

Fast Algorithms for Discrete Polynomial Transforms

Daniel Potts, Gabriele Steidl, and Manfred Tasche

Dedicated to Professor Gerhard Maeß on the occasion of his 60th birthday

Abstract. Consider the Vandermonde-like matrix $\mathbf{P} := (P_k(\cos \frac{j\pi}{N}))_{j,k=0}^N$, where the polynomials P_k satisfy a three-term recurrence relation. If P_k are the Chebyshev polynomials T_k , then \mathbf{P} coincides with $\mathbf{C}_{N+1} := (\cos \frac{jk\pi}{N})_{j,k=0}^N$. This paper presents a new fast algorithm for the computation of the matrix-vector product $\mathbf{P}\mathbf{a}$ in $O(N \log^2 N)$ arithmetical operations. The algorithm divides into a fast transform which replaces $\mathbf{P}\mathbf{a}$ with $\mathbf{C}_{N+1}\tilde{\mathbf{a}}$ and a subsequent fast cosine transform. The first and central part of the algorithm is realized by straightforward cascade summation based on properties of associated polynomials and by fast polynomial multiplications. Numerical tests demonstrate that our fast polynomial transform realizes $\mathbf{P}\mathbf{a}$ with almost the same precision as the Clenshaw algorithm, but is much faster for $N \geq 128$.

1991 *Mathematics Subject Classification.* Primary 65T99, 42C10, 33C25.

Key words and phrases. Discrete polynomial transform, Vandermonde-like matrix, fast cosine transform, fast polynomial transform, Chebyshev knots, cascade summation.

1 Introduction

Let w be a non-negative, integrable weight function with

$$\int_{-1}^1 w(x) dx > 0$$

and let $L_w^2[-1, 1]$ denote the real Hilbert space with inner product

$$\langle f, g \rangle := \int_{-1}^1 w(x) f(x) g(x) dx \quad (f, g \in L_w^2[-1, 1])$$

and norm $\|\cdot\|$. As example we consider the weight functions

$$w(x) := (1 - x^2)^{\lambda-1/2} \quad (\lambda > -1/2; x \in (-1, 1)). \quad (1.1)$$

Let $\{P_n\}_{n \in \mathbb{N}_0}$ be a sequence of orthogonal polynomials $P_n \in \Pi_n$ with respect to $\langle \cdot, \cdot \rangle$. Here Π_n denotes the set of polynomials of degree $\leq n$. Then every $P \in \Pi_N$ can be represented as

$$P = \sum_{k=0}^N \frac{\langle P, P_k \rangle}{\|P_k\|^2} P_k, \quad (1.2)$$

where $\langle P, P_k \rangle$ can be computed by a convenient quadrature rule, for example by

$$\langle P, P_k \rangle = \sum_{j=0}^{2N} w_j^{2N} P(c_j^{2N}) P_k(c_j^{2N}) \quad (1.3)$$

with the weights

$$w_j^{2N} := \int_{-1}^1 w(x) \frac{L(x)}{L'(c_j^{2N})(x - c_j^{2N})} dx, \quad L(x) := \prod_{j=0}^{2N} (x - c_j^{2N})$$

and with the *Chebyshev nodes*

$$c_j^N := \cos \frac{j\pi}{N} \quad (j = 0, \dots, N).$$

For $w := 1$, i.e. for Legendre polynomials P_k , the quadrature rule (1.3) coincides with the *Clenshaw–Curtis quadrature* with positive weights

$$w_j^{2N} := \frac{1}{N} \varepsilon_j^{2N} \sum_{l=0}^N \varepsilon_l^N \frac{-2}{4l^2 - 1} c_{lj}^N \quad (j = 0, \dots, 2N).$$

Here $\varepsilon_0^N = \varepsilon_N^N := \frac{1}{2}$ and $\varepsilon_j^N := 1$ ($j = 1, \dots, N-1$). Similar, but more complicated expressions of w_j^{2N} can be given for the weight functions (1.1). Notice that such weights can be computed via fast cosine transforms.

