An Inversion Formula and Fast Algorithms for Cauchy-Vandermonde Matrices

Tilo Finck, Georg Heinig, and Karla Rost
Technische Universität Chemnitz
Fachbereich Mathematik
PSF 964, Chemnitz, D-0-9010, Germany

Submitted by Vlastimil Pták

ABSTRACT

Matrices of composed type consisting of a Vandermonde and a Cauchy part and their connection to rational interpolation are investigated. An inversion formula is presented, and fast solution algorithms with $O(n \log^2 n)$ complexity for Cauchy-Vandermonde systems are developed.

1. INTRODUCTION

To start with let us introduce some notation. For a given vector $c = (c_j)_1^n \in \mathbb{C}^n$, let $V_k(c)$ denote the $n \times k$ Vandermonde matrix

$$V_k(c) = \begin{bmatrix} 1 & c_1 & \cdots & c_1^{k-1} \\ 1 & c_2 & \cdots & c_2^{k-1} \\ \vdots & \vdots & & \vdots \\ 1 & c_n & \cdots & c_n^{k-1} \end{bmatrix}.$$

For two vectors $c \in \mathbb{C}^n$ and $d = (d_j)_1^l$ for which $c_i \neq d_j$ $(i = 1, \ldots, n;$

655 Avenue of the Americas, New York, NY 10010 0024-3795/93/$6.00

$j = 1, \ldots, l$), let $C(c, d)$ denote the Cauchy matrix

$$C(c,d) = \begin{bmatrix} \dfrac{1}{c_1 - d_1} & \cdots & \dfrac{1}{c_1 - d_l} \\ \vdots & & \vdots \\ \dfrac{1}{c_n - d_1} & \cdots & \dfrac{1}{c_n - d_l} \end{bmatrix}.$$

The matrices under considerations are of the form

$$A = \begin{bmatrix} C(c,d) & V_k(c) \end{bmatrix}, \tag{1.1}$$

where $k + l = n$ and the components of c and of d are pairwise distinct. In what follows matrices (1.1) will be called Cauchy-Vandermonde matrices, or briefly CV matrices.

CV matrices appear in connection with rational interpolation problems, as will be shown in Section 2, but also in connection with numerical solution of singular integral equations [6]. They were studied in the paper [2], where the method of UV reduction developed in [5] was the main tool of investigation. Also in [2], inversion formulas and inversion algorithms with $O(n^2)$ complexity were presented.

In the present paper we utilize the connection between CV matrices and rational interpolation. This leads to a more instructive and complete inversion formula. More precisely, it turns out that A^{-1} is closely related to the transposed matrix A^T via the relation

$$A^{-1} = \begin{bmatrix} D_1 & 0 \\ 0 & H(s) \end{bmatrix} A^T D_2, \tag{1.2}$$

where D_1 and D_2 are diagonal matrices and $H(s)$ is a triangular Hankel matrix. Moreover, the entries of these matrices can be expressed in terms of the vectors c and d.

The formula (1.2) allows us to construct fast solution algorithms for systems

$$Ax = y \tag{1.3}$$

and

$$A^T x = y. \tag{1.4}$$

We show that there are algorithms with $O(n \log^2 n)$ complexity for sequential and with $O(n)$ complexity for parallel computers with $O(n)$ processors.

The fact that Cauchy systems can be solved with $O(n \log^2 n)$ complexity has been already noted in the literature (see [10, 8]). It is related to the so-called Trummer's problem, which asks for the complexity of the multiplication of a Cauchy matrix by a vector (see [3, 4]).

2. CV MATRICES AND INTERPOLATION

The equation (1.3) is closely related to the following interpolation problem.

PROBLEM I. We are given complex numbers y_i, d_j $(i = 1, \ldots, n;\ j = 1, \ldots, l)$, and ask for $x = (x_i)_1^n \in \mathbb{C}^n$ such that the rational function

$$F(\lambda) = \sum_{j=1}^{l} \frac{x_j}{\lambda - d_j} + \sum_{j=1}^{k} x_{l+j} \lambda^{j-1} \tag{2.1}$$

meets the conditions

$$F(c_i) = y_i \qquad (i = 1, \ldots, n). \tag{2.2}$$

The following fact is obvious.

