knuth, 1997], though about 50% of multiplications can be saved if we allow to precondition the coefficients and if we count only operations depending on $x$ [Knuth, 1997].

More generally, given the coefficients $p_i$ of a polynomial $p(x)$, one may need to evaluate $p(x)$ on a fixed set of points $\{x_0, \ldots, x_{K-1}\}$. The problem can be reduced to $K$ applications of Horner's

rule and solved in $2Kn$ ops. Alternative reduction to polynomial multiplications and divisions yields an $O(M(m)\log m)$ algorithm, where $m = \max\{n, K\}$. The cost turns into $O(m\log^2 m)$ provided that $M(m) = O(m\log m)$, where polynomial arithmetic is implemented based on FFT. We will show the two stages of this approach referring to [Aho et al., 1974, Borodin and Munro, 1975, Bini and Pan, 1994] for further details.

Assume that $K \le 2^k \le n+1$. Note that $p(x_i) = p(x) \bmod (x - x_i)$, $i = 0, \ldots, K-1$. The first stage is based on the fan-in method. Successively compute the polynomials $m_i^{(j)} = m_{2i}^{(j-1)} m_{2i+1}^{(j-1)}$, for $i = 0, \ldots, \dfrac{K}{2^j} - 1$ and $j = 1, \ldots, k-1$, starting with the given moduli $m_i^{(0)} = x - x_i$ for all $i$. The degree of $m_i^{(j)}$ in $x$ is $2^j$, so that for a fixed $j$ and given the $m_i^{(j-1)}$, we compute all $m_i^{(j)}$ in $2^{k-j}M(2^j)$ ops.

The second stage is called the fan-out method and is another instance of the divide-and-conquer approach. Since $m_0^{(k-1)} = \prod_{i=0}^{K/2-1} m_i$ and $m_1^{(k-1)} = \prod_{i=K/2}^{K-1} m_i$, we have

$$p(x) \bmod (x - x_i) = \left(p(x) \bmod m_0^{(k-1)}\right) \bmod (x - x_i),\ i = 0, \ldots, K/2 - 1, \qquad \text{and}$$
$$p(x) \bmod (x - x_i) = \left(p(x) \bmod m_1^{(k-1)}\right) \bmod (x - x_i),\ i = K/2, \ldots, K - 1.$$

Thus, we have reduced the evaluation of $p(x)$ to the evaluation of two polynomials of roughly half degree, by means of two polynomial divisions. This stage continues recursively and has the same asymptotic complexity as the fan-in stage since polynomial division and multiplication have the same asymptotic complexity.

The overall complexity is $O(M(n)\log K)$. The algorithm can be adapted to the case $K > n+1$ with complexity $O((K/n)M(n)\log K)$. Hence the bound $O(M(m)\log m)$, stated earlier, for $m = \max\{n, K\}$, which grows to $O(m^{\log 3}\log m)$, $\log 3 = 1.5849\ldots$, if, instead of FFT, we apply the method of [Karatsuba and Ofman, 1963] for polynomial multiplication.

In short, *evaluation on a set of points reduces to multiplications*, which can be implemented based on DFT. The computation of the latter is, obviously, a special case of the evaluation problem. Thus, all basic problems seen so far can be *reduced to any one of them*, at most with a polylogarithmic asymptotic overhead. This is also the case with interpolation to a function by a polynomial, which is the inverse of the polynomial evaluation problem and is stated as follows: Given two sets of values,

$$\{x_i \ :\quad i = 0, \ldots, n;\quad x_i \ne x_j \text{ for } i \ne j\}\ ,\quad \{r_i \ : i = 0, \ldots, n\},$$

evaluate the coefficients $p_0, p_1, \ldots, p_n$ of the polynomial $p(x)$ of Equation (1) satisfying $p(x_i) = r_i, i = 0, 1, \ldots, n$. The problem always has a unique solution, which the classical interpolation

algorithms compute in $O(n^2)$ ops [Conte and de Boor, 1980, Golub and Van Loan, 1996]. However, by using FFT, $O(n \log^2 n)$ ops suffice. This bound is optimal up to the factor $O(\log n)$ [Borodin and Munro, 1975, Knuth, 1997, Bürgisser et al., 1997].

The fast algorithm [Fiduccia, 1987, Bini and Pan, 1994] uses the Lagrange interpolation formula:

$$p(x) = L(x) \sum_{i=0}^{n} \frac{r_i}{L'(x_i)(x - x_i)}, \qquad \text{where} \qquad L(x) = \prod_{i=0}^{n} (x - x_i), \tag{3}$$

$L'(x_i) = \prod_{k=1,\ k \neq i}^{n} (x_i - x_k)$ for $i = 0, 1, \ldots, n$, and $L'(x)$ is the formal derivative of $L(x)$. Interpolation then reduces to application of, at first, the fan-in method to computing the polynomials $L(x)$ and $L'(x)$, secondly, the evaluation algorithm to finding the values $L'(x_i)$ for all $i$, and, thirdly, polynomial multiplications to obtain $p(x)$ from (3). The overall cost is $O(M(n) \log n)$, where $M(n) = O(n \log n)$ is used.

A generalization of interpolation is the Chinese remainder problem. Here, we shall examine its univariate polynomial version, though its name comes from its application to the integers (cf. Section 3.4), by Chinese mathematicians in the 2nd century A.D. or even earlier [Knuth, 1997]. Let us be given the coefficients of $2h$ polynomials $m_i(x), r_i(x),\ i = 1, \ldots, h$, where $\deg m_i(x) > \deg r_i(x)$ and where the polynomials $m_i(x)$ are pairwise relatively prime, that is, pairwise have only constant common divisors or, equivalently, $\gcd(m_i, m_j) = 1$ when $i \neq j$. Then, we are asked to compute the unique polynomial $p(x) = p(x) \bmod \prod_{i=1}^{h} m_i(x)$, such that $r_i(x) = p(x) \bmod m_i(x),\ i = 1, \ldots, h$. When every $m_i(x)$ is of the form $x - x_i$, we come back to the interpolation problem.

The Chinese remainder theorem states that there always exists a unique solution to this problem. The importance of this theorem cannot be overestimated, since it allows us to reduce computation with a high-degree polynomial to similar computations with several smaller-degree polynomials, namely, to computations modulo each $m_i(x)$. Then the **Chinese remainder algorithm** combines the results modulo each $m_i(x)$ so as to yield $p(x)$.

There are two approaches to recovering $p(x) = p(x) \bmod \prod_{i=1}^{h} m_i(x)$ from given $r_i(x)$ and $m_i(x)$, $i = 1, \cdots, h$. The first uses essentially Lagrange's interpolation formula, simply by generalizing the polynomial values in formula (3) to polynomial remainders. The second approach is named after Newton and is incremental, in the sense that successive steps compute $p_{k-1}(x) = p(x) \bmod \prod_{i=1}^{k} m_i(x)$, for $k = 1, 2, \ldots, h$, where the last step gives the final result:

$$p_k(x) = p_{k-1}(x) + ((r_k(x) - p_{k-1}(x)) s_k(x) \bmod m_k(x)) \prod_{i=1}^{k-1} m_i(x),$$
$$\text{where} \quad s_k(x) \prod_{i=1}^{k-1} m_i(x) \bmod m_k(x) = 1, \quad k = 1, 2, \ldots, h, \quad p_h(x) = p(x).$$

Let $N = \sum_{i=1}^{h} d_i,\ d_i = \deg m_i(x)$. Then the overall complexity of computing $p_h(x) = p(x)$ by this algorithm is $O(M(N)h + \sum_{i=1}^{h} M(d_i)\log d_i)$, which, based on FFT, turns into $O(Nh \log N)$.

Further details can be found in [Aho et al., 1974, Borodin and Munro, 1975, Buchberger et al., 1982, Zippel, 1993, Bini and Pan, 1994, Knuth, 1997].

### 3.3 GCD, LCM, and Padé Approximation

In this section, we will study the computation of the **greatest common divisor (GCD)** and the **least common multiple (LCM) of two polynomials**, and we will list some of their numerous applications to polynomial and rational computations.

The classical solution method is the Euclidean algorithm, which is a major general tool for many algebraic and numerical computations. Despite its ancient origins, the problem of computing greatest common divisors, with its numerous facets, is still an active area of research.

Given the coefficients of two polynomials

$$u(x) = \sum_{i=0}^{m} u_i x^i,\ \ v(x) = \sum_{j=0}^{n} v_j x^j,\ \ m \geq n \geq 0,\ \ u_m v_n \neq 0,$$

their greatest common divisor, denoted $\gcd(u(x), v(x))$, is a common divisor of $u(x)$ and $v(x)$ having the highest degree in $x$. The GCD of $u(x)$ and $v(x)$ is unique up to within constant factors, or it can be assumed monic, and then it is unique. For example,

$$\gcd(x^5 + x^4 + x^3 + x^2 + x + 1, x^4 - 2x^3 + 3x^2 - x - 7) = x + 1,$$

because $x + 1$ divides both polynomials and they have no common divisor of degree greater than or equal to two.

**Algorithm 3.1 (Euclid's).** *Set* $u_0(x) = u(x),\ v_0(x) = v(x)$. *Compute*

$$u_{i+1}(x) = v_i(x),$$

$$v_{i+1}(x) = u_i(x) \bmod v_i(x) = u_i(x) - q_{i+1}(x)v_i(x),\ \ i = 0, 1, \ldots, \ell - 1, \tag{4}$$

where $q_{i+1}(x)$ is the quotient polynomial, and $\ell$ is such that $v_\ell(x) = 0$. At the end of this process, $u_\ell(x) = \gcd(u(x), v(x))$.

The correctness of the algorithm can be deduced from the following equation:

$$\gcd(u_i(x), v_i(x)) = \gcd(u_{i+1}(x), v_{i+1}(x)),$$

which holds for all $i$. Euclid's algorithm only involves arithmetic operations, so that the output coefficients of the GCD are rational functions in the input coefficients. Assuming that $m = n$, the algorithm involves $O(n^2)$ ops, but using a matrix representation of the recurrence and the fan-in method yields an $O(M(n)\log n)$ bound [Bini and Pan, 1994].

The sequence $v_1(x), v_2(x), \ldots, v_l(x)$ is called a polynomial remainder sequence. It can be generalized to the sequence of remainder polynomials obtained if the division step (4) is substituted by

$$a_{i+1}v_{i+1}(x) = b_i u_i(x) \bmod v_i(x) = b_i u_i(x) - q_{i+1}(x)v_i(x),\ i = 0, 1, \ldots, \ell - 1,$$

where the $a_{i+1}$ and $b_i$, for $i = 0, \ldots, \ell - 1$, are scalars. When all scalars are units we recover the original equation. The polynomial pseudo-remainder sequence is obtained by setting $a_{i+1} = 1$ and $b_i = c_i^{d_i}$, for $i = 0, \ldots, \ell - 1$, where $c_i$ is the leading coefficient of $v_i(x)$ and $d_i = \deg u_i(x) - \deg v_i(x) + 1$. There exists the third remainder sequence, called the subresultant sequence, which is closely related to the Sylvester resultant (cf. Chapter 12). These variations aim at palliating the swell of the intermediate coefficients. For details, consult [Aho et al., 1974, Buchberger et al., 1982, Kaltofen, 1989, Zippel, 1993, Bini and Pan, 1994, Diaz and Kaltofen, 1995].

