MATHEMATICS OF COMPUTATION, VOLUME 24, NUMBER 112, OCTOBER 1970

Solution of Vandermonde Systems of Equations

By Åke Björck* and Victor Pereyra

Abstract. We obtain in this paper a considerable improvement over a method developed earlier by Ballester and Pereyra for the solution of systems of linear equations with Vandermonde matrices of coefficients. This is achieved by observing that a part of the earlier algorithm is equivalent to Newton's interpolation method. This allows also to produce a progressive algorithm which is significantly more efficient than previous available methods. Algol-60 programs and numerical results are included. Confluent Vandermonde systems are also briefly discussed.

Introduction. In [1] an algorithm was derived for solving a Vandermonde system of equations

$$V\mathbf{x} = \mathbf{b}, \tag{1}$$

where $V = V(\alpha_0, \alpha_1, \cdots, \alpha_n)$ is the Vandermonde matrix

$$V(\alpha_0, \alpha_1, \cdots, \alpha_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_n \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_0^n & \alpha_1^n & \cdots & \alpha_n^n \end{bmatrix}. \tag{2}$$

These systems, and the corresponding dual systems

$$V^T\mathbf{a} = \mathbf{f} \tag{3}$$

appear naturally, and have to be solved, in many applications. Some important examples are interpolation, construction of spline functions [3], approximation of linear functionals [1], [5], etc.

In this paper a new algorithm for solving (1) will be developed which is faster, needs less storage and, from experimental results, is more accurate than that in [1]. Also, a progressive version of the algorithm will be given, allowing the updating of a solution $\mathbf{x}$ when a new value of α is added. Corresponding algorithms for the dual system (3) will also be given. They compare very favorably with the algorithm proposed in [4].

Only the nonconfluent case, $\alpha_i \neq \alpha_j$ when $i \neq j$, will be treated in detail here, although the generalization to the confluent case will be outlined. In a separate paper Galimberti and Pereyra [8] consider in detail the general confluent case (of Hermite type) with an approach similar to [1]. The original matrix is reduced to block triangular form with diagonal blocks being nonconfluent Vandermonde systems to which the algorithm of this paper is applied. Also Galimberti and Pereyra in [7] use the method of this paper in the solution of multidimensional Vandermonde

Received January 30, 1970.

AMS 1968 *subject classifications.* Primary 6535, 6520.

Key words and phrases. Vandermonde systems, confluent Vandermonde systems.

* The work of the first author was supported in part by the Sweden-America Foundation.

systems. In [9], Gustafson develops algorithms and computer programs for the confluent case.

1. Algorithms for Solving Vandermonde Systems. We will first derive an algorithm for the dual system (3). Let $\mathbf{a}$ be the solution to (3) and introduce the polynomial

$$P(z) = (1, z, \cdots, z^n)\mathbf{a}.$$

Then $P(z)$ is the unique interpolating polynomial of degree at most n, such that

$$P(\alpha_k) = f_k, \qquad k = 0, 1, \cdots, n.$$

One of the most efficient ways to determine $P(z)$ is by Newton's method. We introduce the polynomials

$$Q_0(z) = 1, \qquad Q_k(z) = \prod_{i=0}^{k-1} (z - \alpha_i), \qquad k = 1, 2, \cdots, n, \tag{4}$$

and write $P(z)$ in the form

$$P(z) = (Q_0(z), Q_1(z), \cdots, Q_n(z))\mathbf{c}, \tag{5}$$

where c_k, $k = 0, 1, \cdots, n$, are the divided differences of kth order,

$$c_k = f[\alpha_0, \alpha_1, \cdots, \alpha_k].$$

It is well known that these divided differences can be recursively generated from the relation

$$f[\alpha_{j-k-1}, \cdots, \alpha_j] = \frac{f[\alpha_{j-k}, \cdots, \alpha_j] - f[\alpha_{j-k-1}, \cdots, \alpha_{j-1}]}{\alpha_j - \alpha_{j-k-1}}. \tag{6}$$

When $\mathbf{c}$ is known, we can use, essentially, Horner's scheme to evaluate $P(z)$. We have $P(z) = q_0(z)$, where

$$q_n(z) = c_n, \qquad q_k(z) = (z - \alpha_k)q_{k+1} + c_k, \qquad k = n - 1, \cdots, 1, 0. \tag{7}$$

If we substitute here

$$q_k(z) = a_k^{(k)} + a_{k+1}^{(k)} z + \cdots + a_n^{(k)} z^{n-k}, \tag{8}$$

then we get a recurrence relation for computing the unknowns $a_k = a_k^{(0)}$.