Let $M, N \in \mathbb{N}$ with $M \geq N$ be given powers of 2. We are interested in an efficient solution of the following two problems:

1. Given $a_k \in \mathbb{R}$ ($k = 0, \dots, N$) compute the *discrete polynomial transform* $\text{DPT}(N+1, M+1): \mathbb{R}^{N+1} \rightarrow \mathbb{R}^{M+1}$ defined by

$$\hat{a}_j := \sum_{k=0}^N a_k P_k(c_j^M) \quad (j = 0, \dots, M). \quad (1.4)$$

The transform matrix $\mathbf{P} := (P_k(c_j^M))_{j,k=0}^{M,N}$ is called *Vandermonde-like matrix*.

2. Given $b_j \in \mathbb{R}$ ($j = 0, \dots, M$) compute the *transposed discrete polynomial transform* $\text{TDPT}(M+1, N+1): \mathbb{R}^{M+1} \rightarrow \mathbb{R}^{N+1}$ defined by

$$\tilde{b}_k := \sum_{j=0}^M b_j P_k(c_j^M) \quad (k = 0, \dots, N). \quad (1.5)$$

The first problem addresses the evaluation of polynomials $P \in \Pi_N$ given in the form (1.2) at Chebyshev nodes c_j^M . The second problem is concerned with the approximation of the Fourier coefficients of $P \in \Pi_N$ by a quadrature rule. Clearly, by (1.2) and (1.3), the problems (1.4) and (1.5) with $M = 2N$, $a_k = \|P_k\|^{-2} \langle P, P_k \rangle$ and $b_j = w_j^{2N} P(c_j^{2N})$ are “inverse” in the sense that the corresponding transform matrices \mathbf{P} and \mathbf{P}^T satisfy

$$\mathbf{P} \text{diag} (\|P_k\|^{-2})_{k=0}^N \mathbf{P}^T \text{diag} (w_j^{2N})_{j=0}^{2N} = \mathbf{I}_{2N+1},$$

$$\mathbf{P}^T \text{diag} (w_j^{2N})_{j=0}^{2N} \mathbf{P} \text{diag} (\|P_k\|^{-2})_{k=0}^N = \mathbf{I}_{N+1}$$

with the $(N+1, N+1)$ -identity matrix \mathbf{I}_{N+1} .

In general the realization of (1.4) or (1.5) requires $O(NM)$ arithmetical operations, too much for practical purposes with large N . Hence we look for a fast algorithm to solve our problems only with $O(N \log^2 N) + O(M \log M)$ arithmetical operations. A fast algorithm for (1.4) implies the factorization of the transform matrix \mathbf{P} into a product of sparse matrices. Consequently, once a fast algorithm for (1.4) is known, a fast algorithm for the “transposed” problem (1.5) with the transform matrix \mathbf{P}^T is also available by transposing the

sparse matrix product. Therefore, we restrict our attention to the fast computation of (1.4).

There are several papers addressing the above problems (see [7, 6, 9, 11, 12, 15, 16]). If the orthogonal polynomials are the *Chebyshev polynomials of first kind*

$$T_n(x) := \cos(n \arccos x) \quad (x \in [-1, 1]),$$

which are orthogonal with respect to $w(x) := (1 - x^2)^{-1/2}$ ($x \in (-1, 1)$), problem (1.4) can be computed via fast cosine transforms (see [17, 18, 19, 2]) in $O(M \log M)$ arithmetical operations. Hence, a straightforward idea for the fast solution of (1.4) with arbitrary orthogonal polynomials P_n is to realize a basis exchange from $\{P_n\}_{n=0}^N$ to $\{T_n\}_{n=0}^N$ followed by a fast cosine transform. In the case of Legendre polynomials P_n , Alpert and Rokhlin [1] have proposed an $O(N \log 1/\varepsilon)$ basis exchange algorithm based on the approximation of the elements in the basis transform matrix. Here ε denotes the desired precision.

