Fast solution of confluent Vandermonde-like linear systems using polynomial arithmetic

H.-J. Fischer

Abstract. In the present paper we propose a method for the fast solution of some confluent Vandermonde-like systems by polynomial arithmetic. The results (though less general than known results of Hao Lu) can be formulated explicitly in terms of numerator polynomials. In order to avoid numerical difficulties with the canonical (monomial) base, algorithms for multiplication and division in arbitrary orthogonal bases are developed. We discuss an application to the numerical computation of Gauss-rules from given modified moments.

1. Introduction

Let σ be a given positive measure with infinite support $S(\sigma)$ and finite moments of all orders. Then there exists a unique family of monic polynomials π_j with

$$(1) \qquad \int \pi_l(x)\pi_j(x)\,d\sigma(x) = 0 \quad \text{for } l < j \text{ and } \quad \int \pi_j^2(x)\,d\sigma(x) = \gamma_j^{-2} > 0$$

for $j = 0, 1, \ldots$ (γ_j is the leading coefficient of the *orthonormal* polynomial of degree j). They satisfy a three-term recurrence relation

$$(2) \qquad \pi_{j+1}(x) = (x - \alpha_j)\pi_j(x) - \beta_j \pi_{j-1}(x) \quad \text{for} \quad j = 0, 1, \ldots,$$

if we set $\pi_{-1}(x) \equiv 0$.

From the standard theory of orthogonal polynomials we know that there is an n-point quadrature formula with nodes $\{\tau_i\}_{i=1,\ldots,n}$ and weights $\{\sigma_i\}_{i=1,\ldots,n}$ integrating exactly all polynomials of degree $\le 2n-1$:

$\{\tau_i\}_{i=1,\ldots,n}$ are the zeros of π_n and the weights σ_i are defined by

$$\sigma_i^{-1} = \gamma_{n-1}^2 \pi_{n-1}(\tau_i)\pi_n'(\tau_i) \quad \text{for} \quad i = 1, \ldots, n$$

(for the last equation, compare Exercise 6.2 and Equation (4.10) in Chapter I of Chihara's book [Chi78]).

If we are given modified moments $m_j = \int p_j(x)\,d\sigma(x)$ for some appropriate set of polynomials $\{p_j\}$, then our orthogonal polynomials and thus the recurrence coefficients α_j and β_j are well defined in terms of them. Usually, the polynomials p_j are classical orthogonal polynomials with known recurrence coefficients. Then, the recurrence coefficients α_j and β_j can be computed with the well-known method of modified moments. In order to study the numerical condition of this algorithm, Gautschi considered the map from modified moments m_j to $\{\sigma_i\}, \{\tau_i\}$ instead (see [Gau78], [Gau82] and [Gau86]). In this analysis, he used known results on inverses of confluent Vandermonde-like matrices. The (well-conditioned) map from $\{\alpha_j\}, \{\beta_j\}$ to $\{\sigma_i\}, \{\tau_i\}$ is an eigenvalue problem for the tridiagonal Jacobi matrix. These problems can be solved very efficiently, see [Gau90], [Gau96] and [Gau94].

However, we can try to calculate directly the nodes $(\tau_i)_{i=1,\dots,n}$ and the weights $(\sigma_i)_{i=1,\dots,n}$ of a Gaussian quadrature formula for the measure σ from our modified moments m_j. Since the quadrature formula is exact for polynomials of degree $2n-1$, we obtain the nonlinear system of equations

$$\sum_{i=1}^{n} \sigma_i \, p_j(\tau_i) = m_j, \qquad j = 0, \dots, 2n-1.$$

This can be written as

$$F(\sigma, \tau) = m$$

with $\sigma = [\sigma_1, \dots, \sigma_n]^T$, $\tau = [\tau_1, \dots, \tau_n]^T$, $m = [m_0, \dots, m_{2n-1}]^T$, $p = [p_0, \dots, p_{2n-1}]^T$ and the vector-valued function

$$F(\sigma, \tau) = \sum_{i=1}^{n} \sigma_i \, p(\tau_i).$$

The method of Newton leads to an iteration

$$\mathbf{P}_c \mathbf{D} \begin{bmatrix} \sigma^{(k+1)} - \sigma^{(k)} \\ \tau^{(k+1)} - \tau^{(k)} \end{bmatrix} = m - F(\sigma^{(k)}, \tau^{(k)}),$$

where $\mathbf{P}_c = \mathbf{P}_c(\tau^{(k)})$ will be defined in the next section and

$$\mathbf{D} = \mathbf{D}(\sigma^{(k)}) = \operatorname{diag}(1, \dots, 1, \sigma_1^{(k)}, \dots, \sigma_n^{(k)}).$$

The calculation of the residual vector can be avoided. We have

$$F(\sigma, \tau) = \mathbf{P}_c(\tau) \begin{bmatrix} \sigma \\ \mathbf{0} \end{bmatrix} = \mathbf{P}_c(\tau) \mathbf{D}(\sigma) \begin{bmatrix} \sigma \\ \mathbf{0} \end{bmatrix},$$

and this implies that we have to solve the system

$$\mathbf{P}_c \mathbf{D} \begin{bmatrix} \sigma^{(k+1)} \\ \tau^{(k+1)} - \tau^{(k)} \end{bmatrix} = m$$

in each iteration.