PROPOSITION 2.1. *The vector x is a solution of* (1.3), $y = (y_i)_1^n$, *if and only if x is a solution of Problem I.*

For completeness let us show that equation (1.4) can also be interpreted as an interpolation problem. Clearly, (1.4) can be written as the two equations

$$C(c, d)^T x = w \tag{2.3}$$

and

$$V_k(c)^T x = z, \tag{2.4}$$

where $w = (y_i)_1^l$, $z = (y_{i+l})_1^k$. We introduce the rational function

$$f(\lambda) = \sum_{j=1}^{n} \frac{x_j}{c_j - \lambda}. \tag{2.5}$$

Then (2.3) is equivalent to the interpolation conditions

$$f(d_i) = y_i \qquad (i = 1, \ldots, l). \tag{2.6}$$

Furthermore, if

$$f(\lambda) = f_1\lambda^{-1} + f_2\lambda^{-2} + \cdots$$

is the Laurent series expansion at infinity for $f(\lambda)$, then (2.4) is satisfied if and only if $f_i = -y_{i+l}$ $(i = 1, \ldots, k)$. Thus we arrived at the following problem.

PROBLEM II. We are given complex numbers y_i, c_i $(i = 1, \ldots, n)$ and d_j $(j = 1, \ldots, l)$, and ask for $x = (x_i)_1^n \in \mathbb{C}^n$ such that $f(\lambda)$ defined by (2.5) fulfills (2.6) and $y_{l+1}, \ldots, y_n$ are the first coefficients of the Laurent series expansion of $-f(\lambda)$ at infinity.

PROPOSITION 2.2. *The vector x is a solution of* (1.4), $y = (y_i)_1^n$, *if and only if x is a solution of Problem II.*

3. INVERSION FORMULA

In this section we deduce an inversion formula for matrices (1.1). For this we utilize the connection between the equation (1.3) and Problem I.

Introduce the polynomial

$$h(\lambda) = \prod_{j=1}^{l} (\lambda - d_j). \tag{3.1}$$

Then any function $F(\lambda)$ of the form (2.1) admits a representation $F(\lambda) = p(\lambda)/h(\lambda)$, where $p(\lambda)$ is a polynomial with $\deg p(\lambda) < n$. Now (2.2) is

equivalent to

$$p(c_i) = y_i h(c_i) \qquad (i = 1, \ldots, n), \tag{3.2}$$

which is a classical Lagrange interpolation problem. Introducing the polynomial

$$g(\lambda) = \prod_{i=1}^{n} (\lambda - c_i)$$

its solution can be written as follows:

$$p(\lambda) = \sum_{i=1}^{n} \frac{h(c_i)}{g'(c_i)} \frac{g(\lambda)}{\lambda - c_i} y_i. \tag{3.3}$$

For the first l coefficients x_j in (2.1) we get

$$x_j = \frac{p(d_j)}{h'(d_j)} = \frac{g(d_j)}{h'(d_j)} \sum_{i=1}^{n} \frac{1}{d_j - c_i} \frac{h(c_i)}{g'(c_i)} y_i \qquad (j = 1, \ldots, l). \tag{3.4}$$

The remaining coefficients x_j $(j = l + 1, \ldots, n)$ are obtained by polynomial division. Introducing $v(\lambda) = \sum_{j=1}^{k} x_{l+j} \lambda^{j-1}$, we have

$$p(\lambda) = h(\lambda) v(\lambda) + r(\lambda) \tag{3.5}$$

for a certain polynomial $r(\lambda)$ with $\deg r(\lambda) < l$. Denoting

$$p(\lambda) = \sum_{i=1}^{n} p_i \lambda^{i-1}, \qquad h(\lambda) = \sum_{i=1}^{l+1} h_i \lambda^{i-1}, \qquad g(\lambda) = \sum_{i=1}^{n+1} g_i \lambda^{i-1},$$

and comparing coefficients, we get the equivalence of (3.5) to the equation

$$\tilde{p} = T(h) v, \tag{3.6}$$

where $\tilde{p} = (p_i)_{l+1}^{n}$, $v = (x_{l+1})_1^k$, and $T(h)$ denotes the $k \times k$ triangular

Toeplitz matrix

$$T(h) = \begin{bmatrix} 1 & h_l & \cdots & h_{l-k+2} \\ & 1 & \ddots & \vdots \\ & & \ddots & h_l \\ & 0 & & 1 \end{bmatrix}.$$

Moreover, for a given vector $s = (s_i)_1^k$ we denote by $H(s)$ the Hankel matrix

$$H(s) = \begin{bmatrix} s_1 & s_2 & \cdot & \cdot & \cdot & s_k \\ s_2 & & & & \cdot & \\ \cdot & & & \cdot & & \\ \cdot & & \cdot & & & \\ \cdot & \cdot & & & & \\ s_k & & & & & 0 \end{bmatrix}.$$

Now we are in a position to prove the following inversion formula.