Now introduce, for $k = 0, 1, \cdots, n$, the vectors

$$\mathbf{c}^{(k)} = (c_0, \cdots, c_k, f[\alpha_1, \cdots, \alpha_{k+1}], \cdots, f[\alpha_{n-k}, \cdots, \alpha_n])^T$$

and

$$\mathbf{a}^{(k)} = (c_0, \cdots, c_{k-1}, a_k^{(k)}, \cdots, a_n^{(k)})^T.$$

From these definitions it follows immediately that

$$\mathbf{c}^{(0)} = \mathbf{f}, \qquad \mathbf{c}^{(n)} = \mathbf{a}^{(n)} = \mathbf{c}, \qquad \mathbf{a}^{(0)} = \mathbf{a},$$

and from (6), (7) and (8) we get:

Algorithm for the Dual System.

Step (i). Put $\mathbf{c}^{(0)} = \mathbf{f}$ and, for $k = 0, 1, \cdots, n - 1$, compute

$$\begin{aligned} c_j^{(k+1)} &= (c_j^{(k)} - c_{j-1}^{(k)})/(\alpha_j - \alpha_{j-k-1}), && j = n, n-1, \cdots, k+1, \\ &= c_j^{(k)}, && j = k, \cdots, 1, 0. \end{aligned} \tag{9}$$

Step (ii). Put $\mathbf{a}^{(n)} = \mathbf{c}$ and, for $k = n-1, \cdots, 1, 0$, compute

$$\begin{aligned} a_j^{(k)} &= a_j^{(k+1)}, && j = 0, 1, \cdots, k-1, n, \\ &= a_j^{(k+1)} - \alpha_k a_{j+1}^{(k+1)}, && j = k, k+1, \cdots, n-1. \end{aligned} \tag{10}$$

Now define the lower bidiagonal matrix $L_k(\alpha)$ of order $n+1$ by

$$L_k(\alpha) = \begin{bmatrix} 1 & & & & & & \\ 0 & 1 & & & & 0 & \\ & 0 & \ddots & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & -\alpha & \ddots & & \\ & 0 & & & \ddots & & \\ & & & & & 1 & \\ & & & & & -\alpha & \end{bmatrix} \quad k\text{th row.} \tag{11}$$

kth column

It then follows from (9) that the vectors $\mathbf{c}^{(k)}$ are recursively generated by

$$\mathbf{c}^{(0)} = \mathbf{f}, \qquad \mathbf{c}^{(k+1)} = D_k^{-1} M_k \mathbf{c}^{(k)}, \qquad k = 0, 1, \cdots, n-1,$$

where

$$M_k = L_k(1), \qquad D_k = \operatorname{diag}\{1, \cdots, 1, (\alpha_{k+1} - \alpha_0), \cdots, (\alpha_n - \alpha_{n-k-1})\}.$$

Also from (10) it follows that

$$\mathbf{a}^{(n)} = \mathbf{c}, \qquad \mathbf{a}^{(k)} = N_k^T \mathbf{a}_k^{(k+1)}, \qquad k = n-1, \cdots, 1, 0,$$

where

$$N_k = L_k(\alpha_k).$$

Collecting these results, we find that Newton's algorithm for solving (3) can be written in matrix form as

$$\mathbf{c} = U^T \mathbf{f}, \qquad \mathbf{a} = L^T \mathbf{c}, \tag{12}$$

where

$$U^T = D_{n-1}^{-1} M_{n-1} \cdots D_0^{-1} M_0, \qquad L^T = N_0^T N_1^T \cdots N_{n-1}^T \tag{13}$$

are lower triangular and upper unit triangular, respectively. Since $\mathbf{a} = V^{-T}\mathbf{f}$, we have $V^{-T} = L^T U^T$ or

$$V^{-1} = UL, \qquad V = L^{-1} U^{-1}.$$

Thus, we have found a factorization in bidiagonal factors of the unique triangular matrices in the UL-decomposition of the inverse Vandermonde matrix V^{-1}.