The solution of linear systems with a general $n \times n$ matrix, of course, requires $O(n^3)$ arithmetic operations. Fortunately, there are known fast algorithms for this special structure of matrices with only $O(n^2)$ operations, if the polynomials p_j satisfy a three-term recurrence relation, see the bibliography in [Lu94]. In this 1994 paper of Hao Lu an even faster algorithm is proposed for confluent Vandermonde matrices, which is based on fast polynomial arithmetic and requires only $O(n \log^2 n)$ arithmetic operations. Our results, Theorem 1 and Theorem 2, were inspired by this article. In the meantime, Hao Lu generalized his results considerably in the article [Lu96] (of which the author became aware only in February 1998, after finishing the investigations reported here). In fact, Hao Lu is able to handle the case of polynomials satisfying a k-term recurrence relation. Thus, our result is much weaker — it is just the identification of what Hao Lu calls "J-match" in terms of numerator polynomials in the special case $k = 3$. Some drawbacks and advantages of our formulation will be discussed below. Moreover, we handle here only the special case of confluent Vandermonde-like matrices (n double nodes) necessary for our iterative algorithm above. The general case could be formulated in the same way as in [Lu94] and [Lu96].

2. Vandermonde-like matrices and their inversion

Many problems in interpolation or construction of quadrature formulas require the solution of linear systems with (possibly confluent) Vandermonde-like matrices or their transposes. If we are given a set of polynomials $\{p_j\}_{j=0,\dots,n-1}$, where p_j has degree j, and a set of distinct nodes $\{\tau_i\}_{i=1,\dots,n}$, then the $n \times n$ Vandermonde-like matrix is defined by

$$\mathbf{P} = \begin{bmatrix} p_0(\tau_1) & p_0(\tau_2) & \cdots & p_0(\tau_n) \\ p_1(\tau_1) & p_1(\tau_2) & \cdots & p_1(\tau_n) \\ \vdots & \vdots & & \vdots \\ p_{n-1}(\tau_1) & p_{n-1}(\tau_2) & \cdots & p_{n-1}(\tau_n) \end{bmatrix}. \tag{3}$$

We consider here only the simplest case of a $2n \times 2n$ confluent Vandermonde-like matrix related to the set of polynomials $\{p_j\}_{j=0,\dots,2n-1}$ and the set of nodes $\{\tau_i\}_{i=1,\dots,n}$ defined by

$$\mathbf{P}_c = \begin{bmatrix} p_0(\tau_1) & \cdots & p_0(\tau_n) & p_0'(\tau_1) & \cdots & p_0'(\tau_n) \\ p_1(\tau_1) & \cdots & p_1(\tau_n) & p_1'(\tau_1) & \cdots & p_1'(\tau_n) \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{2n-1}(\tau_1) & \cdots & p_{2n-1}(\tau_n) & p_{2n-1}'(\tau_1) & \cdots & p_{2n-1}'(\tau_n) \end{bmatrix}. \tag{4}$$

Results on the structure of the inverses of (confluent) Vandermonde-like matrices have been known for some time. Usually, they are formulated in terms of interpolation polynomials at the nodes τ_i. If we write interpolation polynomials in Newton's form, we obtain algorithms due to Higham, see his papers [Hig88] and [Hig90]. Kailath and Olshevsky use another approach in their paper [KO97], where many bibliographical references are given.

3. The method of Hao Lu

In this section, we assume that the polynomials p_j are orthogonal with respect to some measure s. There are several normalizations due to historical reasons, so we do not expect them to be monic or orthonormal. Instead, we assume the leading coefficient of p_j to be K_j. The polynomials satisfy a three-term recurrence

$$xp_j(x) = \frac{K_j}{K_{j+1}} p_{j+1}(x) + a_j p_i(x) + \frac{K_j}{K_{j-1}} b_j p_{j-1}(x) \quad \text{for} \quad j = 0, 1, \ldots, \tag{5}$$

where $p_{-1} \equiv 0$ and $p_0 \equiv K_0$. Their orthogonality relation is

$$\int p_j(x)\, p_k(x)\, ds(x) = \frac{K_j^2}{c_j^2}\, \delta_{kj} = H_j\, \delta_{kj}, \tag{6}$$