Our direct approach computes the basis exchange with $O(N \log^2 N)$ arithmetical operations by a divide-and-conquer technique combined with fast polynomial multiplications. The algorithm can be designed for arbitrary polynomials P_n satisfying a three-term recurrence relation. It requires multiplications with precomputed values of associated polynomials of P_n occupying $O(N \log N)$ elements of storage. Numerical tests for various orthogonal polynomials, especially ultraspherical polynomials, result in small relative errors between the “exact” solution calculated in high precision arithmetic and the solution obtained by our algorithm in double precision arithmetic. It is interesting that the “transposed” version of our (slightly modified) algorithm for the solution of the “transposed” problem (1.5) can be considered as a modified Driscoll–Healy algorithm [6, 7, 10] in which the original fast Fourier transforms are replaced by fast cosine transforms. In our feeling, the following approach to fast polynomial transforms is simpler and more straightforward than the original Driscoll–Healy algorithm for the problem (1.5).

This paper is organized as follows: Taking into account that our whole polynomial transform algorithm is based on fast realizations of different discrete cosine transforms, Section 2 deals with discrete cosine transforms. Section 3 describes our fast polynomial transform. A modified Driscoll–Healy algorithm and the relation with our algorithm is sketched in Section 4. Numerical results are presented in Section 5. Finally, Section 6 contains some concluding remarks.

2 Discrete cosine transforms

The heart of our fast polynomial transform consists in the fast polynomial multiplication via fast cosine transforms. Let

$$\begin{aligned}\mathbf{C}_{N+1} &:= (c_{jk}^N)_{j,k=0}^N, & \mathbf{D}_{N+1} &:= \text{diag}(\varepsilon_j^N)_{j=0}^N, \\ \tilde{\mathbf{C}}_N &:= (c_{j(2k+1)}^{2N})_{j,k=0}^{N-1}, & \tilde{\mathbf{D}}_N &:= \text{diag}(\varepsilon_j^N)_{j=0}^{N-1}.\end{aligned}$$

Then the following transforms are referred as *discrete cosine transforms* (DCT) of type I – III, respectively:

DCT-I($N + 1$): $\mathbb{R}^{N+1} \rightarrow \mathbb{R}^{N+1}$ with

$$\hat{\mathbf{a}} := \mathbf{C}_{N+1} \mathbf{D}_{N+1} \mathbf{a},$$

$\mathbf{a} := (a_k)_{k=0}^N$, $\hat{\mathbf{a}} := (\hat{a}_j)_{j=0}^N \in \mathbb{R}^{N+1}$, i.e.

$$\hat{a}_j := \sum_{k=0}^N \varepsilon_k^N a_k c_{jk}^N = \sum_{k=0}^N \varepsilon_k^N a_k T_k(c_j^N),$$

DCT-II(N): $\mathbb{R}^N \rightarrow \mathbb{R}^N$ with

$$\hat{\mathbf{b}} := \tilde{\mathbf{C}}_N \mathbf{b},$$

$\mathbf{b} := (b_k)_{k=0}^{N-1}$, $\hat{\mathbf{b}} := (\hat{b}_j)_{j=0}^{N-1} \in \mathbb{R}^N$, i.e.

$$\hat{b}_j := \sum_{k=0}^{N-1} b_k c_{j(2k+1)}^{2N} = \sum_{k=0}^{N-1} b_k T_j(c_{2k+1}^{2N}),$$

DCT-III(N): $\mathbb{R}^N \rightarrow \mathbb{R}^N$ with

$$\hat{\mathbf{b}} := \tilde{\mathbf{C}}_N^\top \tilde{\mathbf{D}}_N \mathbf{b},$$

i.e.

$$\hat{b}_j := \sum_{k=0}^{N-1} \varepsilon_k^N b_k c_{k(2j+1)}^{2N} = \sum_{k=0}^{N-1} \varepsilon_k^N b_k T_k(c_{2j+1}^{2N}).$$

In the following, let $N = 2^t$ ($t \in \mathbb{N}$). There exist various fast algorithms performing the above discrete cosine transforms with $O(N \log N)$ instead of $O(N^2)$ arithmetical operations. For DCT-III and DCT-II we prefer the fast algorithms in [18] because of their low arithmetical complexity and since the corresponding data permutations allow a simple, efficient implementation