The factorization of V^{-1} can obviously be used to write down an algorithm for solving (1). From

$$\mathbf{x} = V^{-1}\mathbf{b} = (M_0^T D_0^{-1} \cdots M_{n-1}^T D_{n-1})(N_{n-1} \cdots N_1 N_0) b,$$

we obtain the recurrence relations for computing $\mathbf{x} = \mathbf{x}^{(0)}$,

$$\mathbf{d}^{(0)} = \mathbf{b}, \qquad \mathbf{d}^{(k+1)} = N_k\mathbf{d}^{(k)}, \qquad k = 0, 1, \cdots, n-1,$$

$$\mathbf{x}^{(n)} = \mathbf{d}^{(n)}, \quad \mathbf{x}^{(k)} = M_k^T D_k^{-1}\mathbf{x}^{(k+1)}, \quad k = n-1, \cdots, 1, 0.$$

Writing down these relations in component form we get:

Algorithm for the Primal System.

Step (i). Put $\mathbf{d}^{(0)} = \mathbf{b}$ and, for $k = 0, 1, \cdots, n-1$, compute

$$\begin{aligned} d_j^{(k+1)} &= d_j^{(k)} - \alpha_k d_{j-1}^{(k)}, && j = n, n-1, \cdots, k+1, \\ &= d_j^{(k)}, && j = k, \cdots, 1, 0. \end{aligned} \tag{14}$$

Step (ii). Put $\mathbf{x}^{(n)} = \mathbf{d}^{(n)}$ and, for $k = n-1, \cdots, 1, 0$, compute

$$\begin{aligned} x_j^{(k+1/2)} &= x_j^{(k+1)}, && j = 0, 1, \cdots, k, \\ &= x_j^{(k+1)}/(\alpha_j - \alpha_{j-k-1}), && j = k+1, \cdots, n-1, n, \\ x_j^{(k)} &= x_j^{(k+1/2)}, && j = 0, 1, \cdots, k-1, n, \\ &= x_j^{(k+1/2)} - x_{j+1}^{(k+1/2)}, && j = k, \cdots, n-1. \end{aligned} \tag{15}$$

2. Progressive Algorithms. We now consider the problem of updating the solution when a new value α is added. We write the systems (1) and (3)

$$V_n\mathbf{x}_n = \mathbf{b}_n, \qquad V_n^T\mathbf{a}_n = \mathbf{f}_n,$$

where

$$V_n = V(\alpha_0, \alpha_1, \cdots, \alpha_n).$$

From well-known properties of the LU-decomposition and triangular matrices it follows that the decomposition $V_n^{-1} = U_nL_n$ can be written in partitioned form as

$$\left[\begin{array}{c|c} V_{n-1} & \begin{matrix}1\\ \vdots \\ \alpha_n^{n-1}\end{matrix} \\ \hline \alpha_0^n \cdots \alpha_{n-1}^n & \alpha_n^n \end{array}\right]^{-1} = \left[\begin{array}{c|c} U_{n-1} & \begin{matrix}u_0^{(n)}\\ \vdots \\ u_{n-1}^{(n)}\end{matrix} \\ \hline 0 & u_n^{(n)} \end{array}\right] \left[\begin{array}{c|c} L_{n-1} & 0 \\ \hline m_0^{(n)} \cdots m_{n-1}^{(n)} & m_n^{(n)} \end{array}\right].$$