where the leading coefficients c_j of *orthonormal* polynomials can be found from

$$c_j^{-2} = c_0^{-2} \prod_{k=1}^{j} b_k$$

($c_0^{-2} = \int 1\, ds(x)$ is the full weight of the measure). It is well known that the polynomials

$$q_j(x) = \int \frac{p_j(x) - p_j(t)}{x - t}\, ds(t) \tag{7}$$

for $j > 0$ satisfy the same recurrence (5) with the initial values $q_0 \equiv 0$ and $q_1 \equiv c_0^{-2} K_1$ (and this means that the degree of q_j is $j - 1$ — the polynomials are called *associated polynomials*, *polynomials of second kind* or *numerator polynomials*). We can use them to formulate the following identity:

Lemma 1. *Let p_j be the orthogonal polynomials with respect to some measure s satisfying Equation* (5) *and the orthogonality conditions* (6)*, and let the numerator polynomials be defined by* (7)*. Then we have*

$$\frac{p_n(x) - p_n(\tau)}{x - \tau} = \sum_{j=0}^{n-1} \frac{1}{H_j} \left[p_j(x)\, q_n(x) - p_n(x)\, q_j(x)\right] p_j(\tau). \tag{8}$$

Proof: We follow closely the proof of [NS91, Chapter 2, Lemma 7.4]. The expression

$$\frac{p_n(x) - p_n(\tau)}{x - \tau}$$

is a polynomial of degree $n - 1$ in τ and thus it can be written as

$$\frac{p_n(x) - p_n(\tau)}{x - \tau} = \sum_{j=0}^{n-1} d_j\, p_j(\tau).$$

Multiplying both sides by $p_j(\tau)$ and integrating we obtain from (6)

$$\int \frac{p_n(x) - p_n(\tau)}{x - \tau}\, p_j(\tau)\, ds(\tau) = H_j\, d_j.$$

The proposition now follows from the identity

$$\begin{aligned}\frac{p_n(x)-p_n(\tau)}{x-\tau}\,p_j(\tau) &= \frac{p_n(x)-p_n(\tau)}{x-\tau}\,p_j(x) - \frac{p_j(x)-p_j(\tau)}{x-\tau}\,p_n(x) + \\ &+ \frac{p_j(x)-p_j(\tau)}{x-\tau}\,p_n(\tau),\end{aligned}$$

since

$$\int \frac{p_n(x)-p_n(\tau)}{x-\tau}\,p_j(x)\,ds(\tau) = q_n(x)\,p_j(x),$$

$$\int \frac{p_j(x)-p_j(\tau)}{x-\tau}\,p_n(x)\,ds(\tau) = q_j(x)\,p_n(x)$$

and

$$\int \frac{p_j(x)-p_j(\tau)}{x-\tau}\,p_n(\tau)\,ds(\tau) = 0$$

due to orthogonality (observe that $\frac{p_j(x)-p_j(\tau)}{x-\tau}$ is a polynomial of degree $j-1<n$ in τ). □

This auxiliary result enables us to formulate the method of Hao Lu for arbitrary orthogonal polynomials. First, we introduce some notations.

Definition 1. *Let q and r be polynomials. There are unique polynomials q and t with $\deg(t) < \deg(r)$ and $q(x) = r(x)\,s(x) + t(x)$, and we will write $s = q \div r$ and $t = q \bmod r$.*

Now we are in a position to formulate a variant of Theorem 1.1 of [Lu94].

Theorem 1. *The solution $\mathbf{y} = [y_1, \ldots, y_n]^T$ of the Vandermonde-like system*

$$\mathbf{P}\mathbf{y} = \mathbf{r}$$

with $\mathbf{r} = [r_0, \ldots, r_{n-1}]^T$ is given by

$$y_i = \frac{g(\tau_i)}{\omega_n'(\tau_i)},$$

where

$$\omega_n(x) = (x-\tau_1)\ldots(x-\tau_n), \quad g(x) = \omega_n(x) r(x) \div p_n(x)$$

and

$$r(x) = q_n(x)\sum_{j=0}^{n-1}\frac{r_j}{H_j}\,p_j(x) \bmod p_n(x).$$

Proof: We multiply the equation

$$\sum_{i=1}^{n} y_i p_j(\tau_i) = r_j$$

by $\dfrac{1}{H_j}[p_j(x)\, q_n(x) - p_n(x)\, q_j(x)]$ and sum over j. This (in view of the identity (8)) gives the equation

$$\begin{aligned}\sum_{i=1}^{n} y_i\, \frac{p_n(x)-p_n(\tau_i)}{x-\tau_i} &= \sum_{j=0}^{n-1} \frac{r_j}{H_j}\,[p_j(x)\, q_n(x) - p_n(x)\, q_j(x)] \\ &= q_n(x) \sum_{j=0}^{n-1} \frac{r_j}{H_j}\, p_j(x) - p_n(x) \sum_{j=0}^{n-1} \frac{r_j}{H_j}\, q_j(x).\end{aligned}$$