A closely related problem is the least common multiple (LCM) computation. Given the coefficients of the two polynomials $u(x)$ and $v(x)$, compute the coefficients of $\mathrm{lcm}(u(x), v(x))$, that is, of a common multiple of $u(x)$ and $v(x)$ having the minimum degree. In the previous example, the LCM is $x^8 - 2x^7 + 4x^6 - 3x^5 - 3x^4 - 3x^3 - 4x^2 - x - 7$. Given $u(x)$, $v(x)$, and their GCD, we may immediately compute

$$\mathrm{lcm}(u(x), v(x)) = \frac{u(x)v(x)}{\gcd(u(x), v(x))}.$$

The cited references for the GCD problem also present alternative algorithms for the LCM; the best asymptotic complexity bound is $O(M(n)\log n)$, if $n \geq m$.

In the $(m, n)$ Padé approximation problem, we are given two natural numbers $m$ and $n$ and the first $N + 1 = m + n + 1$ Taylor coefficients of an analytic function $V(x)$ decomposed at $x = 0$, and we are seeking two polynomials $R(x)$ and $T(x)$ satisfying the relations

$$R(x) - T(x)V(x) = 0 \bmod x^{N+1},\ N = m + n,\ \deg T(x) \leq n,\ \deg R(x) \leq m.$$

This is actually a special case of Hermite's interpolation problem (cf. [Bini and Pan, 1994] on both subjects). Its complexity is $O(M(N)\log N)$ ops. An alternative solution of [Brent et al., 1980] relying on Toeplitz computations (discussed in Section 4.2) requires $O(N\log^3 N)$ ops but leads to a faster parallel algorithm [Bini and Pan, 1994].

A well-known application of Padé approximation is to computing the *minimum span for a linear recurrence*, also known as the Berlekamp-Massey problem and having important applications to algebraic coding theory, sparse polynomial interpolation, and parallel matrix computations [Berlekamp, 1968, Grigoryev et al., 1990, Kaltofen et al., 1990, Zippel, 1993, Bini and Pan, 1994]. Given a natural $s$ and $2s$ numbers $v_0, \ldots, v_{2s-1}$, compute the minimum natural $n \leq s$ and $n$ numbers $t_0, \ldots, t_{n-1}$ such that $v_i = t_{n-1}v_{i-1} + \cdots + t_0 v_{i-n}$, for $i = n, n+1, \ldots, 2s-1$. Its solution can be obtained by extending the solution of Padé approximation problem and requires $O(M(s)\log s)$ ops [Berlekamp, 1968, Brent et al., 1980, Pan, 1997].

### 3.4 Integer Arithmetic

This section carries over the results seen so far for univariate polynomials to the domain of integers. The basic premise of this correlation, which goes in both directions, is that any binary integer can be thought of as a polynomial with coefficients in $\{0, 1\}$. For instance, the binary representation of 24 is 11000, which corresponds to the univariate polynomial $u(x) = x^4 + x^3$, which has degree 4 and the coefficient vector $[1, 1, 0, 0, 0]$. One difference is that operating with integers we must take care of the carry bits.

As a result, the algorithms for some basic operations on univariate polynomials can be applied to the integers and vice versa, and in many cases the complexity estimates do not change significantly when we shift from arithmetic operations with polynomials to binary operations with integers [Aho et al., 1974, Bini and Pan, 1994]. This is illustrated below for the multiplication and GCD algorithms. This section also emphasizes the role of Chinese remaindering, which, as we mentioned already, has been historically introduced in the context of integers but also applies to polynomials. Likewise, the algorithms of Section 3.1 for polynomial multiplication can be applied to integer multiplication modulo $2^N + 1$, whose complexity we denote $\mu(N)$. Assume that for a given pair of $N$-bit integers $u$ and $v$,

$$u = \sum_{i=0}^{N-1} u_i 2^i, \quad v = \sum_{i=0}^{N-1} v_i 2^i, \quad u_i, v_i \in \{0, 1\},$$

we seek the integer $uv$. The classical algorithm has complexity $O(N^2)$. [Karatsuba and Ofman, 1963] recursively applies the equation:

$$uv = U_0 V_0 (1 - 2^{N-1}) + (U_1 + U_0)(V_1 + V_0)2^{N-1} + U_1 V_1 (2^{2N-2} - 2^{N-1}),$$

where $u = U_0 + 2^{N-1}U_1$ and $v = V_0 + 2^{N-1}V_1$; see Equation (2). This algorithm uses $O(N^{\log 3})$ Boolean operations, where $\log 3 = 1.5849\ldots$, and has a small overhead constant. The evaluation-interpolation idea demonstrated for polynomial multiplication enable us to reduce the complexity to $O(N^{1+\epsilon})$ for any positive $\epsilon$. Finally, by exploiting fast convolution by means of FFT over finite rings of constants, Schönhage and Strassen reduced the asymptotic complexity bound to $O(N \log N \log\log N)$; see [Schönhage and Strassen, 1971, Schönhage et al., 1994], [Aho et al., 1974, pp. 270–274], or [Bini and Pan, 1994, pp. 78–79]. Only the (obvious) information lower estimate of order $N$ is known for $\mu(N)$.

In practice, the Schönhage-Strassen algorithm is used only for very large $N$, for instance, in applications to polynomial root-finding, because the overhead constant is considerable. At present, the algorithms used in practice otherwise are either the classical method or the one of [Karatsuba and Ofman, 1963]. Recently, however, in addition to the two latter algorithms, the algorithm of [Schönhage and Strassen, 1971] has been implemented on modern computers [Biehl et al., 1995, GNU, 1996].

Integer division is the problem where, given two positive integers $u, v$ with the bit sizes $n, m$, respectively, we seek the unique pair of integers $q, r$ for which $u = qv + r$, where $0 \le r < v$. The classical algorithm has Boolean complexity $O(mn)$, whereas the fast multiplication algorithm yields the $O(\mu(m))$ bound [Aho et al., 1974, Borodin and Munro, 1975, Knuth, 1997]. The asymptotic complexity of integer multiplication and division is actually the same [Aho et al., 1974, Bini and Pan, 1994].

Given two **integers** $u$ and $v$, their **greatest common divisor (GCD)** is the largest integer that divides both $u$ and $v$, and their **least common multiple (LCM)** is the smallest integer that is divisible by both $u$ and $v$. For instance, $\gcd(16, 24) = 8$, whereas $\mathrm{lcm}(16, 24) = 48$. The Euclidean algorithm was the historically first algorithm for integer GCD. The GCD of a pair of positive integers less than $2^n$ can be computed in $O(\mu(n) \log n) = O(n \log^2 n \log\log n)$ Boolean ops, where $\mu(n)$ denotes the Boolean complexity of multiplying two integers modulo $2^n + 1$ [Aho et al., 1974, Hardy and Wright, 1979, Bini and Pan, 1994, Knuth, 1997].

Likewise, historically, the Chinese remainder algorithm has been first devised for integers. Given the **integer residues** $r_i$ with respect to fixed integer moduli $m_i$, for $i = 0, 1, \ldots, k$, where $\gcd(m_i, m_j) = 1$ for $i \ne j$, we seek an integer $p = p \bmod \prod_{i=0}^{k} m_i$ such that $r_i = p \bmod m_i$ for all $i$. The Chinese remainder theorem states that such an integer $p$ exists and is unique. The Boolean

complexity of computing such an integer $p$ is $O(\mu(N)\log k)$ ops, where $N = \sum_i \lceil \log m_i \rceil$.

An important application of the Chinese remainder theorem is in reducing the bulk of an arbitrary-precision integer computation to computations with fixed-precision integers. We map the input integers into their residues moduli $m_i$, then perform the computation in the finite field or ring of integers modulo $m_i$ for each $i$, and, finally, use the Chinese remainder algorithm in order to compute the exact answer (see [Emiris, 1997] for extensions).

The above technique of modular arithmetic is one of the most efficient methods for conducting integer (as well as rational) arithmetic on modern computers. Typically, the moduli used are primes that fit in a computer word. The implementation of the Chinese remainder algorithm relies either on an extension of Lagrange's formula (3) or on Newton's incremental approach (cf. Section 3.2). For a comprehensive discussion as well as other alternatives, see [Aho et al., 1974, Borodin and Munro, 1975, Buchberger et al., 1982, Zippel, 1993, Bini and Pan, 1994, Knuth, 1997].

## 3.5 Multivariate Polynomials

Polynomials in several variables generalize the univariate case, which we have examined so far. Multivariate polynomials appear in a wide variety of scientific and engineering applications (cf. Chapter 12 of this handbook). This section outlines some basic results regarding multiplication, evaluation and interpolation of multivariate polynomials over arbitrary rings of constants.

Just as we did in the univariate case, we do not consider in depth the question of representation, but assume that polynomials are represented by a coefficient vector indexed by the corresponding monomials in some order. Other operations, such as addition, subtraction, taking integer powers and division, as well as the various representations are treated extensively in [Buchberger et al., 1982, Zippel, 1993].

A polynomial in $n$ variables is of the form

$$p(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_n} p_{i_1,i_2,\ldots,i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n},$$

where the coefficients correspond to distinct monomials. Recall that the degree of $p$ in $x_j$ is the maximum $i_j$ for which $p_{i_1,i_2,\ldots,i_n} \neq 0$, the total degree of a monomial $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$ is $i_1 + \cdots + i_n$, and the total degree of the polynomial $p$ is the maximum total degree of any monomial. If $d$ is the maximum degree in any variable, then the total number of terms is $O(d^n)$. A polynomial most of whose coefficients are nonzero is called dense.

To compute the product of two such polynomials, we may reduce the problem to the univariate case by means of Kronecker's substitution:

$$x_1 = y \ , \quad x_{k+1} = y^{D_1 \cdots D_k} \ , \quad k = 1, \ldots, n-1 \ .$$

Here, $D_j = 2d_j + 1$ exceeds the degree in $x_j$ of the product polynomial provided that both input polynomials have degrees at most $d_j$ in $x_j$, $j = 1, 2, \ldots, n$. Once the univariate product is computed, we may recover the product polynomial in the $n$ variables $x_i$ by inverting the Kronecker map. This method yields the complexity bound $O(N \log N \log\log N)$, where $N = \prod_{i=1}^{n} D_i$.

A certain improvement for polynomials with a large number of variables in [Pan, 1994] relies on the evaluation-interpolation method and (over any field of constants) yields complexity bound

$$O(N \log N \log\log D) = O(nD^n \log D \log\log D), \tag{5}$$

where $D = \max\{D_1, \ldots, D_n\}$, $N = D^n$, and the number of terms is of order $D^n$.

Another algorithm for dense polynomials is due to [Canny et al., 1989], also relies on the evaluation-interpolation scheme, but only applies over fields of characteristic 0. The algorithm supports, for a product polynomial with at most $T$ terms, each of a total degree at most $T$, the complexity bound $O(M(T) \log T) = O(T \log^2 T \log\log T)$, which is inferior to the estimate (5) under the bound $D$ on the degree of each variable but superior under the bound $T$ on the total number of terms.