THEOREM 3.1. *Let A be the* CV *matrix defined by* (1.1). *Then A is regular, and its inverse is given by*

$$A^{-1} = \begin{bmatrix} -D_1 & 0 \\ 0 & H(s) \end{bmatrix} A^T D_2, \tag{3.7}$$

where D_1, D_2 are the diagonal matrices

$$D_1 = \operatorname{diag}\left(\frac{g(d_1)}{h'(d_1)}, \ldots, \frac{g(d_l)}{h'(d_l)} \right), \qquad D_2 = \operatorname{diag}\left(\frac{h(c_1)}{g'(c_1)}, \ldots, \frac{h(c_n)}{g'(c_n)} \right),$$

and $s = (s_i)_1^k$ is the solution of $T(h)s = \tilde{g}$, $\tilde{g} = (g_{i+l+1})_{i=1}^k$.

Proof. First let us prove the nonsingularity of A. The equality $Ax = 0$ is equivalent to $F(c_i) = 0$ $(i = 1, \ldots, n)$ for the function (2.1), which implies $p(c_i) = 0$ $(i = 1, \ldots, n)$ for the numerator $p(\lambda)$ of $F(\lambda)$. Since $\deg p < n$ we get $p(\lambda) \equiv 0$; consequently, $x = 0$ and the regularity is proved.

Let now

$$x = \begin{bmatrix} u \\ v \end{bmatrix} \qquad (u \in \mathbb{C}^l, \quad v \in \mathbb{C}^k)$$

be a solution of (1.3). Then (3.4) can be written in the form

$$u = -D_1 C(c,d)^T D_2 y. \tag{3.8}$$

It remains to establish the equality

$$v = H(s) V_k(c)^T D_2 y. \tag{3.9}$$

The equalities (3.8) and (3.9) together provide (3.7).

In order to prove (3.9) we compare the first k coefficients of the Laurent series expansions at infinity of

$$\frac{p(\lambda)}{g(\lambda)} = \sum_{i=1}^{n} \frac{1}{\lambda - c_i} \frac{h(c_i)}{g'(c_i)} y_i. \tag{3.10}$$

Let $p(\lambda)/g(\lambda) = w_1 \lambda^{-1} + w_2 \lambda^{-2} + \cdots$ and $w = (w_i)_1^k$. Then (3.10) implies

$$w = V_k(c)^T D_2 y. \tag{3.11}$$

Furthermore, we obtain from (3.10)

$$p(\lambda) = \left[w_1 \lambda^{-1} + \cdots + w_k \lambda^{-k} + o(\lambda^{-k}) \right] g(\lambda).$$

Comparing the coefficients in this relation, we get

$$\tilde{p} = H(\tilde{g}) w. \tag{3.12}$$

The equalities (3.6), (3.11), and (3.12) lead to

$$v = T(h)^{-1} H(\tilde{g}) V_k(c)^T D_2 y.$$

It remains to check that $T(h)^{-1} H(\tilde{g}) = H(s)$ to get (3.9) and therefore (3.7). ∎

COROLLARY 3.1. *In case $k = 0$ $(l = n)$ we get the following inversion formula for Cauchy matrices*:

$$C(c, d)^{-1} = -D_1 C(c, d)^T D_2. \tag{3.13}$$

In case $l = 0$ $(k = n)$ we get

$$V_n(c)^{-1} = H(s) V_n(c)^T D^{-1}, \tag{3.14}$$

where $s = (g_{i+1})_1^n$, $D = \operatorname{diag}(g'(c_1), \ldots, g'(c_n))$.

Formulas for the inverse of Vandermonde matrices coming from the Lagrange form of the interpolating polynomial are well known; see e.g. [9]. Inversion formulas of the form (3.13), (3.14) have been already presented in [5, II, 2.3, 2.4]. The formula (3.13) can be found (in another form) in [7].