Since the left-hand side (and thus the right-hand side) is a polynomial of degree $\leq n-1$, it coincides with $r(x)$. If we multiply this equation by $\omega_n(x)$, we have

$$p_n(x) \sum_{i=1}^{n} y_i\, \frac{\omega_n(x)}{x-\tau_i} - \sum_{i=1}^{n} y_i\, p_n(\tau_i) \frac{\omega_n(x)}{x-\tau_i} = \omega_n(x) r(x).$$

Now the polynomial $\displaystyle\sum_{i=1}^{n} y_i\, p_n(\tau_i) \frac{\omega_n(x)}{x-\tau_i}$ has degree $\leq n-1$, consequently

$$\sum_{i=1}^{n} y_i\, \frac{\omega_n(x)}{x-\tau_i} = \omega_n(x) r(x) \div p_n(x) = g(x).$$

Substituting $x = \tau_i$, we obtain $y_i\, \omega_n'(\tau_i) = g(\tau_i)$, and this proves our theorem. □

Now we formulate the result for the confluent case.

Theorem 2. *The solution*

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = [y_1, \ldots, y_n, z_1, \ldots, z_n]^T$$

of the confluent Vandermonde-like system

$$\mathbf{P}_c \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \mathbf{r}$$

with $\mathbf{r} = [r_0, \ldots, r_{2n-1}]^T$ *is given by*

$$y_i = \frac{g'(\tau_i)}{\omega_n'^2(\tau_i)} - 2\, \frac{\omega_n''(\tau_i)\, g(\tau_i)}{\omega_n'^3(\tau_i)} \quad \textit{and} \quad z_i = \frac{g(\tau_i)}{\omega_n'^2(\tau_i)},$$

where

$$\omega_n(x) = (x-\tau_1)\ldots(x-\tau_n), \quad g(x) = \omega_n^2(x) r(x) \div p_{2n}(x)$$

and

$$r(x) = q_{2n}(x) \sum_{j=0}^{2n-1} \frac{r_j}{H_j}\, p_j(x) \bmod p_{2n}(x).$$

Proof: From the identity (8) with $2n$ instead of n we obtain by differentiating with respect to τ

$$\frac{p_{2n}(x) - p_{2n}(\tau)}{(x-\tau)^2} - \frac{p'_{2n}(\tau)}{x-\tau} = \sum_{j=0}^{2n-1} \frac{1}{H_j}\left[p_j(x)\, q_{2n}(x) - p_{2n}(x)\, q_j(x)\right] p'_j(\tau).$$

Together with the original identity we obtain

$$\sum_{i=1}^{n} \left[y_i \, \frac{p_{2n}(x) - p_{2n}(\tau_i)}{x-\tau_i} + z_i \left\{ \frac{p_{2n}(x) - p_{2n}(\tau_i)}{(x-\tau_i)^2} - \frac{p'_{2n}(\tau_i)}{x-\tau_i} \right\} \right] = r(x)$$

as above. Multiplying by $\omega_n^2(x)$ we have

$$p_{2n}(x) \sum_{i=1}^{n} \left[y_i \, \frac{\omega_n^2(x)}{x-\tau_i} + z_i \, \frac{\omega_n^2(x)}{(x-\tau_i)^2} \right] + R_{2n-1}(x) = r(x)\, \omega_n^2(x),$$

where

$$R_{2n-1}(x) = -\sum_{i=1}^{n} \left[(y_i\, p_{2n}(\tau_i) + z_i\, p'_{2n}(\tau_i)) \, \frac{\omega_n^2(x)}{x-\tau_i} + z_i\, p_{2n}(\tau_i) \, \frac{\omega_n^2(x)}{(x-\tau_i)^2} \right]$$

is a polynomial of degree $\le 2n-1$. This gives the equation

$$\sum_{i=1}^{n} \left[y_i \, \frac{\omega_n^2(x)}{x-\tau_i} + z_i \, \frac{\omega_n^2(x)}{(x-\tau_i)^2} \right] = r(x)\, \omega_n^2(x) \div p_{2n}(x) = g(x).$$

Now substituting $x = \tau_i$, we obtain

$$z_i\, \omega_n'^2(\tau_i) = g(\tau_i),$$

and differentiating with respect to x and substituting $x = \tau_i$, we have

$$y_i\, \omega_n'^2(\tau_i) + 2 z_i\, \omega''_n(\tau_i)\, \omega'_n(\tau_i) = g'(\tau_i).$$