Alternatively, we may extend the algorithm of [Karatsuba and Ofman, 1963], having complexity $O(D^{n \log 3})$.

Evaluation and interpolation of dense multivariate polynomials may use points on a grid (or lattice) in which each variable is assigned the values in a fixed set. Let $E(d)$ and $I(d)$ denote, respectively, the complexity of evaluating and interpolating a univariate polynomial of degree bounded by $(d-1)/2$ on $d$ points. Then, for grids of $d_j$ values for each variable $x_j$, the complexity of evaluation and interpolation of a multivariate polynomial is $d^{n-1}E(d)$ and $nd^{n-1}I(d)$, respectively, where $d = \max\{d_1, d_2, \ldots, d_n\}$. By applying FFT, we yield the bounds $O(d^n \log^2 d)$ and $O(nd^n \log^2 d)$, for evaluation and interpolation, respectively. For dense polynomials, these bounds are satisfactory. The approach of [Canny et al., 1989] adapts the algorithm of [Ben-Or and Tiwari, 1988] and also relies on solving a transposed Vandermonde system fast. It yields the bounds

$$O(T \log T \log\log T \log t) \qquad \text{and} \qquad O(T \log^2 T \log\log T)$$

for evaluation and interpolation, respectively, over fields of characteristic zero, where $t$ is the actual number of nonzero terms and $T$ is an upper bound on the number of input terms. Alternatively, we

can use the total degree of the product polynomial to estimate $T$, which is typically the case when this technique is applied to multiplication.

With multivariate polynomials, sparsity considerations become more important. Typically, the critical computation is interpolation. On the other hand, evaluation depends strongly on the form in which the polynomial is expressed and which is often a **determinant** formula [Manocha and Canny, 1993]. In addition to multiplication, computing the GCD can also be reduced to interpolation [Zippel, 1993]. So, in the rest of this section, we concentrate on sparse multivariate interpolation.

There are two main approaches with different advantages and drawbacks. One approach is Zippel's randomized algorithm. Its merit is that it does not require a bound on the number of nonzero terms as input but requires a bound $d$ on the degree in each variable. The computation involves Vandermonde matrices (cf. Section 4.1), and its complexity is $O^*(nd^2t)$, where $t$ denotes the number of nonzero terms of the input polynomial, and $O^*(\cdot)$ indicates that some polylogarithmic factors may have been omitted. There exists a deterministic version of this algorithm with higher, but still polynomial, complexity. For further information, see [Kaltofen and Lakshman, 1988, Grigoryev et al., 1990, Zippel, 1993].

Another, historically the first, approach is due to [Ben-Or and Tiwari, 1988]. The algorithm of [Ben-Or and Tiwari, 1988] does not need any degree bounds, but uses a bound on the actual number of terms $t$, on which both the algorithm and its estimates complexity, $O^*(ndt)$, depend. The algorithm applies to fields of characteristic equal to zero or to a very large positive integer. It proceeds by finding the exponents of the nonzero monomials which is reduced to the solution of the Berlekamp-Massey problem (cf. Section 3.3). Then, at the cost $O^*(ndt)$, the algorithm computes the corresponding coefficients, by exploiting the structure of Toeplitz and Vandermonde matrices [Ben-Or and Tiwari, 1988, Kaltofen and Lakshman, 1988].

# 4 Structured Matrices

Matrices with special structure are encountered in several applications to problems in sciences and engineering. These matrices have several repeated entries or entries that satisfy certain relations. More formally, an $n \times n$ **structured matrix** is typically defined by $O(n)$ entries, say by less than $2n$ entries, instead of the $n^2$ entries required in order to specify a general matrix. Moreover, structured matrices can typically be multiplied by a vector in $O(n \log n)$ or $O(n \log^2 n)$ ops, instead of $2n^2 - n$ ops, required for a general matrix. Structured matrices arise in numerous applications such as

control, signal and image processing, coding, a variety of algebraic computations, solution of PDEs, integral equations, singular integrals, conformal mappings, particle simulation, and Markov chains [Neuts, 1989, Chan, 1996]. In addition, several fundamental parallel computations with general matrices can be effectively reduced to computations with structured matrices [Bini and Pan, 1994].

This section focuses on computations with dense structured matrices, including Vandermonde, generalized Hilbert or Cauchy, circulant, Toeplitz, Hankel, and Bézout matrices. Computations with matrices of these classes are strongly related to computations with polynomials, thus enabling us to employ FFT in order to arrive at a dramatic acceleration of the algorithms versus the case of general matrices. For example, solving a nonsingular linear system of $n$ equations with a structured matrix of coefficients takes $O(n \log^2 n)$ or $O(n \log n)$ ops. Furthermore, a substantial improvement of parallel computations with general matrices can be obtained based on their reduction to computations with dense structured matrices and polynomials.

We refer the reader to [Bini and Pan, 1994] for information about other important classes of structured matrices, for instance, Frobenius (companion) matrices.

Hereafter, $(A)_{i,j}$ denotes the $(i,j)$-th entry of a matrix $A$.

## 4.1 Vandermonde and Cauchy (generalized Hilbert) Matrices

This section defines two important classes of structured matrices and demonstrates the improvement (versus general matrices) in the running time of basic matrix operations when this structure is exploited, based on correlation to computations with polynomials and FFT.

An $m \times n$ Vandermonde matrix $V$ has its entries $(V)_{i,j} = v_i^j$ for $i = 0, 1, \ldots, m$, $j = 0, 1, \ldots, n$. If $m = n$, then $V$ has the determinant

$$\det V = \prod_{0 \le i < k \le n} (v_k - v_i). \tag{6}$$

Clearly, a square Vandermonde matrix is nonsingular if and only if $v_i \neq v_k$ for $i \neq k$.

For example, for $m = 1$ and $n = 3$ we have

$$V = \begin{pmatrix} 1 & v_0 & v_0^2 & v_0^3 \\ 1 & v_1 & v_1^2 & v_1^3 \end{pmatrix}.$$

Some authors call $V^T$, rather than $V$, a Vandermonde matrix. A generalized Vandermonde matrix $G$ (cf. [Zippel, 1993]) is defined by $(G)_{i,j} = v_i^{e_j}$, $i = 0, \ldots, m, j = 0, \ldots, n$, for some sequence $0 \le e_0 < e_1 < \cdots < e_{n-1}$ of integers.

An important example of a Vandermonde matrix is given by the matrix $\sqrt{K}\Omega$, which is the scaled Fourier matrix $\Omega = (\omega^{ij}/\sqrt{K})$, $K = n+1$, associated to the DFT and the inverse DFT, where $\omega$ is a primitive $n$-th root of 1. Different FFT algorithms correspond to different factorizations of the matrix $\omega$; see [Van Loan, 1992] and [Bini and Pan, 1994, sect. 3.4].

Multiplication of a Vandermonde matrix $V = (v_i^j)$ by a vector $\mathbf{p}$ is equivalent to the evaluation at the points $v_0, \cdots, v_m$ of the polynomial with the coefficient vector $\mathbf{p}$. The solution of a linear system $V\mathbf{x} = \mathbf{v}$, where $\mathbf{x}$ and $\mathbf{v}$ are vectors and $V$ is a nonsingular square Vandermonde matrix, is equivalent to interpolating from the vector $\mathbf{v}$ of the polynomial values at the points $v_0, \cdots, v_m$ to the coefficient vector $\mathbf{x}$. Due to the algorithms of Section 3.2, both of these operations with Vandermonde matrices can be performed in $O((m+n)\log^2(m+n))$ ops, which is a dramatic decrease versus the case of a general $m \times n$ matrix. On the other hand, the estimated parallel complexity of these operations with general matrices can be decreased based on their reduction to operations with other structured matrices (see Section 4.4), and the Vandermonde-polynomial correlations have also been exploited in the reverse direction, in order to improve the known methods for polynomial evaluation and interpolation [Pan et al., 1993, Pan et al., 1997].

The same operations with $V^T$ can be performed in $O(n\log^2 n)$ ops, where $V$ is an $n \times n$ Vandermonde matrix (see Algorithm 4.1 in Section 4.2). The same bound also holds for computing the absolute value $|\det V|$. The speed of performing all these computations relies on their reduction to basic polynomial operations and, ultimately, to FFT. Straightforward computation of the determinant of an $n \times n$ Vandermonde matrix $V$ requires $O(n^2)$ ops, based on Equation (6), but for a real matrix $V$ [Gohberg and Olshevsky, 1994], also determines the sign of $\det V$, by means of ordering $v_0, \cdots, v_n$ and then using $O(n\log n)$ comparisons.

The next class of structured matrices is named after Cauchy. An $m \times n$ Cauchy matrix $C$ is defined by two vectors $\mathbf{s} = [s_i]$ and $\mathbf{t} = [t_j]$, such that $s_i \neq t_j$ for $i = 0, \ldots, m-1$, $j = 0, 1, \ldots, n-1$, and $(C)_{ij} = 1/(t_i - s_j)$. For instance, if $m = 2$, $n = 3$, then

$$C = \begin{pmatrix} (t_0 - s_0)^{-1} & (t_0 - s_1)^{-1} & (t_0 - s_2)^{-1} \\ (t_1 - s_0)^{-1} & (t_1 - s_1)^{-1} & (t_1 - s_2)^{-1} \end{pmatrix}.$$

Cauchy matrices generalize Hilbert matrices $(C)_{i,j} = 1/(i+j-1)$ and are sometimes called generalized Hilbert matrices. On the other hand, Cauchy matrices form a special subclass of Loewner matrices [Fiedler, 1984, Fiedler and Ptak, 1988], that is, matrices $B$ such that $(B)_{i,j} = (u_i - v_j)/(s_i - t_j)$.

Both (post)multiplication of a Cauchy matrix by a vector and solution (for vector $\mathbf{x}$) of the

equation $C\mathbf{x} = \mathbf{v}$, where $C$ is a square and nonsingular Cauchy matrix and $\mathbf{v}$ is a vector, reduce to polynomial evaluation and interpolation and take $O((m+n)\log^2(m+n))$ ops, due to application of FFT. The algorithm of [Rokhlin, 1985] approximates the product $Cv$ at the cost $c_\epsilon(m+n)\log(m+n)$ ops where $c_\epsilon$ depends on the approximation error $\epsilon$ and the minimum distance $\min_{i,j}|t_i - s_j|$. Some typical applications of Cauchy matrices include the study of integral equations, conformal mappings, and singular integrals [Rokhlin, 1985, O'Donnell and Rokhlin, 1989] (see also Section 4.4).

## 4.2 Circulant, Toeplitz, and Hankel Matrices

Toeplitz matrices and matrices closely related to them are among the most used structured matrices.