For the future it seems to be of interest to describe all matrices the inverses of which equal their transposes up to diagonal and triangular Hankel factors.

Let us note a representation of Cauchy matrices, which is as far as we know new.

PROPOSITION 3.2. *The $n \times n$ Cauchy matrix $C(c, d)$ admits the representation*

$$C(c, d) = D_1^{-1} V_n(c) H(h) V_n(d)^T, \tag{3.15}$$

where $h = (h_{i+1})_1^n$ is the vector of the coefficients of $h(\lambda)$, $D_1 = \operatorname{diag}(h(c_1), \ldots, h(c_n))$.

Proof. We make use of the above arguments for the case $k = 0$. Suppose that $C(c, d)x = y$. Then (3.2) holds and

$$x_i = \frac{p(d_i)}{h'(d_i)}.$$

This relation and (3.2) can be written in the form

$$V_n(c)\, p = D_1 y, \qquad V_n(d)\, p = D_2 x,$$

$$D_2 = \operatorname{diag}(h'(d_1), \ldots, h'(d_n)).$$

This implies

$$C(c, d) = D_1^{-1} V_n(c) V_n(d)^{-1} D_2.$$

Replacing $V_n(d)^{-1}$ according to (3.14), we get (3.15). ■

4. ALGORITHMS

In this section we discuss how the inversion formula (3.7) can be utilized to solve the equations (1.3) and (1.4). Our main result will be the following.

THEOREM 4.1. *An equation of the form* (1.3) *or* (1.4) *can be solved with* $O(n \log^2 n)$ *complexity on a sequential computer with* $O(n)$ *complexity on an n-processor parallel computer.*

Now we are going to prove this assertion.

To compute the solution $x = A^{-1}y$ or $x = (A^T)^{-1}y$ with the help of (3.7) we need the following procedures:

(1) Formation of a polynomial from its roots [to get $g(\lambda)$ and $h(\lambda)$].
(2) Computation of n (or l) values of a given polynomial (to get D_1 and D_2). This means the multiplication of a column vector by a Vandermonde matrix.
(3) Multiplication of a column vector by a triangular Hankel matrix [$H(s)$].
(4) Solution of a triangular Toeplitz system [to get s].
(5) Multiplication of a column vector by a Cauchy matrix.
(6) Multiplication of a column vector by a transposed Vandermonde matrix.

The remaining computations have complexity $O(n)$ for sequential and $O(1)$ for parallel computers.

Leg $\alpha_i(n)$ be the amount of computation required for the evaluation of the n unknowns of problem (i), $i = 1, \ldots, 6$. We show now that the complexity for solving each of these six problems is equal or less than stated in Theorem 4.1.

(1): The formation of $g(\lambda) = \prod_{i=1}^{n}(\lambda - c_i)$ can be reduced to the formation of $g_1(\lambda) = \prod_{i=1}^{m}(\lambda - c_i)$ and $g_2(\lambda) = \prod_{i=m+1}^{n}(\lambda - c_i)$ followed by

the multiplication $g(\lambda) = g_1(\lambda)g_2(\lambda)$. We choose m as the integer part of $n/2$. If the multiplication is carried out with the help of the FFT, it requires $O(n \log n)$ flops (see e.g. [1]). Thus we get a recursion

$$\alpha_1(n) = 2\alpha_1\left(\frac{n}{2}\right) + O(n \log n).$$

This leads to $\alpha_1(n) = O(n \log^2 n)$ for sequential computation.

In parallel processing it is recommended to compute $g(\lambda)$ by classical recursion via

$$g^1(\lambda) = \lambda - c_1, \qquad g^{k+1}(\lambda) = g^k(\lambda)(\lambda - c_{k+1}).$$

In that way $g^n(\lambda) = g(\lambda)$ is evaluated in n steps with n processors.

(2): An $O(n \log^2 n)$ procedure to compute n values of a polynomial with degree n is presented in [1]. The idea is to apply a divide and conquer strategy, which means the reduction to two problems of size about $n/2$ and $O(n \log n)$ flops. Let us briefly describe this.