Our proposition follows immediately from these two equations. □

We return here to the general results in [Lu96]. As we mentioned above, they are valid for polynomials p_j satisfying some k-term recurrence relation. The polynomial r of our theorems is obtained there in terms of a J-match without a polynomial multiplication and division (see Theorems 3.2 and 3.3 in [Lu96]). For numerical computations, however, it has to be converted into the polynomial base used for the polynomial arithmetic. This needs some effort, too, since the J-match is defined by another recurrence (see Theorem 2.3 in [Lu96]. Hao Lu proposed the use of the canonical base $\{x^j\}$ for computations, allowing fast multiplications/divisions by FFT. We suspect that this could lead to serious numerical difficulties in some cases. If the p_j are Chebyshev polynomials of first kind, the J-match consists of Chebyshev polynomials of second kind. The coefficients of these polynomials in the canonical base are prohibitively large. The sum of their absolute values is of order $(1+\sqrt{2})^j$ for U_j (see Lemma 2(ii) in our paper [Fis96]). Already for $n = 1000$ this number is too large to be represented as a floating-point number on many computers. Moreover, the error of the final result will be influenced by the *absolute* error of these coefficients due to cancelation of leading digits. We only mention here that this exponential growth of the relevant condition number was found already in Gautschi's paper [Gau78] (compare his Equation (3.10) for $\omega = 1$).

For these reasons we feel that the arithmetic operations with polynomials should be performed in the more natural base of the p_j themselves. Algorithms for the arithmetic operations are developed in Section 5 below. As we will see, the algorithms use only the recurrence coefficients of the base. Then, the identification of the J-match in terms of the numerator polynomials is an advantage, because they satisfy essentially the same recurrence relation.

4. Some numerical experiments

In this section, we report the results of some numerical experiments with the iterative method mentioned in the Introduction. More precisely, we considered a variant, namely the system

$$\mathbf{P}_c\mathbf{D}(\sigma^{(k+1)}) \begin{bmatrix} \sigma^{(k+1)} \\ \tau^{(k+1)} - \tau^{(k)} \end{bmatrix} = m.$$

We assume that we are given Chebyshev moments, i.e., $p_k = T_k$. This makes our algorithm very efficient: Up to a numerical factor, the U_n are the second kind polynomials for T_n. Division by T_n is trivial in view of the identities

$$2T_n(x)T_m(x) = T_{n+m}(x) + T_{|n-m|}(x)$$

and

$$2U_n(x)T_m(x) = \begin{cases} U_{n+m}(x) + U_{n-m}(x) & , n \geq m \\ U_{2m-1} & , n = m-1 \\ U_{n+m}(x) + U_{m-n-2}(x), & n \leq m-2 \end{cases}$$

and the product of two polynomials of degree n can be calculated by fast DCT (discrete cosine transform) in $O(n \log n)$ arithmetical operations. This is interpolation at the zeros of Chebyshev polynomials, and it was also used for the computation of the polynomial ω_n in Theorem 2.

As starting values $\tau^{(0)}$ we used the zeros of T_n, $\sigma^{(0)}$ is not needed. We did not even try to find conditions for convergence. The measure σ in our experiments was the measure of orthogonality of Jacobi polynomials in the following three cases:

1. $\alpha = \beta = 0$ — Legendre polynomials,
2. $\alpha = \beta = \frac{1}{2}$ — Chebyshev polynomials of second kind,
3. $\alpha = -\beta = -\frac{1}{2}$ — Chebyshev polynomials of third kind.

As a defect criterion, we used the largest (in absolute value) component of the computed correction $\tau^{(k+1)} - \tau^{(k)}$. The results are displayed in the Tables 1–3.

Surprisingly, in the first case our iteration converged for all values of n we tried, up to $n = 500$. However, to obtain a defect of the order of the machine precision, we needed 7 iterations. The CPU-time required was 7.09 seconds, whereas

TABLE 1. Defect after k iterations, 1. case, $n = 500$

k	defect
1	4.22E-04
2	1.44E-04
3	2.87E-05
4	4.67E-06
5	5.94E-08
6	4.95E-12
7	1.65E-16

TABLE 2. Defect after k iterations, 2. case, $n = 51$

k	defect
1	8.65E-03
2	6.03E-03
3	5.06E-03
4	2.73E-03
5	9.90E-04
6	1.29E-04
7	2.32E-06
8	5.62E-10
9	1.76E-15

TABLE 3. Defect after k iterations, 3. case, $n = 25$

k	defect
1	3.33E-02
2	2.29E-02
3	1.78E-02
4	6.55E-03
5	1.61E-03
6	9.11E-05
7	2.40E-07
8	1.14E-12
9	1.17E-15

the traditional method (solving the eigenvalue problem for the Jacobi matrix) required only 0.88 seconds. The zeros of Chebyshev polynomials of second and third kinds, of course, are not so near to the starting values. The values of n displayed in these cases were the largest for which the iteration converged at all.