$T$ is a Toeplitz matrix if $(T)_{i,j} = (T)_{i+k,j+k}$, for all positive $k$, that is, if all the entries of $T$ are invariant in their shifts in the diagonal direction. $T$ is completely defined by its first row and its first column. For example, below we display a $4 \times 3$ Toeplitz matrix specified by the coefficients of polynomial $v(x) = v_2x^2 + v_1x + v_0$ in such a way as to express the multiplication of polynomials $u(x) = u_2x^2 + u_1x + u_0$ and $v(x)$ as (post)multiplication of the matrix by the column vector of the coefficients of $u(x)$.

$$\begin{pmatrix} v_2 & 0 & 0 \\ v_1 & v_2 & 0 \\ v_0 & v_1 & v_2 \\ 0 & v_0 & v_1 \\ 0 & 0 & v_0 \end{pmatrix} \begin{pmatrix} u_2 \\ u_1 \\ u_0 \end{pmatrix} = \begin{pmatrix} v_2u_2 \\ v_1u_2 + v_2u_1 \\ v_0u_2 + v_1u_1 + v_2u_0 \\ v_0u_1 + v_1u_0 \\ v_0u_0 \end{pmatrix}. \tag{7}$$

In general, in this way, Toeplitz matrices of such a form express polynomial multiplication and, hence, vector convolution. Furthermore, a general Toeplitz matrix $T$ can be embedded into a matrix of a such form as its middle block of rows. Then the product of $T$ by a vector can be immediately extracted from the associated polynomial product (convolution).

$H$ is a Hankel matrix if $(H)_{i,j} = (H)_{i-k,j+k}$, that is, if all the entries of $H$ are invariant in their shifts in the antidiagonal direction. $H$ is completely defined by its first row and its last column. For example, a $4 \times 3$ Hankel matrix is

$$H = \begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 & v_4 \\ v_2 & v_3 & v_4 & v_5 \end{pmatrix}.$$

The correlation of Hankel and Toeplitz matrices is formalized by introducing the reversion matrix.

$$J = J^{-1} = \begin{pmatrix} 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \end{pmatrix}. \tag{8}$$

Then $TJ$ and $JT$ are Hankel matrices for any Toeplitz matrix $T$, whereas $HJ$ and $JH$ are Toeplitz matrices for any Hankel matrix $H$. Consequently, computations with Hankel matrices can be reduced to computations with Toeplitz matrices, and vice versa. Hereafter, we will focus on the Toeplitz class.

$C_f = C_f(v)$, for a vector $v = [v_0, \ldots, v_{m-1}]^T$ and for a scalar $f$, is an $f$-circulant $m \times n$ matrix if $(C_f)_{i,j} = v_{i-j \bmod m}$ for $i \geq j$; $(C_f)_{i,j} = f v_{i-j \bmod m}$ for $i < j$. For example, for $m = n = 4$ we have

$$C_f(v) = \begin{pmatrix} v_0 & f v_3 & f v_2 & f v_1 \\ v_1 & v_0 & f v_3 & f v_2 \\ v_2 & v_1 & v_0 & f v_3 \\ v_3 & v_2 & v_1 & v_0 \end{pmatrix}.$$

$1-$circulant and $(-1)-$circulant matrices are called circulant and anticirculant, respectively. For any $f$, an $f$-circulant matrix is a special Toeplitz matrix, completely defined by its first column $v$ and by the scalar $f$. It is possible to embed an $m \times n$ Toeplitz matrix into an $m \times (m+n-1)$ circulant matrix. Therefore, certain operations on the former matrix, such as multiplication by a vector, can be reduced to multiplication of a circulant matrix by a vector.

Circulant matrix manipulation is fast due to FFT and the following well-known result:

**Theorem 4.1** [Cline et al., 1974, Davis, 1974] *Let $C_f$ be an $n \times n$ $f$-circulant matrix, with complex $f \neq 0$, and let $c_f^T$ denote its first row. Let $\Omega$ be the $n \times n$ Fourier matrix, $(\Omega)_{i,j} = \omega^{ij}/\sqrt{n}$, $i, j = 0, 1, \ldots, n-1$, where $\omega$ is a primitive $n$-th root of unity, $(\Omega^H)_{i,j} = \omega^{-ij}/\sqrt{n}$, and $\Omega^H \Omega = I$. Let $D_f = diag(1, g, g^2, \ldots, g^{n-1})$, $g^n = f$, and let $D$ be another diagonal matrix with entries given by the vector $\sqrt{n}\ \Omega D_f c_f$. Then*

$$\Omega D_f C_f D_f^{-1} \Omega^H = D, \quad \textit{or, equivalently,} \quad C_f = D_f^{-1} \Omega^H D \Omega D_f.$$

This immediately implies that multiplication of an $n \times n$ $f$-circulant matrix by a vector reduces to vector convolution, which can be implemented by three DFTs on $n$ elements and thus has complexity $O(n \log n)$. Consequently, multiplication of an $m \times n$ Toeplitz matrix by a vector takes $O((m+$

$n)\log(m+n))$ ops, and similarly for a Hankel matrix. Such an alternative reduction of $Tv$ to convolution is slightly more effective because the total size of FFTs involved is smaller. Conversely, computing the DFT of an $n$-vector, for a prime $n$, reduces to multiplying a vector by an $(n-1)\times(n-1)$ circulant matrix [Winograd, 1980].

For a nonsingular $n \times n$ Toeplitz matrix, the solution of a linear system $Tx = b$ has been extensively studied (see [Bini and Pan, 1994] and the references thereof). Different algorithms have different advantages with respect to running time and numerical stability, the record time-complexity bound being $O(n\log^2 n)$ and based on reductions to FFT. If the field of constants does not support FFT, then the alternative construction of [Cantor and Kaltofen, 1991] gives us all stated time bounds multiplied by $O(\log\log(m+n))$.

Given a square Toeplitz matrix $T$, the problem of computing its determinant and the coefficients of its characteristic polynomial both reduce to a sequence of multiplications of this matrix $T$ by vectors. This technique is effective for any matrix that can be multiplied by a vector at a low computational cost; it has been introduced in [Wiedemann, 1986] and is based on the computation of the associated Krylov sequence (see Chapter 12 of this handbook). If the field of constants supports FFT, then the complexity of both operations is $O(n^2\log n)$. To compute only the absolute value of the determinant, $O(n\log^2 n)$ ops suffice; moreover, if an $n\times n$ Toeplitz matrix is strongly nonsingular, its determinant is computed in $O(n\log^2 n)$ ops [Bini and Pan, 1994].

The inverse of a Toeplitz or Hankel matrix generally contains roughly $n^2/2$ distinct entries. There exist asymptotically optimal algorithms that compute the inverse in $O(n^2)$ ops, as well as slightly slower algorithms having better numerical stability (see [Bini and Pan, 1994] and the references thereof). A major result in this area is the following theorem of Gohberg and Semencul (see [Brent et al., 1980, Heinig and Rost, 1984, Bini and Pan, 1994] for this fundamental result and its variants).

**Theorem 4.2** *Let $T$ be a nonsingular $n\times n$ Toeplitz matrix, $(T)_{i,j} = t_{i-j}$, $i,j = 0,1,\dots,n-1$; $t = [s, t_{1-n}, t_{2-n}, \dots, t_{-1}]^T$, for any fixed scalar $s$; $x = [x_0,\dots,x_{n-1}]^T = T^{-1}t$; $v = [-1, x_{n-1},\dots,x_1]^T$; $y = [y_0,\dots,y_{n-1}]^T = T^{-1}[1,0,\dots,0]^T$; $u = [0, y_{n-1},\dots,y_1]^T$. For a given $n\times 1$ vector $a$, matrix $L(a)$ is an $n\times n$ lower triangular Toeplitz matrix whose first column is $a$. Then $T^{-1} = L(x)L^T(u) - L(y)L^T(v)$.*

In the special cases of a lower (upper) triangular Toeplitz or $f$-circulant matrix, the inverse is again a lower (upper) triangular Toeplitz or $f$-circulant matrix, respectively, and can be computed in

$O(n \log n)$ ops.

Generally, the product of two Toeplitz matrices is not a Toeplitz matrix but is in the generalized class of Toeplitz-like matrices, to be introduced in Section 4.4.

The classes of block Toeplitz and block Hankel matrices generalize Toeplitz and Hankel matrices, respectively. These are Toeplitz or Hankel matrices whose entries are matrices themselves. A $p \times q$ block Toeplitz (Hankel) matrix with blocks of size $r \times s$ can be turned into an $r \times s$ block matrix with $p \times q$ Toeplitz (Hankel) blocks and vice versa, by a sequence of row and column permutations.

Another class of structured matrices are banded matrices, where all nonzero entries are concentrated on a relatively small number of diagonals. Banded Toeplitz matrices are important in several applications [Bini and Pan, 1994, sect. 2.11]. The sine transform enables us to simplify computations with such matrices. For instance, it is possible to embed a symmetric $(2k+1)$-diagonal $n \times n$ Toeplitz matrix $T$ into an $(n+2\lfloor k/2 \rfloor) \times (n+2\lfloor k/2 \rfloor)$ matrix of the $\tau$ algebra, defined in Section 2.3. Here is an example with $k = 2$ and $n = 4$:

Embedding the matrix

$$T = \begin{pmatrix} a_0 & a_1 & a_2 & 0 \\ a_1 & a_0 & a_1 & a_2 \\ a_2 & a_1 & a_0 & a_1 \\ 0 & a_2 & a_1 & a_0 \end{pmatrix} \quad \text{yields} \quad \begin{pmatrix} a_0 - a_2 & a_1 & a_2 & 0 & 0 & 0 \\ a_1 & a_0 & a_1 & a_2 & 0 & 0 \\ a_2 & a_1 & a_0 & a_1 & a_2 & 0 \\ 0 & a_2 & a_1 & a_0 & a_1 & a_2 \\ 0 & 0 & a_2 & a_1 & a_0 & a_1 \\ 0 & 0 & 0 & a_2 & a_1 & a_0 - a_2 \end{pmatrix} \in \tau. \qquad (9)$$

This property allows us to reduce the solution of a band Toeplitz linear system to performing the sine transform and to solving a $k \times k$ linear system [Bini and Capovani, 1983].

Block Toeplitz and block Hankel matrices, block matrices with Toeplitz and Hankel blocks, banded Toeplitz matrices, and several other classes of Toeplitz-like and Hankel-like matrices, including proper Toeplitz and $f$-circulant matrices, naturally arise in numerous applications, in particular, to control, signal processing, systems theory, solution of PDEs, and algebraic computations. A simple example is a Sylvester matrix $S$, which is a $2 \times 1$ block matrix with Toeplitz blocks. It can be multiplied with a vector in quasi-linear time. Hence the Krylov sequence associated with $S$ can be computed in quasi-quadratic time, which yields fast algorithms for determinant computation, inversion, etc. Similarly, the structure of various multivariate resultant matrices leads to an acceleration of their construction, of computing the resultant itself, and of approximating the solutions of a polynomial system of equations (cf. Chapter 12 of this handbook). On these and other

computations with structured matrices, see [Heinig and Rost, 1984, Rokhlin, 1985, Chan, 1988, Chan and Strang, 1989, Bini and Pan, 1994, Mourrain and Pan, 1997, Emiris and Pan, 1997] and references therein.