(see [13]). Fast algorithms for DCT-I based on [18] can be found in [2] (see also [19]). Concerning the inverse DCT's, it is easy to check (see [2]):

Lemma 2.1. *It holds that*

$$\begin{aligned} \mathbf{C}_{N+1} \mathbf{D}_{N+1} \mathbf{C}_{N+1} \mathbf{D}_{N+1} &= \frac{N}{2} \mathbf{I}_{N+1}, \\ \tilde{\mathbf{C}}_N^T \tilde{\mathbf{D}}_N \tilde{\mathbf{C}}_N &= \tilde{\mathbf{C}}_N \tilde{\mathbf{C}}_N^T \tilde{\mathbf{D}}_N = \frac{N}{2} \mathbf{I}_N. \end{aligned}$$

Hence $(\mathbf{C}_{N+1} \mathbf{D}_{N+1})^{-1} = \frac{2}{N} \mathbf{C}_{N+1} \mathbf{D}_{N+1}$ and $(\tilde{\mathbf{C}}_N)^{-1} = \frac{2}{N} \tilde{\mathbf{C}}_N^T \tilde{\mathbf{D}}_N$ such that the inverse DCT's can be computed by the same fast cosine transforms.

Let $P \in \Pi_n$ ($n \in \mathbb{N}$) be given with respect to the basis of Chebyshev polynomials, i.e.

$$P = \sum_{k=0}^n a_k T_k$$

with known real coefficients a_k . Further, let $Q \in \Pi_m$ ($m \in \mathbb{N}$) be a fixed polynomial with known values $Q(c_{2j+1}^{2M})$ for $j = 0, \dots, M-1$, where $M = 2^s$ ($s \in \mathbb{N}$) with $M/2 \leq m+n < M$ is chosen. Then the Chebyshev coefficients b_k ($k = 0, \dots, m+n$) in

$$R := PQ = \sum_{k=0}^{n+m} b_k T_k$$

can be computed in a fast way by the following procedure:

Algorithm 2.2 (Fast polynomial multiplication)

Input: $M = 2^s$ ($s \in \mathbb{N}$) with $M/2 \leq m+n < M$,

$Q(c_{2j+1}^{2M}) \in \mathbb{R}$ ($j = 0, \dots, M-1$) with $Q \in \Pi_m$,

$a_k \in \mathbb{R}$ ($k = 0, \dots, n$).

1. Compute

$$(P(c_{2j+1}^{2M}))_{j=0}^{M-1} := \tilde{\mathbf{C}}_M^T (a_k)_{k=0}^{M-1}$$

by fast DCT-III(M) of $(a_k)_{k=0}^{M-1}$ with $a_k := 0$ ($k = n+1, \dots, M-1$).

2. Evaluate the M products

$$R(c_{2j+1}^{2M}) := P(c_{2j+1}^{2M}) Q(c_{2j+1}^{2M}) \quad (j = 0, \dots, M-1).$$

3. Compute

$$(b_k)_{k=0}^{M-1} := \frac{2}{M} \tilde{\mathbf{D}}_M \tilde{\mathbf{C}}_M (R(c_{2j+1}^{2M}))_{j=0}^{M-1}$$

by fast DCT-II(M) of $(R(c_{2j+1}^{2M}))_{j=0}^{M-1}$.

Output: b_k ($k = 0, \dots, m+n$).

The fast DCT-III(2^s) computed by [18] requires $2^{s-1} s$ multiplications and $2^{s-1} (3s-2) + 1$ additions. Hence, Algorithm 2.2 realizes the polynomial multiplication of $P \in \Pi_n$ and $Q \in \Pi_m$ with respect to the basis of Chebyshev polynomials in $2^s(s+2) + 2$ multiplications and $2^s(3s-2) + 2$ additions.

Remark 2.3. A similar algorithm for the fast polynomial multiplication can be derived involving DCT-I instead of DCT-II and DCT-III, if the values $Q(c_j^M)$ ($j = 0, \dots, M$) are known (see [2]).