Suppose we have to compute $h(c_i)$ $(i = 1, \ldots, n)$, where $h(\lambda)$ is a polynomial of degree $\leqslant n$. Let $g(\lambda)$, $g_1(\lambda)$, $g_2(\lambda)$ be the polynomials introduced in (1). We make two polynomial divisions

$$h(\lambda) = q(\lambda)g_1(\lambda) + h_1(\lambda), h(\lambda) = p(\lambda)g_2(\lambda) + h_2(\lambda), \quad (4.1)$$

where $\deg h_1(\lambda) < m$, $\deg h_2(\lambda) < n - m$. Now we have

$$h(c_i) = \begin{cases} h_1(c_i), & i = 1, \ldots, m, \\ h_2(c_1), & i = m + 1, \ldots, n. \end{cases}$$

For $m = [n/2]$ we get that $\alpha_2(n)$ equals $2\alpha_2(n/2)$ plus the amount for polynomial division. It remains to show that this amount is at most $O(n \log n)$.

For polynomial division one can apply Cook's algorithm (see [1]), which has just complexity $O(n \log n)$. But it is also possible to reduce polynomial division to the problems (3) and (4). Let us show this. Suppose that

$$h(\lambda) = \sum_{i=0}^{n} h_i \lambda^i, \qquad g_1(\lambda) = \sum_{i=0}^{m} a_i \lambda^i,$$

$$q(\lambda) = \sum_{i=0}^{n-m} q_i \lambda^i, \qquad h_1(\lambda) = \sum_{i=0}^{m-1} r_i \lambda^i.$$

Then, comparing the coefficients in (4.1), we get the two equations

$$\begin{bmatrix} h_m \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} a_m & \cdots & a_o \\ & \ddots & \vdots \\ 0 & & a_m \end{bmatrix} \begin{bmatrix} q_o \\ \vdots \\ q_{n-m} \end{bmatrix} \tag{4.2}$$

and

$$\begin{bmatrix} r_o \\ \vdots \\ r_{n-m} \\ \vdots \\ r_{m-1} \end{bmatrix} = - \begin{bmatrix} a_o & & \\ \vdots & \ddots & 0 \\ a_{n-m} & \cdots & a_o \\ \vdots & & \vdots \\ a_{m-1} & \cdots & a_{2m-n-1} \end{bmatrix} \begin{bmatrix} q_o \\ \vdots \\ q_{n-m} \end{bmatrix} + \begin{bmatrix} h_o \\ \vdots \\ h_{n-m} \end{bmatrix} \tag{4.3}$$

for $m \geqslant \frac{1}{2}(n+1)$, and similar equations for $m < \frac{1}{2}(n+1)$. In order to get $q(\lambda)$ we have to solve the triangular Toeplitz system (4.2), and to get $h_1(\lambda)$ we have to multiply a vector by triangular Toeplitz matrices, which is in principle problem (3). In parallel processing the multiplication of a vector by a matrix requires, clearly, $O(n)$ steps if n processors are provided.

(3): The multiplication

$$\begin{bmatrix} a_1 & & 0 \\ \vdots & \ddots & \\ a_n & \cdots & a_1 \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

means to take the first n coefficients of $a(\lambda)u(\lambda)$, where $a(\lambda) = \sum_{j=1}^{n} a_j \lambda^{j-1}$, $u(\lambda) = \sum_{j=1}^{n} u_j \lambda^{j-1}$. Using the FFT, this can be done with $O(n \log n)$ flops. In parallel processing only $O(\log n)$ flops are required to compute a product of polynomials if the FFT is utilized. It remains to note that multiplication of the equality above from the left by $J = [\delta_{i,n-j+1}]_1^n$ leads to problem (3).

(4): Let T be an upper triangular Toeplitz matrix. In order to solve $Tx = y$ we determine the upper triangular Toeplitz matrix T^{-1} and compute $T^{-1}y$. Now T^{-1} is given by its last column, which is the solution of $Tu = e$, where e denotes always the last unit vector. We divide T into blocks of almost equal size, say here, for simplicity,

$$T = \begin{bmatrix} T_1 & T_2 \\ 0 & T_1 \end{bmatrix}, \qquad u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}.$$

Then $Tu = e$ is equivalent to

$$T_1 u_2 = e, \qquad u_1 = -T_1^{-1} T_2 u_2.$$

That means we reduce the computation of T^{-1} to that of T_1^{-1} plus three multiplications of a vector by a triangular Toeplitz matrix. Thus $\alpha_4(n) = \alpha_4(n/2) + O(n \log n)$, which implies $\alpha_4(n) = O(n \log n)$.