In Step (i) of the algorithm for the dual problem we have

$$\mathbf{c}^{(n)} = U_n^T\mathbf{f}_n,$$

and thus, when α_n is added, the first $n-1$ components in $\mathbf{c}^{(n)}$ are unchanged and only $c_n^{(n)}$ has to be computed. If the quantities $c_{n-1}^{(k)}$, $k = 0, 1, \cdots, n-1$, have been saved, then (9) with $j = n$ can be used to compute $c_n^{(k)}$, $k = 0, 1, \cdots, n$. In part (ii) we have

$$\mathbf{a}_n = L_n^T\mathbf{c}^{(n)} = \begin{bmatrix}\mathbf{a}_{n-1}\\ 0\end{bmatrix} + c_n^{(n)}(\mathbf{m}^{(n)})^T,$$

where $\mathbf{m}^{(n)}$ is the last row in L_n. Thus *all* components in $\mathbf{a}^{(n)}$ will change and Step (ii) requires greater modification. From (4) and (12) it follows that

$$Q_n(z) = m_0^{(n)} + m_1^{(n)}z + \cdots + m_n^{(n)}z^n.$$

From this it is easy to derive a recursion formula to compute $\mathbf{m}^{(n)}$, and we get:

Progressive Algorithm for the Dual System. Put $a_0^{(0)} = c_0^{(0)} = f_0$, $m_0^{(0)} = 1$ and, for $n = 1, 2, 3, \cdots$, compute

$$\begin{aligned} &c_n^{(0)} = f_n, \qquad c_n^{(k+1)} = (c_n^{(k)} - c_{n-1}^{(k)})/(\alpha_n - \alpha_{n-k-1}), \quad k = 0, 1, \cdots, n-1, \\ &m_n^{(n)} = 1, \qquad a_n^{(n)} = c_n^{(n)}, \qquad m_{-1}^{(n-1)} = 0, \\ &m_k^{(n)} = m_{k-1}^{(n-1)} - \alpha_{n-1}m_k^{(n-1)}, \qquad a_k^{(n)} = a_k^{(n-1)} + m_k^{(n)}c_n^{(n)}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k = n-1, \cdots, 1, 0. \end{aligned} \tag{16}$$

For the primal system we have in Step (i) of the algorithm

$$\mathbf{d}^{(n)} = L_n\mathbf{b}_n.$$

Again, the first $n-1$ components are unchanged. Provided $d_{n-1}^{(k)}$, $k = 0, 1, \cdots, n-1$, have been saved, $d_n^{(k)}$, $k = 0, 1, \cdots, n$, can be computed by taking $j = n$ in (14). In Step (ii) we have

$$\mathbf{x}_n = U_n\mathbf{d}^{(n)} = \begin{bmatrix} \mathbf{x}_{n-1} \\ 0 \end{bmatrix} + d_n^{(n)}\mathbf{u}^{(n)},$$

where $\mathbf{u}^{(n)}$ is the last column in U_n. Thus *all* components in $\mathbf{x}_n$ change, and we must find a way to generate $\mathbf{u}^{(n)}$, $n = 0, 1, 2, \cdots$. Taking the last component of the relation $\mathbf{c}^{(n)} = U_n^T\mathbf{f}_n$ we get

$$f[\alpha_0, \alpha_1, \cdots, \alpha_n] = \sum_{k=0}^{n} u_k^{(n)}f_k.$$

Thus $u_k^{(n)}$ are coefficients which express the divided difference of nth order in terms of function values. It is a well-known result that these coefficients are

$$u_k^{(n)} = [(\alpha_k - \alpha_0)\cdots(\alpha_k - \alpha_{k-1})(\alpha_k - \alpha_{k+1})\cdots(\alpha_k - \alpha_n)]^{-1}.$$

Using this expression we easily derive:

Progressive Algorithm for the Primal System. Put $x^{(0)} = d^{(0)} = b_0$, $u_0^{(0)} = 1$ and, for $n = 1, 2, 3, \cdots$, compute

$$\begin{aligned} &d_n^{(0)} = b_n, \qquad d_n^{(k+1)} = d_n^{(k)} - \alpha_k d_{n-1}^{(k)}, \qquad k = 0, 1, \cdots, n-1, \\ &u_n^{(n)} = [(\alpha_n - \alpha_0)(\alpha_n - \alpha_1)\cdots(\alpha_n - \alpha_{n-1})]^{-1}, \qquad x_n^{(n)} = d_n^{(n)}u_n^{(n)}, \\ &u_k^{(n)} = u_k^{(n-1)}(\alpha_k - \alpha_n), \quad x_k^{(n)} = x_k^{(n-1)} + d_n^{(n)}u_k^{(n)}, \quad k = n-1, \cdots, 1, 0. \end{aligned} \tag{17}$$