We see that our direct method is far inferior to traditional methods, even for very large n. However, the explicit formulas for the solution of (confluent)

Vandermonde-like systems are interesting in their own. Moreover, the resulting iterative procedure is interesting from a theoretical point of view, since it shows that time complexity can be reduced to $O(n\log^2 n)$.

The implementation of the algorithm in the case of Chebyshev moments uses fast polynomial arithmetic (the algorithms for fast discrete cosine transforms are taken from the paper of Steidl and Tasche [ST91]). This makes our algorithm faster than that of Gohberg and Olshevsky in [GO94], though their method is based essentially on the same identity (8).

The idea of using nested polynomial division (and FFT) to speed up the evaluation of a polynomial in n given points (this gives the operation count of $O(n\log^2 n)$ in [Lu94]) was used already in the paper of Finck, Heinig and Rost [FHR93] in an algorithm for Cauchy-Vandermonde systems (with the same operation count). We add the remark that the evaluation of a polynomial in the Chebyshev base at n given points can be speeded up (asymptotically) by another method based on interpolation and FFT, see the article of Dutt and Rokhlin [DR93].

5. The arithmetic of orthogonal polynomials

To implement our algorithms in the general case, we must have algorithms for arithmetical operations with polynomials. Of course, we assume that any polynomial is given as a linear combination of our orthogonal polynomials p_j. Thus, the polynomials are encoded as vectors, and it is clear how to add or subtract or multiply by a scalar factor. Multiplication by a polynomial is far more complicated. In the general case, the products $p_n\, p_m$ cannot be written easily as linear combinations of p_j. Even if the linearization coefficients are known, this approach would require $O(n^3)$ operations for the computation of the product of two polynomials of degree n.

Before we explain how to multiply or divide by polynomials, we look at the problem of evaluation in some point t. The starting point is the recurrence relation (5). Consequently, in this section we assume that our polynomials are *monic* (i.e. $K_j \equiv 1$) — this assumption saves many multiplications. Now for the evaluation of the polynomial

$$q(x) = \sum_{j=0}^{n} q_j p_j(x)$$

at $x = t$ we have Clenshaw's algorithm (see Algorithm 1).

Since we assumed $K_j \equiv 1$, the recurrence (5) can be written in the form

$$xp_j(x) = p_{j+1}(x) + a_j p_i(x) + b_j p_{j-1}(x)$$

for $j = 0, 1, \ldots$. As we can see easily, this equation allows to multiply a polynomial by the independent variable x. Obviously, the corresponding transformation of the coefficient vector is just a multiplication by a tridiagonal matrix. We will denote

$u_n \leftarrow q_n, \quad u_{n+1} \leftarrow 0,$
for $j = n-1$ downto 0 **do**
  $u_j \leftarrow (t - a_j) \times u_{j+1} - b_{j+1} \times u_{j+2} + q_j$
end for
$q(t) \leftarrow u_0$

Algorithm 1: Calculate $q(t)$

this transformation by Ψ (for a polynomial q of degree n, $\Psi(q)$ can be computed in $O(n)$ operations).

Thus, we can use Clenshaw's algorithm not only with some numerical value t, but with the independent variable x, assuming the quantities u_j to be polynomials. This seems to be a rather strange idea, since we just reconstruct the polynomial $q(x)$. Consequently, this is an implementation of the identity map in $O(n^2)$ arithmetical operations. However, this can be very practical!

First, we can perform the operations of the algorithm in *another* orthogonal base satisfying another recurrence of type (5). This gives a conversion algorithm from one orthogonal base to another in $O(n^2)$ arithmetical operations.

Second, we can assume that instead of the numerical values q_j we have the polynomials $q_j r(x)$, where $r(x)$ is some given polynomial. Then the algorithm obviously calculates

$$\sum_{j=0}^{n} q_j r(x) p_j(x) = q(x) r(x).$$

Again, we formulate this as Algorithm 2 (where we simply write $c \times u$ for the multiplication of vector u by a number c). If the degree of $r(x)$ is m, then the

$u_n \leftarrow q_n \times r, \quad u_{n+1} \leftarrow 0,$ $\{u_j$ and r are polynomials$\}$
for $j = n-1$ downto 0 **do**
  $u_j \leftarrow \Psi(u_{j+1}) - a_j \times u_{j+1} - b_{j+1} \times u_{j+2} + q_j \times r$ $\{\Psi$ multiplies by $x\}$
end for
$s \leftarrow u_0$ {the result is a polynomial}

Algorithm 2: Calculate $s(x) = q(x)r(x)$

operation count of our algorithm obviously is $O((m + n/2)n)$. Here the order of factors is crucial — we should guarantee $n \leq m$ for efficiency (if at all possible, since of course $r(x)$ and thus the result could be represented in another orthogonal base).