To conclude this section, we recall an algorithm of [Canny et al., 1989] for a linear system $V^T\mathbf{x} = \mathbf{w}$. An alternative and simpler approach would be to reduce this question to one on $V$ by applying Tellegen's theorem; see [Bürgisser et al., 1997, thm. 13.20]. Still, we recall below some interesting techniques from [Canny et al., 1989]. Matrix $V^T$ is an $n \times n$ transposed Vandermonde matrix, such that $(V)_{i,j} = v_i^j$, for $i, j = 0, \ldots, n-1$, and $\mathbf{w}$ is a given column vector. The problem is reduced to computing the vector $\mathbf{w}^T V^{-1}$, $\mathbf{w}^T \left(V^T V\right)^{-1} V^T$. Observe that

$$(V^T V)_{i,j} = \sum_{k=0}^{n-1} v_k^{i+j}, \qquad i, j = 0, \ldots, n-1,$$

so that $V^T V$ is a Hankel matrix. Consider the values $v_i$ as the roots of the (monic) polynomial $p(x) = \sum_{i=0}^{n-1} p_i x^i = \prod_{i=0}^{n-1}(x - v_k)$, $p_{n-1} = 1$, and compute all $2n-1$ distinct entries of $V^T V$, that is, $s_j = \sum_{i=0}^{n-1} v_i^j$, $j = 0, \ldots, 2n-2$, as the first $2n-1$ power sums of the roots $v_0, \ldots, v_{n-1}$.

Formally, we solve a linear system $V^T\mathbf{x} = \mathbf{w}$ as follows:

**Algorithm 4.1**

*1. Compute the coefficients $p_0, \ldots, p_{n-2}$ of the above monic polynomial $p(x)$ by the fan-in method used in polynomial evaluation (cf. Section 3.2). This requires $O(n \log^2 n)$ ops. Write $p_{n-1} = 1$.*

*2. Compute the $s_0, \ldots, s_{2n-2}$ by solving in $O(n \log n)$ ops the lower triangular Toeplitz system of Newton's identities,*

$$\begin{pmatrix} 1 & & & & \\ p_{n-1} & \ddots & & & O \\ \vdots & \ddots & \ddots & & \\ p_0 & & \ddots & \ddots & \\ & \ddots & & \ddots & \ddots \\ O & & p_0 & \ldots & p_{n-1} \quad 1 \end{pmatrix} \begin{pmatrix} s_0 \\ \\ \vdots \\ \\ s_{2n-2} \end{pmatrix} = \begin{pmatrix} -p_{n-1} \\ -2p_{n-2} \\ \vdots \\ -np_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

*3. Compute the vector $\mathbf{y} = (V^T V)^{-1}\mathbf{w}$ by solving the Hankel linear system $(V^T V)\mathbf{y} = \mathbf{w}$. This requires $O(n \log^2 n)$ ops.*

*4. Compute and output the desired vector $\mathbf{x} = V\mathbf{y} = (V^T)^{-1}\mathbf{w}$. Due to the results of Section 4.1, this step is equivalent to polynomial evaluation and uses $O(n \log^2 n)$ ops.*

In summary, the multiplication of $V^T$ by a vector takes $O(n \log^2 n)$ ops, and the same complexity bound holds for solving a linear system defined by the transpose of a Vandermonde matrix.

Algorithm 4.1 has been applied in [Canny et al., 1989] to multivariate polynomial multiplication (cf. Section 3.5).

## 4.3 Bézout Matrices

Bézout matrices are structured matrices arising in several fundamental algebraic operations. In particular, they play a central role in studying the solutions of a system of two univariate polynomials and the GCD of two univariate polynomials, especially, when Boolean complexity and numerical stability issues become critical. Moreover, this study can be generalized to the solution of a system of several multivariate polynomial equations. This section introduces Bézout's matrices, demonstrates their structure and correlations to structured matrices studied above, and discusses some applications and computational complexity issues.

Consider two polynomials, $u(x) = \sum_{i=0}^{n} u_i x^i$ and $v(x) = \sum_{i=0}^{m} v_i x^i$, of degrees $n$ and $m$, respectively, where $m \leq n$. The expression

$$\frac{u(z)v(w) - v(z)u(w)}{z - w} = \sum_{i,j=0}^{n-1} b_{i,j} z^i w^j \tag{10}$$

is easily verified to be a polynomial in $w$ and $z$. This polynomial is called the generating function of the $n \times n$ Bézout matrix $B$ or, simply, Bezoutian of $u(x)$ and $v(x)$, with $(B)_{i,j} = b_{i,j}$. The polynomial of (10) is sometimes also called the Bezoutian of $u(x)$ and $v(x)$.

By definition, $B(u, v) = -B(v, u)$. Observe, as particular cases, that

$$B(u,1) = \begin{pmatrix} u_1 & \dots & u_n \\ \vdots & \iddots & \\ u_n & & O \end{pmatrix}, \qquad B(u, x^n) = -\begin{pmatrix} O & & u_0 \\ & \iddots & \vdots \\ u_0 & \dots & u_{n-1} \end{pmatrix}$$

are triangular Hankel matrices. It has been proven that $B(u, v)$, for general $u(x)$ and $v(x)$, can be written in terms of matrices of this form (see, for instance, [Lancaster and Tismenetsky, 1985]):

$$B(u, v) = B(v, 1)JB(u, x^n) - B(u, 1)JB(v, x^n), \tag{11}$$

where $J$ is the reversion matrix of (8).

There is a deeper connection of Bezoutians to Hankel matrices [Bini and Pan, 1994, sect. 2.9]. Given univariate polynomials $u(x), v(x)$, of respective degrees $n, m$, where $n > m$, consider the infinite sequence of values $h_0, h_1, \ldots$, in the coefficient field of $u(x), v(x)$, known as the Markov parameters and defined as follows:

$$\frac{v(x^{-1})}{xu(x^{-1})} = \sum_{i=0}^{\infty} h_i x^i \quad \text{or, equivalently,} \quad \frac{v(x)}{u(x)} = \sum_{i=0}^{\infty} h_i x^{-i-1}.$$

The Markov parameters generate an $n \times n$ Hankel matrix $H(u, v)$ such that $(H)_{i,j} = h_{i+j}$ for $i, j = 0, 1, \ldots, n-1$. Given $u(x), v(x)$, the entries of $H(u, v)$ are computed by solving a lower triangular Toeplitz system of $2n-1$ equations in $O(n \log n)$ ops. These equations are obtained by truncating the first formal power series above and expressing polynomial multiplication by a Toeplitz matrix as in Equation (7).

If $u(x)$ and $v(x)$ are relatively prime, then $H(u, v)$ is nonsingular. Moreover, for any nonsingular $n \times n$ Hankel matrix $H$, there exists a pair of relatively prime polynomials $u(x)$, $v(x)$, of degrees $n$ and $m < n$, respectively, such that $u(x)$ is monic and $H = H(u, v)$. Matrix $H(u, v)$, for polynomials $u(x)$ and $v(x)$ whose degrees $m$ and $n$ satisfy $n > m$, is related to the Bezoutian $B(u, v)$ of the same polynomials by

$$B(u, v) = B(u, 1)H(u, v)B(u, 1) \quad \text{and} \quad B(u, w)H(u, v) = I_n,$$

where $w(x)$ is a univariate polynomial of degree less than $n$ and such that $w(x)v(x) = 1 \bmod u(x)$ and where $I_n$ is the $n \times n$ identity matrix. The second expression states, in words, that the inverse of any nonsingular Hankel matrix is a Bezoutian. This expression, together with Equation (11), extends the Gohberg-Semencul formula and theorem 4.2 to Hankel matrices.

An important application of these results is to compute the polynomial GCD [Bini and Pan, 1994]. Assume that $u(x)$, $v(x)$ are monic and their degrees satisfy $m < n$. Let $k \geq 1$ be such that the degree of $\gcd(u(x), v(x))$ in $x$ is $n - k$. Then $\operatorname{rank}(H(u, v)) = k$, $\det H_k \neq 0$, where $H_k$ is the $k \times k$ leading principal submatrix of $H(u, v)$. Moreover, if $H_{k+1}\mathbf{w} = 0$, $\mathbf{w} = [w_0, \ldots, w_k]$, $w_k = 1$, then

$$u(x) = \gcd(u(x), v(x)) \sum_{i=0}^{k} w_i x^i.$$

This result yields an algorithm for the GCD using $O(n^2 \log n)$ ops and supporting the record parallel complexity. In addition, this algorithm is quite stable numerically (since the Bezoutian entries have bounded moduli) and supports the record Boolean complexity estimates for computing the GCD of a pair of polynomials defined over the integers or rationals.

Bezoutians have been studied in classical elimination theory (starting with univariate polynomials and then extended to the multivariate case), in stability theory, as well as in computing the reduction of a general matrix to the tridiagonal form. The reader may consult [Bini and Pan, 1994, ch. 2], for a comprehensive introduction, and [Householder, 1970, Fiedler, 1984, Lancaster and Tismenetsky, 1985, Cardinal and Mourrain, 1996] for further information.

## 4.4 Correlations Among Structured Matrices

In this section we extend effective algorithms for Toeplitz matrix computations to computations with more general classes of dense structured matrices. In this way, we will unify efficient computations with various dense structured matrices. This approach has further extensions to computations with polynomials, rational functions, and general matrices. In particular, we apply certain shifts and scaling operators to the matrices in order to represent in a simpler and more convenient form.

The main tool are linear operators $F$ that map an $m \times n$ structured matrix $A$ into the product of two matrices $F(A) = GH^T$, where $G$ is $m \times d$ and $H$ is $n \times d$ matrices for a smaller $d$. Then $r = rank(F(A))$, the rank of the matrix $F(A)$, is called the $\underline{F\text{-rank}}$ of $A$, and the pair of the matrices $G$ and $H$ is called an $\underline{F\text{-generator}}$ of $A$ of $\underline{\text{length}}$ $d$.

For the structured matrices seen so far, there exist linear operators $F$ such that the $F$-generators of $A$ have constant length. Moreover, $A$ can be efficiently expressed via the generators. Thus, an $m \times n$ matrix $F(A)$ with its generator $(G, H)$ of length $r$ can be represented by using only $(m + n)r$, rather than $mn$, words of storage space. Furthermore, we shall see that to compute the vector $A\mathbf{v}$, for a given pair of matrices $G$, $H$, satisfying $F(A) = GH^T$, and for a vector $\mathbf{v}$, we only need $O((m + n)r \log^h(m + n))$ ops, for $h = 1$ or $h = 2$, rather than $(2n - 1)m$ ops. Now, for a given dense structured matrix $A$, we look for an operator $F$ that transforms $A$ into a low rank matrix, from which we could easily recover the original matrix. Then we may take all the advantages of operating with low rank matrices, even though the structured input matrix may have the full rank. Below we provide some further specifications; more details are found in [Kailath et al., 1979, Bini and Pan, 1994, Gohberg and Olshevsky, 1994], including a discussion on the powerful theorems that relate the $F$-rank of a nonsingular matrix and that of its inverse.

Let $Z = Z_k$ be the following $k \times k$ square down-shift matrix,

$$Z = \begin{pmatrix} 0 & \dots & & \dots & 0 \\ 1 & 0 & & & \vdots \\ 0 & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 1 & 0 \end{pmatrix}.$$

We generalize the class of Toeplitz matrices to the class of Toeplitz-like $m \times n$ matrices $A$, whose $F$-rank is bounded from above by a fixed constant independent of $m$ and $n$, where $F$ is one of the two following operators $F_+$ or $F_-$:

$$F_+(A) = F_{Z,Z^T}(A) = A - Z_m A Z_n^T, \quad F_-(A) = F_{Z^T,Z}(A) = A - Z_m^T A Z_n.$$

$F(A) = A - PAQ$ and $F(A) = PA - QA$, for a pair of fixed matrices $P, Q$, are typical forms of linear operators defining various classes of structured matrices.