(5): To compute the products $y = C(c, d)x$, where

$$C(c, d) = \left[(c_i - d_j)^{-1}\right]_{i=1,j=1}^{k,n} (k \leqslant n),$$

we again apply the divide and conquer principle. Start with computing the numerator $p(\lambda)$ of

$$\sum_{j=1}^{n} \frac{x_j}{\lambda - d_j} = \frac{p(\lambda)}{h(\lambda)}. \tag{4.4}$$

Let $h_1(\lambda) = \prod_{j=1}^{m}(\lambda - d_j)$, $h_2(\lambda) = \prod_{j=m+1}^{n}(\lambda - d_j)$. In order to get $p(\lambda)$ we evaluate $p_1(\lambda)$, $p_2(\lambda)$ given by

$$\sum_{j=1}^{m} \frac{x_j}{\lambda - d_j} = \frac{p_1(\lambda)}{h_1(\lambda)}, \qquad \sum_{j=m+1}^{N} \frac{x_j}{\lambda - d_j} = \frac{p_2(\lambda)}{h_2(\lambda)}.$$

Then we have $p(\lambda) = h_2(\lambda)p_1(\lambda) + h_1(\lambda)p_2(\lambda)$. So we divide the problem of size n into two problems of half the size and two polynomial multiplications. That means $\alpha_5(n) = 2\alpha_5(n/2) + O(n \log n)$, which implies $\alpha_5(n) = O(n \log^2 n)$. From $p(\lambda)$ we get $y = (y_i)_1^k$ by $y_i = p(c_i)/h(c_i)$ using an algorithm described under (2).

Let us note that an $O(n \log n)$ algorithm to compute $C(c, d)x$ was first presented in [3].

(6): We discuss now the computation of $y = V_k(d)^T x$, where $d = (d_i)_1^n$, $x = (x_i)_1^n$, $y = (y_i)_1^k$ $(k \leqslant n)$. For simplicity we assume $k = n$. Using ideas like those in Section 2, Problem II, we have to compute the numerator $p(\lambda) = \sum_{i=1}^{n} p_i \lambda^{i-1}$ of (4.4). This can be done in the same manner as explained under (5). It remains to state that $y_1, \ldots, y_n$ are the first coefficients of the Laurent series expansion of $f(\lambda) = p(\lambda)/h(\lambda)$ at infinity, i.e. $f(\lambda) = y_1\lambda^{-1} + y_2\lambda^{-2} + \cdots$. Comparing coefficients, we obtain that y is the solution of

$$H(\tilde{h})y = p, \tag{4.5}$$

where $p = (p_i)_1^n$, and $\tilde{h} = (h_{i+1})_1^n$ denotes the vector of the last n coefficients of $h(\lambda)$. How to solve (4.5) is in principle explained under (4).

REFERENCES

1 A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1976.

2 T. Finck and K. Rost, Fast inversion of Cauchy-Vandermonde matrices, *Sem. Anal., Oper. Equations and Numer. Anal.*, Karl-Weierstrass-Inst. für Mathematik, Akad. Wiss., Berlin, pp. 69–79 (1990).

3 N. Gastinel, Inversion d'une matrice généralisant la matrice de Hilbert, *Chiffres* 3:149–152 (1960).

4 A. Gerasoulis, M. D. Grigoriadis, and L. Sun, A fast algorithm for Trummer's problem, *SIAM J. Sci. Statist. Comput.* 8(1):135–138 (1987).

5 G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Akademie-Verlag, Berlin, Birkhäuser, Basel, 1984.

6 P. Junghanns and D. Oestreich, Numerische Lösung des Staudammproblems mit Drainage, *Z. Angew. Math. Mech.* 69:83–92 (1989).

7 D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

8 V. Pan, On computations with dense structured matrices, *Math. Comp.* 55 (191):179–190 (1990).

9 J. F. Traub, Associated polynomials and uniform methods for the solution of linear problems, *SIAM Rev.* 8:277–301 (1966).

10 Z. Vavřin, Remarks on complexity of polynomial and special matrix computations, *Linear Algebra Appl.* 122/123/124:539–564 (1989).

Received 27 February 1992; final manuscript accepted 7 April 1992