Interestingly, the computation of $q \div r$ and $q \bmod r$ can be performed along the same lines. In fact, any polynomial u_j in the polynomial version of Clenshaw's Algorithm 1 can be witten uniquely as

$$u_j = v_j r + w_j \quad \text{with} \quad \deg(w_j) < \deg(r).$$

For $\deg(u_j) < \deg(r)$ we have $v_j \equiv 0$ and $w_j = u_j$, this is true for the initial values u_{n+1} and u_n. Obviously, addition, subtraction and multiplication by a constant of the representations (v_j, w_j) of u_j can be performed componentwise. The multiplication by the variable x is not difficult, if we first multiply the "lower" component w_j by x and then reduce the product mod r to obtain the "carry" for the higher part v_j. Since $\deg(\Psi(w_j)) = \deg(w_j) + 1 \leq \deg(r)$, this carry is always a number d_j. Now, a careful analysis of the operations we perform with the polynomials v_j shows that this is exactly the symbolic version of Clenshaw's algorithm. Thus, it reconstructs the polynomial from its coefficients, and this means that the numbers d_j are the coefficients of $q \div r$. These observations give an elegant algorithm for polynomial division in the base $\{p_j\}$:

$w_n \leftarrow q_n, \quad w_{n+1} \leftarrow 0$, $\{w_j$ are polynomials$\}$
for $j = n-1$ downto 0 **do**
  $w_j \leftarrow \Psi(w_{j+1}) - a_j \times w_{j+1} - b_{j+1} \times w_{j+2} + q_j$ $\{\Psi$ multiplies by $x\}$
  $d_j \leftarrow p^*_m(w_j)/p^*_m(r)$ {quotient of highest coefficients, $m = \deg(r)$}
  $w_j \leftarrow w_j - d_j \times r$
end for
$s \leftarrow [d_0, \ldots, d_{n-m}]$
$t \leftarrow w_0$ {the result is a polynomial}

Algorithm 3: Calculate $s = q \div r$ and $t = q \bmod r$

6. Another application of the method of Hao Lu

Some problems in weighted L^2-approximation require the solution of linear systems with Gram matrices. The $n \times n$ Gram matrix is defined by

$$\mathbf{G} = [h_{ij}]_{i,j=1}^n \text{ with } h_{ij} = \int p_{i-1}(x) p_{j-1}(x)\, d\sigma(x),$$

where σ is some measure. Of course, all integrals have to be finite. Furthermore, we assume that the support $S(\sigma)$ of our measure contains at least n points. The monic orthogonal polynomials π_j with respect to σ satisfy the three-term recurrence relation (2).

The Gaussian quadrature formula for the measure σ gives a factorization of $\mathbf{G}$ (we omit the elementary proof):

Lemma 2. *With the notations defined before and* $\mathbf{D} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ *we have the representation*

$$\mathbf{G} = \mathbf{P}\mathbf{D}\mathbf{P}^T.$$

Now we can formulate our result for Gram systems.

Theorem 3. *Let* $\mathbf{y} = [y_0, \ldots, y_{n-1}]^T$ *be the solution of the linear system*

$$\mathbf{G}\mathbf{y} = \mathbf{r},$$

where $\mathbf{G}$ *is defined as above and* $\mathbf{r} = [r_0, \ldots, r_{n-1}]^T$*. Then the polynomial*

$$y(x) = \sum_{j=0}^{n-1} y_j p_j(x)$$

can be computed from

$$y(x) = \gamma_{n-1}^2 g(x)\, \pi_{n-1}(x) \bmod \pi_n(x),$$

with

$$g(x) = r(x)\, \pi_n(x) \div p_n(x)$$

and

$$r(x) = q_n(x) \sum_{j=0}^{n-1} \frac{r_j}{H_j}\, p_j(x) \bmod p_n(x).$$

Proof: The solution $\mathbf{g} = [g_1, \ldots, g_n]^T$ of $\mathbf{P}\mathbf{g} = \mathbf{r}$ is given by

$$g_i = \frac{g(\tau_i)}{\pi_n'(\tau_i)}$$

(since we have $\omega_n(x) = \pi_n(x)$), and multiplying by $\mathbf{D}^{-1}$, using the formula for σ_i^{-1} in the Introduction, we obtain

$$\mathbf{P}^T\mathbf{y} = [\gamma_{n-1}^2 \pi_{n-1}(\tau_1) g(\tau_1), \ldots, \gamma_{n-1}^2 \pi_{n-1}(\tau_n) g(\tau_n)]^T.$$