Due to the shifting properties of multiplications by $Z$ and $Z^T$, the operators $F_+, F_-$ turn into zero all the entries of a Toeplitz matrix $A$, except for the first row and column under $F_+$ and for the last row and column under $F_-$, which are invariant in the transition from $A$ to $F(A)$. The above $F$-generators, for $F = F_+$ and $F = F_-$, have lengths at most 2 for Toeplitz matrices. The length only increases to $p + q$ for $p \times q$ block matrices with Toeplitz blocks. For these operators, it is possible to recover a matrix $A$ from $F(A)$. Therefore, in the Toeplitz case, we may shift to operating with $F(A)$, and thus save computational resources of time and space. For instance, we may compute the product $Av$ for arbitrary vector $v$ in time $O(nr \log n)$, where $r$ is the rank of $F(A)$, which is typically much smaller than $n$.

For a concrete illustration, take matrix $A = T$ of Example (9). Then we have

$$F_+(A) = \begin{pmatrix} a_0 & a_1 & a_2 & 0 \\ a_1 & 0 & 0 & 0 \\ a_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} a_0 & 1 \\ a_1 & 0 \\ a_2 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & a_1 & a_2 & 0 \end{pmatrix},$$

where the rank and length of $F(A)$ is 2. In the above notation $F_+(A) = GH$, and let $g_i, h_i$ be the respective column vectors of $G, H$, for $i = 1, 2$. Then, $F_+(A) = \sum_{i=1}^{2} g_i h_i^T$. The important property is that $A$ can be expressed as a sum of $d$ products of triangular Toeplitz matrices. Here, $A = \sum_{i=1}^{2} L(g_i) L^T(h_i)$, where for any $1 \times n$ vector $v$, $L(v)$ is a lower triangular $n \times n$ Toeplitz matrix, with $v$ being its first column. This kind of representation using $F_+(A), F_-(A)$ is possible

for any matrix $A$. A small number of products $L(g_i)L^T(h_i)$ in the sum characterizes matrices with Toeplitz-like structure, and then we yield efficient ways for several basic operations, including vector multiplication and the recovery of $A$ from its $F$-image, for the various $F$ such as $F = F_+$ and $F = F_-$.

Analogously, $A$ is said to be Hankel-like if the $F$-rank does not exceed a fixed constant under one of the following two operators:

$$F_{Z,Z}(A) = A - Z_m A Z_n, \quad F_{Z^T,Z^T}(A) = A - Z_m^T A Z_n^T,$$

and the stated properties of Toeplitz-like matrices can be extended.

A further generalization is to Toeplitz-like + Hankel-like matrices, expressed as the sum of a Toeplitz-like matrix and a Hankel-like matrix. This class includes both Toeplitz-like and Hankel-like matrices. Suppose that such a pair of $n \times n$ matrices is given by their $F$-generators with a constant $F$-length. The $F$-generator of their product can be computed in $O(n \log n)$ ops by using FFT. At the computational cost $O(n^2 \log n)$, we can compute the inverse and the characteristic polynomial and solve a linear system $Tx = b$ of an $n \times n$ Toeplitz-like+Hankel-like matrix $T$. This complexity bound is record and nearly optimal (up to a logarithmic factor) for the inverse and characteristic polynomial, even in the class of Toeplitz matrices, and the approach yields the record parallel complexity bounds also for solving a Toeplitz-like + Hankel-like linear system of equations [Pan, 1992, Bini and Pan, 1993, Bini and Pan, 1994]. Sequential algorithms for Toeplitz-like + Hankel-like linear systems cost $O(n \log^2 n)$ ops (see [Bini and Pan, 1994]). There are also several effective iterative algorithms for Toeplitz and other structured linear systems (see [Pan, 1992a, Pan, 1992b, Pan, 1993, Pan et al., 1995, Pan et al., 1997a]), and the reader is also referred to [Pan, 1992, Bini and Pan, 1993, Pan, 1993, Bini and Pan, 1994] on some other basic properties of Toeplitz-like and Toeplitz-like + Hankel-like matrices.

Let $D(\mathbf{v}) = D_n(\mathbf{v}) = diag(v_0, v_1, \ldots, v_{n-1})$ where $\mathbf{v}$ is the vector $[v_0, v_1, \ldots, v_{n-1}]^T$ and let us extend the class of Cauchy (or generalized Hilbert) matrices as follows: an $m \times n$ matrix $C$ is said to be Cauchy (Hilbert)-like if the $F$-rank of $C$ is bounded by a constant independent of $m$ and $n$, $F$ is the operator

$$F_{\mathbf{s},\mathbf{t}}(C) = D_m(\mathbf{s})C - CD_n(\mathbf{t}),$$

and $\mathbf{s}$ and $\mathbf{t}$ denote a pair of $m$- and $n$-dimensional vectors, respectively.

By definition, a Loewner matrix is Cauchy (Hilbert)-like.

An operator of the Cauchy type is represented by a nonsingular matrix if and only if every entry of $\mathbf{s}$ is distinct from all entries of $\mathbf{t}$. For the Cauchy matrix $C$ with $(C)_{i,j} = \frac{1}{s_i - t_j}$, this operator gives us $F_{\mathbf{s},\mathbf{t}}(C) = \mathbf{e}\mathbf{e}^T$, where $\mathbf{e} = [1, \ldots, 1]^T$. We refer the reader to [Bini and Pan, 1994] on further study of the matrices of this class.

The class of $m \times n$ Vandermonde-like matrices is formed by the matrices having a constant $F$-rank, where $F$ is defined by any of the following matrix equations:

$$F_{\mathbf{v},Z}(V) = D_m(\mathbf{v})V - VZ_n, \qquad F_{\mathbf{v},Z^T}(V) = D_m(\mathbf{v})V - VZ_n^T,$$
$$F_{Z,\mathbf{v}}(V) = VD_n(\mathbf{v}) - Z_mV, \qquad F_{Z^T,\mathbf{v}}(V) = VD_n(\mathbf{v}) - Z_m^TV.$$

If $m = n$ and $V$ is defined by an $n$-dimensional vector $\mathbf{v}$, then

$$F_{\mathbf{v},Z}(V) = \begin{pmatrix} 0 & \cdots & 0 & v_0^n \\ \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & v_{n-1}^n \end{pmatrix} = D^n(\mathbf{v})\,\mathbf{e}\,[0, \ldots, 0, 1].$$

Both Cauchy-like and Vandermonde-like matrices can be recovered from their respective images under the operators specified here.

We conclude this section by following [Pan, 1990] in order to demonstrate how associating the operators unifies the treatment of dense structured matrices, revealing some important but hidden correlations among various classes of such matrices (such correlations have been already demonstrated when we solved a transposed Vandermonde linear system in Section 4.2). The first basic idea is that several operations can be applied to the generators without explicitly constructing the matrices. Addition and subtraction, for the operators examined, satisfy

$$F(A \pm B) = F(A) \pm F(B),$$

and the $F$-generator of the sum or difference is the union of the other two generators.

For multiplication, we may rely on a result of [Pan, 1990], generalizing a result of [Chun et al., 1987], which allows us to compute a generator of the product in terms of the generators of the two given matrices. Let $K$, $L$, $M$ and $N$ be four fixed matrices, $\Delta = LM - I$, $I$ be the identity matrix, and let $F_{(P,Q)}$ be an operator defined by the equation $F_{(P,Q)}(A) = A - PAQ$, for any matrices $P, Q$ and $A$. Then

$$F_{(K,N)}(AB) = F_{(K,L)}(A)B + KALF_{(M,N)}(B) + KA(LM - I)BN.$$

Let $r$=rank $F_{(K,N)}(AB)$, $r_1$ = rank $F_{(K,L)}(A)$, and $r_2$ = rank $F_{(M,N)}(B)$. It follows that $r \leq$ rank $\{(LM - I) + r_1 + r_2\}$.

Based on this result, [Pan, 1990] extends the algorithms for structured matrices seen so far to matrices with similar structures. For instance, specialize $L$ and $M$ as follows:

$$L = Z,\ M = Z^T, \qquad \text{or} \qquad L = Z^T,\ M = Z,$$

where $Z$ is the down-shift matrix defined above. If we require all three operators be of Hankel and/or Toeplitz type, then $K$ and $N$ are implicitly defined. If all matrices are of size $n \times n$ and the length of $F$-generator of $A, B$ is $d_1, d_2$, respectively, then an $F$-generator for $AB$ of a length at most $d_1 + d_2 + 1$ can be computed in $O(n(d_1 + d_2)^2 \log(n(d_1 + d_2)))$ ops. Moreover, the rank of $LM - I$ for both specializations is 1, so the $F$-rank of the product is bounded by $1 + r_1 + r_2$. This implies that the product of two Toeplitz- or Hankel-like matrices is again Toeplitz- or Hankel-like, with $F$-rank bounded as in the second to last line of the following table from [Pan, 1990] and [Bini and Pan, 1994, sect. 2.12].

Different specializations of matrices $K, L, M$, and $N$ imply the rest of the entries in the table. We write HT, C and V, for the classes of Hankel-like+Toeplitz-like, Cauchy-like and Vandermonde-like matrices, respectively.

| $A$ | $B$ | $AB$ | $F$-rank of $AB$ |
|---|---|---|---|
| HT | V | V | $r \le r_1 + r_2 + 1$ |
| V | V | C | $r \le r_1 + r_2 + 1$ |
| V | V | HT | $r \le r_1 + r_2$ |
| C | V | V | $r \le r_1 + r_2$ |
| HT | HT | HT | $r \le r_1 + r_2 + 1$ |
| C | C | C | $r \le r_1 + r_2 + 1$ |

By using these transitions, we may reduce the original computational problem to other problems for which we have effective algorithms. For instance, suppose that, given an $F_{\mathbf{s},\mathbf{t}}$-generator for a Cauchy-like matrix $A$, with $\mathbf{s}, \mathbf{t}$ vectors, we wish to compute its determinant and its inverse, when the latter exists. We may choose $B$ to be the Vandermonde matrix defined by the inverse entries of $\mathbf{t}$ and compute an $F$-generator for $AB$. Then the matrix $AB$ is Vandermonde-like, with rank bounded as in the fourth line of the table above. Now, $\det A = \det(AB)/\det B$ and $A^{-1} = B(AB)^{-1}$, so we have reduced the original problems to a sequence of operations on Vandermonde-like matrices.