3. Efficiency. In the nonprogressive algorithms for the primal and dual system we transform the right-hand side, by a sequence of simple transformations, into the solution vector. If the components are modified in a suitable order, then each new quantity can over-write an old one, and no extra storage is needed. The number of operations required by the nonprogressive primal and dual algorithms is by construction exactly the same. It is easily verified that this number is

$$\tfrac{1}{2}n(n+1)\cdot(3A + 2M),$$

where A stands for one addition or one subtraction and M for one multiplication or one division.

In the progressive versions, the storage of two extra $(n+1)$-vectors is needed. The required number of operations, when the points $\alpha_0, \alpha_1, \cdots, \alpha_n$ are introduced one at a time, is for the

$$\text{primal algorithm:} \quad \tfrac{1}{2}n(n+1)(3A+4M),$$
$$\text{dual algorithm:} \quad \tfrac{1}{2}n(n+1)(4A+3M).$$

This is only slightly more than for the nonprogressive versions and compares fairly well with the number of operations required in [4] (see [1, p. 300]).

If the points α_i are symmetrically situated around zero, then the amount of work can be halved by first applying the preliminary transformation given in [4]. This transforms by simple row and column operations the Vandermonde matrix $V = V(-\alpha_n, \cdots, -\alpha_1, \alpha_1, \cdots, \alpha_n)$ to a 2×2 block-diagonal form

$$V = \begin{pmatrix} V' & 0 \\ 0 & V' \end{pmatrix}, \qquad V' = V(\alpha_1^2, \cdots, \alpha_n^2).$$

This means that two problems, each with only half the number of points, have to be solved.

We note that it is possible to compute V^{-1} by applying the primal or dual algorithm to the unit matrix. This will, however, be an n^3-process. Since in [5] Traub has given an algorithm which computes V^{-1} in $(1/2)n(7n-9)M$ and $(5/2)n(n-1)A$, this is clearly inefficient.

Note also that even when V^{-1} is explicitly known it takes $n^2(M+A)$ to compute $\mathbf{x} = V^{-1}\mathbf{b}$ in the ordinary way. This is only slightly fewer operations than for our nonprogressive algorithms. Since also n^2 memory locations are needed to store V^{-1}, it may often be better not to handle it explicitly.

4. Confluent Vandermonde Systems. Let us replace, in the Vandermonde matrix $V(\alpha_0, \alpha_0+\epsilon, \alpha_2, \cdots, \alpha_n)$, the second column by the difference of the second and first column, divided by ϵ. In the limit, when $\epsilon \to 0$, we obtain the confluent Vandermonde matrix

$$V(\alpha_0, 2; \alpha_2, \cdots, \alpha_n) = \begin{bmatrix} 1 & 0 & 1 & \cdots & 1 \\ \alpha_0 & 1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_0^2 & 2\alpha_0 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \alpha_0^n & n\alpha_0^{n-1} & \alpha_2^n & \cdots & \alpha_n^n \end{bmatrix}.$$

In the same way, in the Vandermonde matrix $V(\alpha_0, \cdots, \alpha_{p-1}, \alpha_p, \cdots, \alpha_n)$, where $\alpha_j = \alpha_0 + j\epsilon$, $j = 0, 1, \cdots, p-1$, we can replace column $(j+1)$ by ϵ^{-j} times the jth order difference of the first $(j+1)$ columns. In the limit when $\epsilon \to 0$ we get the Vandermonde matrix $V(\alpha_0, p; \alpha_p, \cdots, \alpha_n)$, where the order of confluency at α_0 is $(p-1)$.