But we have

$$\mathbf{P}^T\mathbf{y} = [y(\tau_1), \ldots, y(\tau_n)]^T$$

by the definition of the polynomial y, and if the polynomial y of degree $\leq n-1$ coincides with $\gamma_{n-1}^2 \pi_{n-1}(x) g(x)$ in all zeros of π_n, then this implies

$$y(x) = \gamma_{n-1}^2 \pi_{n-1}(x) g(x) \bmod \pi_n(x). \quad \square$$

This result was inspired by an article of Gemignani [Gem97], where he re-invents a known method (see [HHR89]).

References

[Chi78] T. S. Chihara. *An Introduction to Orthogonal Polynomials.* Gordon & Breach, New York, 1978.

[DR93] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data. *SIAM J. Sci. Comput.*, 14:1368–1393, 1993.

[FHR93] Tilo Finck, Georg Heinig, and Karla Rost. An inversion formula and fast algorithms for Cauchy-Vandermonde matrices. *Linear Algebra Appl.*, 183:179–191, 1993.

[Fis96] Hans-Jürgen Fischer. On the condition of orthogonal polynomials via modified moments. *Zeitschrift für Analysis und ihre Anwendungen*, 15(1):223–244, 1996.

[Gau78] Walter Gautschi. Questions of numerical condition related to polynomials. In C. de Boor and G. H. Golub, editors, *Recent Advances in Numerical Analysis*, pages 45–72. Academic Press, New York, 1978.

[Gau82] Walter Gautschi. On generating orthogonal polynomials. *SIAM J. Sci. Statist. Comput.*, 3(3):289–317, 1982.

[Gau86] Walter Gautschi. On the sensitivity of orthogonal polynomials to perturbations in the moments. *Numer. Math.*, 48:369–382, 1986.

[Gau90] Walter Gautschi. Computational aspects of orthogonal polynomials. In Paul Nevai, editor, *Orthogonal Polynomials, Theory and Practice*, volume 294 of *NATO Adv. Sci. Inst. Ser. C: Math. Phys. Sci.*, pages 181–216. Kluwer Acad. Publ., Dordrecht, Boston, London, 1990.

[Gau94] Walter Gautschi. Algorithm 726: ORTHPOL — a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules. *ACM Trans. Math. Software*, 20:21–62, 1994.

[Gau96] Walter Gautschi. Orthogonal polynomials: applications and computation. In *Acta Numerica 1996*, pages 45–119. Cambridge University Press, Cambridge, 1996.

[Gem97] Luca Gemignani. A fast algorithm for generalized Hankel matrices arising in finite-moment problems. *Linear Algebra Appl.*, 267:41–52, 1997.

[GO94] I. Gohberg and V. Olshevsky. Fast inversion of Chebyshev-Vandermonde matrices. *Numer. Math.*, 67:71–92, 1994.

[HHR89] Georg Heinig, Wolfram Hoppe, and Karla Rost. Structured matrices in interpolation and approximation problems. *Wiss. Z. Tech. Univ. Karl-Marx-Stadt*, 31:196–202, 1989.

[Hig88] Nicholas J. Higham. Fast solution of Vandermonde-like systems involving orthogonal polynomials. *IMA J. Numer. Anal.*, 8:473–486, 1988.

[Hig90] Nicholas J. Higham. Stability analysis of algorithms for solving confluent Vandermonde-like systems. *SIAM J. Matrix Anal. Appl.*, 11:23–41, 1990.

[KO97] Thomas Kailath and Vadim Olshevsky. Displacement-structure approach to polynomial Vandermonde and related matrices. *Linear Algebra Appl.*, 261:49–90, 1997.

[Lu94] Hao Lu. Fast solution of confluent Vandermonde linear systems. *SIAM J. Matrix Anal. Appl.*, 15:1277–1289, 1994.

[Lu96] Hao Lu. Solution of Vandermonde-like systems and confluent Vandermonde-like systems. *SIAM J. Matrix Anal. Appl.*, 17:127–138, 1996.

[NS91] E. M. Nikishin and V. N. Sorokin. *Rational Approximation and Orthogonality*. Translations of Mathematical Monographs, 92, Amer. Math. Soc., Providence, RI, 1991.

[ST91] G. Steidl and M. Tasche. A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms. *Math. Comp.*, 56(193):281–296, 1991.

Technische Universität Chemnitz, Fakultät für Mathematik, D-09107 Chemnitz, Germany
E-mail address: hfischer@mathematik.tu-chemnitz.de