After the transformations displayed in the above table have been established in [Pan, 1990], some of them have been further simplified, due to the following result of [Gohberg et al., 1995]:

**Theorem 4.3** *$C = \Omega T D_{2n} \Omega^{-1}$ is a Cauchy-like matrix, having a r-generator $G^* = \Omega G$, $H^* = \Omega D(w) H$, for $F = F_{D_n, D_{-n}} = F(D_n, D_{-n})$, that is,*

$$F_{D_n, D_{-n}} C = D_n C - C D_{-n} = G^* (H^*)^T,$$

*provided that $T$ is a Toeplitz-like matrix with an $F$-generator $G, H$ for $F = F_{Z_1, Z_{-1}} = F(Z_1, Z_{-1})$,*

$$F_{Z_1, Z_{-1}} T = Z_1 T - T Z_{-1} = G H^T,$$

*$D_k = diag(1, w_k, w_k^2, \ldots, w_k^{n-1})$, $w_k = exp(2\pi\sqrt{-1}/k)$, and $\Omega = 1/\sqrt{n}(w_n^{ij})$ is a Fourier matrix, $\Omega^{-1} = 1/\sqrt{n}(w_n^{-ij})$, $i, j = 0, 1, \ldots, n-1$.*

Due to this result, the transformation from any Toeplitz-like matrix into a Cauchy-like matrix is immediately reduced to FFT, which leads to effective practical algorithms for Toeplitz-like linear systems [Gohberg et al., 1995]. Another example of applications of this kind is the fast algorithms for polynomial evaluation and interpolation based on manipulation with structured matrices of the classes $HT$, $V$ and $C$ [Pan et al., 1993, Pan et al., 1997].

Additional bibliography on the material of this section can be found in [Bini and Pan, 1994].

# 5 Defining Terms

**Chinese Remainder algorithm:** the algorithm recovering a unique integer (or polynomial) $p$ modulo $M$ from the $h$ residues of $p$ modulo pairwise relatively prime integers (or, respectively, polynomials) $m_1, \cdots, m_h$, where $M$ is the product $m_1, \cdots, m_h$. (The **residue** of $p$ modulo $m$ is the remainder of the division of $p$ by $m$.)

**convolution of two vectors:** a vector that contains the coefficients of the product of two polynomials whose coefficients make up the given vectors. The positive and negative wrapped convolutions are the coefficient vectors of the 2 polynomials obtained via reduction of the polynomial product by $x^n + 1$ and $x^n - 1$, respectively.

**determinant (of an $n \times n$ matrix):** a polynomial of degree $n$ in the entries of the matrix with the property of being invariant in the elementary transformations of a matrix used in Gaussian

elimination; the determinant of the product of matrices is the product of their determinants; the determinant of a triangular matrix is the product of its diagonal entries; the determinant of any matrix is non-zero if and only if the matrix is invertible (nonsingular).

**discrete Fourier transform (DFT):** the vector of the values of a given polynomial at the set of all the $K$-th roots of unity. **The inverse discrete Fourier transform (IDFT)** of a vector $\mathbf{v}$: the vector of the coefficients of a polynomial whose values at the $K$-th roots of 1 form a given vector $\mathbf{v}$.

**divide-and-conquer:** a general algorithmic method of dividing a given problem into two (or more) subproblems of smaller sizes which are easier to solve, then synthesizing the overall solution from the solutions to the subproblems.

**fast Fourier transform (FFT):** an algorithm that uses $1.5K \log K$ ops in order to compute the DFT at the $K$-th roots of 1, where $K = 2^k$, for a natural $k$. It also computes the IDFT in $K + 1.5K \log K$ ops.

**greatest common divisor (GCD) of 2 or several integers (or polynomials):** the largest positive integer (or a polynomial of the largest degree ) that divides both or all of the input integers (or polynomials).

**least common multiple (LCM) of 2 or several integers (or polynomials):** the smallest positive integer (or a polynomial of the smallest degree) divisible by both or by all of the input integers (or polynomials).

**interpolation:** the computation of the coefficients of a polynomial in one or more variables, given its values at a set of points. The inverse problem is the **evaluation** of a given polynomial **on a set of points**.

**ops:** arithmetic operations, i.e., additions, subtractions, multiplications, or divisions.

**structured matrix:** a matrix whose each entry can be derived by a formula depending on a few parameters. For instance, the Hilbert matrix has $\frac{1}{i+j-1}$ as the entry in row $i$ and column $j$.

**Taylor shift of the variable:** recovery of the coefficient vector of a polynomial after a linear substitution of its variable, $y = x + \Delta$ for $x$, where $\Delta$ is a fixed shift value.

## 6 Research Issues and Summary

We have reviewed the known highly effective algorithms for the discrete Fourier transform (DFT) and its inverse, as well as for some related transforms, all of them based on the celebrated fast Fourier transform (FFT) algorithm. We have shown immediate application of FFT to computing convolution of vectors (polynomial products), which is a fundamental operation of computer algebra and signal processing. We have also demonstrated further applications to other basic operations with integers, univariate and multivariate polynomials and power series, as well as to the fundamental computations with circulant, Toeplitz, Hankel, Vandermonde, Cauchy (generalized Hilbert), and Bézout matrices, as well as to other structured matrices related to the above classes of matrices via the associated linear operators. We exemplified some major techniques establishing such relations and other major basic techniques for extending the power of FFT to numerous other computational problems, and we supplied pointers to further bibliography. Some of these techniques are quite recent, and further research is very promising, in particular, via the study of structured matrices and polynomial systems of equations [Canny et al., 1989, Mourrain and Pan, 1997, Emiris and Pan, 1997]. New practical and theoretical research directions have emerged recently related to the numerical implementation of FFT, leading to interesting algebraic techniques [Buhler et al., 1997] and certain finite group applications [Cole and Hariharan, 1996, Farach et al., 1995].

## References

[Aho et al., 1974] Aho, A.V., Hopcroft, J.E., and Ullman, J.D. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachussetts.

[Aho et al., 1975] Aho, A., Steiglitz, K., and Ullman, J. 1975. Evaluating polynomials at fixed set of points. *SIAM J. Comput.*, 4:533–539.

[Bailey, 1993] Bailey, D.H. 1993. A high-performance FFT algorithm for vector supercomputers. *ACM Trans. on Math. Soft.*, 19(3):288–319.

[Bailey, 1993b] Bailey, D.H. 1993b. Multiprecision translation and execution of FORTRAN programs. *ACM Trans. on Math. Soft.*, 19(3):288–319.

[Ben-Or and Tiwari, 1988] Ben-Or, M. and Tiwari, P. 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. ACM Symp. Theory of Computing*, pages 301–309, New York. ACM Press.

[Berlekamp, 1968] Berlekamp, E.R. 1968. *Algebraic Coding Theory.* McGraw-Hill, New York.

[Biehl et al., 1995] Biehl, I., Buchmann, J., and Papanikolaou, T. 1995. LiDIA: A library for computational number theory. Technical Report SFB 124–C1, Universität des Saarlandes, Saarbrücken 66041 Germany. `http://www-jb.cs.uni-sb.de/linktop/LiDIA`.

[Bini and Bozzo, 1993] Bini, D. and Bozzo, E. 1993. Fast Discrete Transform by means of eigenpolynomials, *Comp. & Math. (with Appl.)*, 26(9):35–52.

[Bini and Capovani, 1983] Bini, D. and Capovani, M. 1983. Spectral and computational properties of band symmetric Toeplitz matrices. *Linear Algebra Appl.*, 52/53:99–126.

[Bini and Favati, 1993] Bini, D. and Favati, P. 1993. On a matrix algebra related to the Hartley transform. *SIAM J. Matrix Anal. Appl.*, 14(2):500–507.

[Bini and Pan, 1993] Bini, D. and Pan, V.Y. 1993. Improved Parallel Computation with Toeplitz-like and Hankel-like Matrices. *Linear Algebra Appl.*, 188/189:3–29.

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

[Blahut, 1984] Blahut, R.E. 1984. *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, New York, 1984.

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

[Brent et al., 1980] Brent, R.P., Gustavson, F.G., and Yun, D.Y.Y. 1980. Fast solution of Toeplitz systems of equations and computation of Padé approximations. *J. Algorithms*, 1:259–295.

[Brigham, 1974] Brigham, E.O. 1974. *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[Buchberger et al., 1982] Buchberger, B., Collins, G.E., and Loos, R., editors. 1982. *Computer Algebra: Symbolic and Algebraic Computation*, volume 4 of *Computing Supplementum*. Springer, Wien, 2nd edition.

[Buhler et al., 1997] Buhler, J., Shokrollahi, M.A., and Stemann, V. 1997. Fast and precise computations of discrete Fourier transforms using cyclotomic integers. In *Proc. ACM Symp. Theory of Comp.*, pages 40–47. ACM Press.

[Bürgisser et al., 1997] Bürgisser, P., Clausen, M., and Shokrollahi, M.A. 1997. *Algebraic Complexity Theory*. Springer, Berlin.

[Canny et al., 1989] Canny, J., Kaltofen, E., and Lakshman, Y. 1989. Solving systems of non-linear polynomial equations faster. In *Proc. ACM Intern. Symp. Symbolic Algebraic Comput. (ISSAC '89)*, pages 121–128. ACM Press.

[Cantor and Kaltofen, 1991] Cantor, D.G. and Kaltofen, E. 1991. On Fast Multiplication of Polynomials over Arbitrary Rings, *Acta Informatica*, 28(7): 697–701.

[Cardinal and Mourrain, 1996] Cardinal, J.-P. and Mourrain, B. 1996. Algebraic approach of residues and applications. In Renegar, J., Shub, M., and Smale, S., editors, *The Mathematics of Numerical Analysis*, volume 32 of *Lectures in Applied Math.*, pages 189–210. AMS.

[Chan, 1988] Chan, T.F. 1988. An optimal circulant preconditioner for Toeplitz systems. *SIAM J. Sci. Stat. Comput.*, 9:766–771.

[Chan, 1996] Chan, R. 1996. Scientific applications of iterative Toeplitz solvers. *Calcolo*, 33:249–267.

[Chan and Strang, 1989] Chan, R.H. and Strang, G. 1989. Toeplitz equations by conjugate gradients with circulant preconditioner. *SIAM J. Sci. Stat. Comput.*, 10:104–119.

[Chun et al., 1987] Chun, J., Kailath, T., and Lev-Ari, H. 1987. Fast parallel algorithm for QR-factorization of structured matrices. *SIAM J. Sci. Stat. Comput.*, 8(6):899–913.

[Clausen, 1989] Clausen, M. 1989. Fast generalized FFT. *Theor. Computer Science*, 56:55–63.

[Cline et al., 1974] Cline, R.E., Plemmons, R.J., and Worm, G. 1974. Generalized inverses of certain Toeplitz matrices. *Linear Algebra Appl.*, 8:25–33.

[Cole and Hariharan, 1996] Cole, R. and Hariharan, R. 1996. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. In *Proc. 7th ACM-SIAM Symp. Discrete Algorithms*, pages 323–332.

[Conte and de Boor, 1980] Conte, C.D. and de Boor, C. 1980. *Elementary Numerical Analysis: an Algorithmic Approach.* McGraw-Hill, New York.

[Cooley and Tukey, 1965] Cooley, J.W. and Tukey, J.W. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. of Comp.*, 19(90):297–301.

[Davis, 1974] Davis, P. 1974. *Circulant Matrices.* Wiley, New York.

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

[Duhamel and Vetterli, 1990] Duhamel, P. and Vetterli, M. 1990. Fast Fourier transforms: a tutorial review. *Signal Processing*, 19:259-299.

[Elliott and Rao, 1982] Elliott, D.F. and Rao, K.R. 1982. *Fast Transform Algorithms, Analyses, and Applications.* Academic Press, New York.