In the general case, the order of confluency is $(\gamma_i - 1)$ at the $(m+1)$ points β_j, $j = 0, 1, \cdots, m$, and we denote the corresponding Vandermonde matrix by

$$V_n = V(\beta_0, \gamma_0; \beta_1, \gamma_1; \cdots; \beta_m, \gamma_m). \tag{18}$$

This matrix has $(n + 1)$ columns, where

$$n = \left(\sum_{i=0}^{m} \gamma_i\right) - 1,$$

and the elements in the γ_i columns corresponding to β_i are given by

$$v_{i,n_i+k} = \left(\frac{d^k(\beta^i)}{d\beta^k}\right)_{\beta=\beta_i}, \qquad n_i = \sum_{i=0}^{j-1} \gamma_i, \qquad k = 0, 1, \cdots, \gamma_i - 1. \tag{19}$$

We now want to find out rules on how to modify our algorithms for solving (1) and (3) in the confluent case. To do this we again start from the dual system (3), where V now is the matrix (18). The polynomial $P(z)$ defined by (4) now solves the interpolation problem

$$P^{(k)}(\beta_i) = f_i^{(k)}, \qquad k = 0, 1, \cdots, \gamma_i - 1, \qquad j = 0, 1, \cdots, m,$$

where the right-hand side of (3) has been denoted by

$$\mathbf{f} = (f_0, \cdots, f_0^{(\gamma_0-1)}, \quad f_1, \cdots, f_1^{(\gamma_1-1)}, \cdots, f_m, \cdots, f_m^{(\gamma_m-1)})^T.$$

It is well known that Newton's method of interpolation can be generalized to the confluent case. An excellent survey is given in an appendix in [6]. If we let $\alpha_{n_j+k} = \beta_j, k = 0, 1, \cdots, \gamma_j - 1, j = 0, 1, \cdots, m$, then the fundamental polynomials $Q_k(z)$ in (4) are unchanged and only the divided differences have to be generalized. As long as a divided difference has at least two different arguments, (6) can be used to reduce the order. When a divided difference with all arguments equal is reached this is defined by

$$f[\beta_i, k + 1] = f[\underbrace{\beta_i, \cdots, \beta_i}_{(k+1 \text{ times})}] = f_i^{(k)}/k!.$$

From this we deduce the important rule: if the maximum order of confluency is

$$q = \max_{0 \le i \le m} \gamma_i - 1,$$

then in the dual algorithm only Step (i) for $k = 0, 1, \cdots, q - 1$, (in the primal algorithm only Step (ii) for $k = q - 1, \cdots, 1, 0$) is modified. To simplify the discussion we restrict ourselves in the following to the case when only one point β_i is confluent i.e., $\gamma_j = 1$ if $j \neq i$, $\gamma_i = q + 1$. We then have $V^{-T} = L^T U^T$, where the factorization of L^T is given by (13) but that of U^T is modified to

$$U^T = D_{n-1}^{-1} M_{n-1} \cdots D_q^{-1} M_q (D'_{q-1})^{-1} M'_{q-1} \cdots (D'_0)^{-1} M'_0.$$

Here the matrix M'_k is equal to M_k except in the rows $i + k, \cdots, i + q$. These $(q - k + 1)$ rows are now of the form

$$\begin{array}{ccccccccccc}
0 & \cdots & 0 & 1 & \cdots & \cdots & \cdots & \cdots & 0 & \longleftarrow & i + k \\
 & & & 0 & & 1 & & & & & \\
 & & & & \ddots & & \ddots & & & & \\
0 & \cdots & -1 & \cdots & \cdots & 0 & & 1 \quad \cdots & 0 & \longleftarrow & i + q. \\
 & & & i + k & & & & & & &
\end{array}$$

The matrix D_k' is changed in the rows $i + k, \cdots, i + q - 1$, where the diagonal entries now should be $(k + 1)$. Thus, in the dual algorithm, (9) is modified for $k = 0, 1, \cdots, q - 1$ so that

$$c_j^{(k+1)} = (c_j^{(k)} - c_{j-1-q+k}^{(k)})/(\alpha_j - \alpha_{j-k-1}), \qquad j = i + q + 1$$
$$= c_j^{(k)}/(k + 1), \qquad j = i + q, i + q - 1, \cdots, i + k + 1.$$

For all other values of j the formula (9) can still be applied. The changes in the primal algorithm can easily be deduced from this.

The general case can be treated by superposing changes from single points of confluency. The corresponding formulas are, however, rather awkward to write down, and there seems to be no point in doing this. Many important special cases can, however, easily be treated, e.g., that with only the two endpoints of confluency greater than one, or that with all points of the same order of confluency. For some more results on this subject, see [9].

5. Test Results. To test the accuracy of the given algorithms, a few test examples have been run on an IBM 360/50. The programs used were FORTRAN-versions of the Algol-procedures 'pvand' and 'dvand' given in the appendix. They were run in double precision, which corresponds to 14 hexadecimal digits in the mantissa, or a unit of precision equal to $u = 16^{-13} = 2.22 \times 10^{-16}$.

For the primal algorithm, test systems were chosen with $\alpha_i = 1/(i + 3)$, $b_i = 1/2^i$, $i = 0, 1, \cdots, n$. The exact solution can be shown to be

$$x_i = (-1)^i \binom{n + 1}{i + 1} \left(1 + \frac{i + 1}{2}\right)^n.$$

Let x_i be the computed solution and take as a measure of the relative error

$$e_n = \max_{0 \le i \le n} |x_i - \bar{x}_i|/|x_i|.$$

The results from runs with $n + 1 = 5(5)30$ are summarized in the following table:

$n + 1$	5	10	15	20	25	30
e_n/u	4	5	10	54	81	280

The same systems were also solved with the algorithm described in [1]. The solution then deteriorated completely after $n = 15$. In fact, considering the ill-conditioned nature of the test systems, the observed errors for the new algorithm are surprisingly small.

It seems as if at least some problems connected with Vandermonde systems, which traditionally have been considered too ill-conditioned to be attacked, actually can be solved with good precision.

For the dual algorithm, test systems were solved for $n + 1 = 5(5)30$ with $\alpha_i = 1/(i + 2)$, $f_i = T_n(\alpha_i)$, $i = 0, 1 \cdots, n$, where $T_n(x)$ is the Chebyshev polynomial of order n. Thus, the problem is to retrieve the coefficients of $T_n(x)$ from function values at the points α_i. Here, however, the solutions from the dual algorithm deteriorated completely after $n = 10$. The systems were also solved with a dual version of the algorithm in [1]. The errors now showed the same behaviour, and were only slightly larger than for the new algorithm.

Appendix. Below we give Algol procedures for the derived algorithms. The direct versions are named "pvand" and "dvand", the progressive versions "pvandprg" and "dvandprg".

```
procedure pvand(n,alpha,x,b);
value n; integer n; array alpha,x,b;
comment The procedure pvand solves the system of equations Vx =b,
where V is the non-confluent Vandermonde matrix V(alpha[0],...,
alpha[n]). The right hand side b is left unchanged, unless the
formal parameters x and b correspond to the same actual parameter;
begin integer j,k;
    for k:= 0 step 1 until n do x[k] := b[k];
    for k:= 0 step 1 until n-1 do
    for j:= n step -1 until k+1 do
    x[j]:= x[j] - alpha[k] x x[j-1];
    for k:= n-1 step -1 until 0 do
    begin for j:= k+1 step 1 until n do
        x[j] := x[j]/(alpha[j] - alpha[j-k-1]);
        for j:= k step 1 until n-1 do
        x[j] := x[j] - x[j+1]
    end
end

procedure dvand(n,alpha,a,f);
value n; integer n; array alpha,a,f;
comment The procedure dvand solves the system of equations V^T a = f,
where V is the non-confluent Vandermonde matrix V(alpha[0],...
alpha[n]). The right hand side f is left unchanged, unless the
formal parameters a and f correspond to the same actual parameter;
begin integer j,k;
    for k:= 0 step 1 until n do a[k] := f[k];
    for k:= 0 step 1 until n-1 do
    for j:=n step -1 until k+1 do
    a[j] :=(a[j] - a[j-1])/(alpha[j] - alpha[j-k-1]);
    for k:= n-1 step -1 until 0 do
    for j:= k step 1 until n-1 do
    a[j] := a[j] - alpha[k] x a[j+1]
end;
```

```
procedure pvandprg(n,alpha,d,u,x,b);
value n; integer n; array alpha,d,u,x,b;
comment The procedure pvandprg updates the solution to the system
of equations Vx = b, where V is the non-confluent Vandermonde matrix
V(alpha[0],...,alpha[n-1]), when the value alpha[n] is added. The
procedure must be called successively with n = 0,1,2,... The content
of the arrays d,u[0:nmax], which are used as working storage, must
not be changed between calls;
begin integer j; real delta,dn;
    d[n] := b[n];
    for j:= n-1 step -1 until 0 do
    d[j]:= d[j+1] - alpha[n-j-1] x d[j];
    dn := d[0]; u[n] := 1;
    for j:= 0 step 1 until n-1 do
    begin delta := alpha[n] - alpha[j];
        u[j] := u[j] x delta;u[n] := u[n] x delta;
        x[j] := x[j] + dn/u[j]
    end; x[n] := dn/u[n]
end;
procedure dvandprg(n,alpha,c,m,a,f);
value n; integer n; array alpha,c,m,a,f;
comment The procedure dvandprg updates the solution to the system
of equations V^T a = f, where V is the non-confluent Vandermonde matrix
V(alpha[0],...,alpha[n-1]), when the value alpha[n] is added. The
procedure must be called successively with n = 0,1,2,... The content
of the arrays c,m[0:nmax], which are used as working storage, must
not be changed between calls;
begin integer j; real cn;
    c[n] := f[n];
    for j:= n-1 step -1 until 0 do
    c[j] :=(c[j+1] - c[j])/(alpha[n] - alpha[j]);
    m[n] := if n=0 then 1 else 0; cn := a[n] := c[0];
    for j:= n step -1 until 1 do
    begin m[j] := m[j] - alpha[n-1] x m[j-1];
        a[n-j] := a[n-j] + m[j] x cn
    end
end;
```