[Emiris, 1997] Emiris, I.Z. 1997. A complete implementation for computing general dimensional convex hulls. *Intern. J. Computational Geometry & Applications, Special Issue on Geometric Software.* To appear. A preliminary version as Tech. Report 2551, INRIA Sophia-Antipolis, France, 1995.

[Emiris and Pan, 1997] Emiris, I.Z. and Pan, V.Y. 1997. The structure of sparse resultant matrices. In *Proc. ACM Int. Symp. on Symb. Alg. Comp. (ISSAC '97)*, pages 189–196, ACM Press.

[Farach et al., 1995] Farach, M., Przytycka, T.M., and Thorup, M. 1995. Computing the agreement of trees with bounded degrees. In Spirakis, P., editor, *Proc. 3rd Annual European Symp. on Algorithms*, volume 979 of *Lect. Notes in Comp. Science*, pages 381–393, New York. Springer.

[Fateman, 1974] Fateman, R.J. 1974. Polynomial multiplication, powers and asymptotic analysis: Some comments. *SIAM J. Comp.*, 3(3):196–213.

[Fiduccia, 1987] Fiduccia, C.M. 1987. A rational view of the fast Fourier transform. In *Proc. 25th Allerton Conf. Commun., Control and Computing.*

[Fiedler, 1984] Fiedler, M. 1984. Hankel and Loewner matrices. *Linear Algebra Appl.*, 58:75–95.

[Fiedler and Ptak, 1988] Fiedler, M. and Ptak, V. 1988. Loewner and Bezout matrices. *Linear Algebra Appl.*, 101:187–220.

[Frigo and Johnson] Frigo, M. and Johnson, S.J. The Fastest Fourier Transform in the west. Available at `http://theory.lcs.mit.edu/~ fftw`.

[Geddes et al., 1992] Geddes, K.O., Czapor, S.R., and Labahn, G. 1992. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Norwell, Massachusetts.

[GNU, 1996] Free Software Foundation. 1996. GNU multiple precision library. `ftp://prep.ai.mit.edu/pub/gnu/gmp-M.N.tar.gz`.

[Gohberg et al., 1995] Gohberg, I., Kailath, T., and Olshevsky, V. 1995. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. of Comp.,* 64(212):1557–1576.

[Gohberg and Olshevsky, 1994] Gohberg, I. and Olshevsky, V. 1994. Complexity of multiplication with vectors for structured matrices, *Linear Algebra Appl.,* 202:163–192.

[Golub and Van Loan, 1996] Golub, G.H. and Van Loan, C.F. 1996. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 3rd edition.

[Grigoryev et al., 1990] Grigoryev, D.Y., Karpinski, M., and Singer, M.F. 1990. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Computing*, 19(6):1059–1063.

[Hardy and Wright, 1979] Hardy, G.H. and Wright, E.M. 1979. *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, 5th edition.

[Heinig and Rost, 1984] Heinig, G. and Rost, K. 1984. *Algebraic Methods for Toeplitz-like Matrices and Operators*, volume 13 of *Operator Theory*. Birkhauser.

[Householder, 1970] Householder, A.S. 1970. *The Numerical Treatment of a Single Nonlinear Equation*. McGraw-Hill, Boston.

[Huckle, 1996] Huckle, T. 1996. Iterative methods for ill-conditioned Toeplitz matrices. *Calcolo*, 33.

[Ja Ja, 1992] Ja Ja, J. 1992. *An Introduction to Parallel Algorithms*, Addison-Wesley, Massachusetts.

[Kailath et al., 1979] Kailath, T., Kung, S.-Y., and Morf, M. 1979. Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, 68(2):395–407.

[Kailath and Olshevsky, 1996] Kailath, T. and Olshevsky, V. 1996. Displacement structure approach to discrete-trigonometric transform based preconditioners of G. Strang type and of T. Chan type. *Calcolo*, 33, 1996.

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

[Kaltofen and Lakshman, 1988] Kaltofen, E. and Lakshman, Y. 1988. Improved sparse multivariate polynomial interpolation algorithms. In *Proc. ACM Intern. Symp. Symbolic Algebraic Comput. (ISSAC '88)*, volume 358 of *Lect. Notes in Comp. Science*, pages 467–474. Springer.

[Kaltofen et al., 1990] Kaltofen, E., Lakshman, Y.N., and Wiley, J.M. 1990. Modular rational sparse multivariate polynomial interpolation. In *Proc. ACM Intern. Symp. Symb. Algebr. Comput. (ISSAC '90)*, pages 135–139, New York. ACM Press.

[Kaltofen and Pan, 1991] Kaltofen, E. and Pan, V.Y. 1991. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. 3rd Ann. ACM Symp. on Parallel Algorithms and Architectures*, pages 180–191, New York. ACM Press.

[Karatsuba and Ofman, 1963] Karatsuba, A. and Ofman, Y. 1963. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl.*, 7:595–596.

[Knuth, 1997] Knuth, D.E. 1997. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Massachusetts.

[Lancaster and Tismenetsky, 1985] Lancaster, P. and Tismenetsky, M. 1985. *The Theory of Matrices*. Academic Press.

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

[Mourrain and Pan, 1997] Mourrain, B. and Pan, V.Y. 1997. Solving special polynomial systems by using structured matrices and algebraic residues. In Cucker, F. and Shub, M., editors, *Proc. Workshop on Foundations of Computational Mathematics*, pages 287–304, Berlin. Springer.

[Neuts, 1989] Neuts, M.F. 1989. *Structured stochastic matrices of M/G/1 type and their applications.* Marcel Dekker, New York.

[O'Donnell and Rokhlin, 1989] O'Donnell, S.T. and Rokhlin, V. 1989. A fast algorithm for numerical evaluation of conformal mappings. *SIAM J. Sci. Stat. Comput.*, 10(3):475–487.

[Pan, 1990] Pan, V.Y. 1990. Computations with dense structured matrices. *Math. of Comp.*, 55(191):179–190.

[Pan, 1992] Pan, V.Y. 1992. Parametrization of Newton's iteration for computations with structured matrices and applications. *Comp. & Math. (with Appl.)*, 24(3):61–75.

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

[Pan, 1992b] Pan, V.Y. 1992. Parallel Solution of Toeplitz-like Linear Systems, *J. of Complexity,* 8:1–21.

[Pan, 1993] Pan, V.Y. 1993. Concurrent iterative algorithm for Toeplitz-like linear systems, *IEEE Trans. on Parallel and Distributed Systems*, 4(5):592–600.

[Pan, 1994] Pan, V.Y. 1994. Simple multivariate polynomial multiplication. *J. Symb. Comp.*, 18:183–186.

[Pan, 1996] Pan, V.Y. 1996. Numerical computation of a polynomial GCD and extensions. Technical Report 2969, INRIA, Sophia-Antipolis, France.

[Pan, 1997] Pan, V.Y. 1997. Faster solution of the key equation for decoding the BCH error-correcting codes. In *Proc. ACM Symp. Theory of Comp.*, pages 168–175. ACM Press.

[Pan et al., 1992] Pan, V.Y., Landowne, E., and Sadikou, A. 1992. Univariate polynomial division with a remainder by means of evaluation and interpolation. *Information Process. Letters*, 44:149–153.

[Pan et al., 1993] Pan, V.Y., Sadikou, A., Landowne, E., and Tiga, O. 1993. A new approach to fast polynomial interpolation and multipoint evaluation. *Comp. & Math. (with Appl.)*, 25(9):25–30.

[Pan et al., 1995] Pan, V.Y., Zheng, A.L., Dias, O., and Huang, X.H. 1995. A fast, preconditioned conjugate gradient Toeplitz and Toeplitz-like solver, *Comp. & Math. (with Appl.)*, 30(8):57–63.

[Pan et al., 1997] Pan, V.Y., Zheng, A.L., Huang, X.H., and Yu, Y.Q. 1997. Fast multipoint polynomial evaluation and interpolation via computations with structured matrices. *Annals of Numerical Mathematics*, 4:483–510.

[Pan et al., 1997a] Pan, V.Y., Zheng, A.L., Huang, X.H., and Dias, O. 1997. Newton's Iteration for Inversion of Cauchy-like and Other Structured Matrices, *J. of Complexity*, 13:108–124.

[Press et al., 1992] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. 1988 and 2nd edition, 1992. *Numerical Recipes in C : the Art of Scientific Computing.* Cambridge University Press, Cambridge.

[Rokhlin, 1985] Rokhlin, F. 1985. Rapid solution of integral equations of classical potential theory. *J. Comput. Physics*, 60:187–207.

[Runge and König, 1924] Runge, C. and König, H. 1924. *Die Grundlehren der mathematischen Wissenshaften*, 11. Springer, Berlin.

[Schönhage et al., 1994] Schönhage, A., Grotefeld, A.F.W., and Vetter, E. 1994. *Fast Algorithms: A Multitape Turing Machine Implementation.* Wissenschaftsverlag, Mannheim, Germany.

[Schönhage and Strassen, 1971] Schönhage, A. and Strassen, V. 1971. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292. In German.

[Strang, 1993] Strang, G. 1993. Wavelet transforms versus Fourier transforms. *Bulletin (New Series) of the American Mathematical Society*, 28(2):288-305.

[Swarztrauber, 1984]
Swarztrauber, P. 1984. FFT algorithms for vector computers. *Parallel Computing*, 1:45–63. Implementation at `http://www.psc.edu/general/software/packages/fftpack/fftpack.html` or `ftp://netlib.att.com/netlib`.

[Toom, 1963] Toom, A.L. 1963. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Math. Doklady*, 3:714–716, 1963.

[Van Loan, 1992] Van Loan, C.F. 1992. *Computational Frameworks for the Fast Fourier Transform.* SIAM Publications, Philadelphia, Penn.

[Wiedemann, 1986] Wiedemann, D.H. 1986. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62.

[Winograd, 1980] Winograd, S. 1980. *Arithmetic Complexity of Computations.* SIAM, Philadelphia.

[Zippel, 1993] Zippel, R. 1993. *Effective Polynomial Computation.* Kluwer Academic Publishers, Boston.

## Further Information

The main research journals in this area are "Computers and Mathematics (with Applications)," the "Journal of the ACM," the "Journal of Symbolic Computation," "Linear Algebra and Its Applications," "Mathematics of Computation," the "SIAM Journal of Computing" the "SIAM Journal of Matrix Analysis and Applications," and "Theoretical Computer Science." New implementations are reported in the "ACM Transactions on Mathematical Software."

The main annual conferences in this area are the "ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation," the "ACM Symposium on Theory of Computing," the "ACM-SIAM Symposium on Discrete Algorithms," the "European Symposium on Algorithms," the "IEEE Symposium on Foundations of Computer Science," and the "Symposium on Parallel Algorithms and Architectures."

The following books contain general information on the topics of this chapter [Aho et al., 1974, Borodin and Munro, 1975, Buchberger et al., 1982, Zippel, 1993, Bini and Pan, 1994, Knuth, 1997], where the last three present the state of the art and [Bini and Pan, 1994] points to an extensive list of applications. Some references on transforms and convolution have been listed at the beginning of Section 2.

Parallel computation is an important subject, which we have barely touched. We refer the interested reader to [Ja Ja, 1992, Pan, 1992a, Bini and Pan, 1994, Pan, 1996].