Acknowledgment. We are indebted to Professor Gene H. Golub of Stanford University for stimulating conversation on this subject and for making the two authors aware of each other.

We also thank Professor Gustavo Galimberti of the Universidad Central de Venezuela for his help in the preliminary testing of the algorithms.

Department of Applied Mathematics
University of Linköping
S-581 83 Linköping, Sweden

Departamento de Computacion
Universidad Central
Caracas, Venezuela 105

1. C. BALLESTER & V. PEREYRA, "On the construction of discrete approximations to linear differential expressions," *Math. Comp.*, v. 21, 1967, pp. 297–302. MR 37 #3751.

2. W. GAUTSCHI, "On the inverses of Vandermonde and confluent Vandermonde matrices. I, II," *Numer. Math.*, v. 4, 1962, pp. 117–123; ibid., v. 5, 1963, pp. 425–430. MR 25 #3059; MR 29 #1734.

3. J. W. JEROME & L. L. SCHUMAKER, *A Note on Obtaining Natural Spline Functions by the Abstract Approach of Laurent*, MRC Technical Report #776, University of Wisconsin, Madison, Wis., 1967.

4. J. N. LYNESS & C. B. MOLER, "Van der Monde systems and numerical differentiation," *Numer. Math.*, v. 8, 1966, pp. 458–464. MR 34 #956.

5. J. F. TRAUB, "Associated polynomials and uniform methods for the solution of linear problems," *SIAM Rev.*, v. 8, 1966, pp. 277–301. MR 34 #7054.

6. J. F. TRAUB, *Iterative Methods for the Solution of Equations*, Prentice-Hall Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, N. J., 1964. MR 29 #6607.

7. G. GALIMBERTI & V. PEREYRA, "Numerical differentiation and the solution of multidimensional Vandermonde systems," Pub. 69–07, Dep. de Comp., U. Central de Venezuela, Caracas, 1969; *Math. Comp.*, v. 24, 1970, pp. 357–364.

8. G. GALIMBERTI & V. PEREYRA, *Solving Confluent Vandermonde Systems of Hermite Type*, Pub. 70–02, Dep. de Comp., U. Central de Venezuela, Caracas, 1970.

9. S.-Å. GUSTAFSON, *Rapid Computation of Interpolation Formulae and Mechanical Quadrature Rules*, Technical Report CS #70–152, Stanford University, Stanford, Calif., 1970.