3 Fast polynomial transform

Let $\{P_n\}_{n \in \mathbb{N}_0}$ be a sequence of polynomials defined by the three-term recurrence relation

$$\begin{aligned} P_{-1}(x) &:= 0, & P_0(x) &:= 1, \\ P_n(x) &= (\alpha_n x + \beta_n) P_{n-1}(x) + \gamma_n P_{n-2}(x) \quad (n = 1, 2, \dots) \end{aligned} \quad (3.1)$$

with $\alpha_n, \beta_n, \gamma_n \in \mathbb{R}$ and $\alpha_n > 0, \gamma_n \neq 0$ ($n \in \mathbb{N}$). By Favard's theorem, $\{P_n\}_{n=0}^\infty$ is an orthogonal polynomial sequence with respect to some quasi-definite moment functional (see [4], Theorem 4.4). In particular, we consider the Chebyshev polynomials T_n with

$$\begin{aligned} T_0(x) &:= 1, & T_1(x) &:= x, \\ T_n(x) &= 2x T_{n-1}(x) - T_{n-2}(x) \quad (n = 2, 3, \dots). \end{aligned}$$

Shifting the index n in (3.1) by $c \in \mathbb{N}_0$, we obtain the *associated polynomials* $P_n(\cdot, c)$ of P_n defined by

$$\begin{aligned} P_{-1}(x, c) &:= 0, & P_0(x, c) &:= 1, \\ P_n(x, c) &:= (\alpha_{n+c} x + \beta_{n+c}) P_{n-1}(x, c) + \gamma_{n+c} P_{n-2}(x, c) \quad (n = 1, 2, \dots). \end{aligned}$$

Now induction yields (see [3])

Lemma 3.1 *For $c, n \in \mathbb{N}_0$, it holds*

$$P_{c+n}(x) = P_n(x, c) P_c(x) + \gamma_{c+1} P_{n-1}(x, c+1) P_{c-1}(x).$$

Lemma 3.1 implies

$$\begin{pmatrix} P_{c+n} \\ P_{c+n+1} \end{pmatrix} = \mathbf{U}_n(\cdot, c)^\top \begin{pmatrix} P_{c-1} \\ P_c \end{pmatrix} \quad (3.2)$$

with

$$\mathbf{U}_n(x, c) := \begin{pmatrix} \gamma_{c+1} P_{n-1}(x, c+1) & \gamma_{c+1} P_n(x, c+1) \\ P_n(x, c) & P_{n+1}(x, c) \end{pmatrix}.$$

Let $N = 2^t$ and $M = 2^s$ ($s, t \in \mathbb{N}$; $s \geq t$) be given. Consider

$$P := \sum_{k=0}^N a_k P_k \in \Pi_N$$

with known real coefficients a_k . Our concern is the fast evaluation of $P(c_j^M)$ ($j = 0, \dots, M$) with $O(N \log^2 N) + O(M \log M)$ instead of $O(MN)$ arithmetical operations. The main part of our algorithm realizes the basis exchange from $\{P_k\}_{k=0}^N$ to $\{T_k\}_{k=0}^N$ in Π_N and produces the Chebyshev coefficients \tilde{a}_k in

$$P = \sum_{k=0}^N \tilde{a}_k T_k. \quad (3.3)$$

Knowing these Chebyshev coefficients \tilde{a}_k , the values $P(c_j^M)$ ($j = 0, \dots, M$) can be computed via fast DCT-I ($M+1$) in $O(M \log M)$ arithmetical operations in a final step (see [19, 2])

$$(P(c_j^M))_{j=0}^M = \mathbf{C}_M (\tilde{a}_k)_{k=0}^M, \quad (3.4)$$

where we have to set $\tilde{a}_k := 0$ for $k = N+1, \dots, M$.

Let us turn to the basis exchange. In the *initial step* we use (3.1) and the fact that $T_1(x) = x$ to obtain

$$P = \sum_{k=0}^{N-1} a_k^{(0)} P_k = \sum_{k=0}^{N/4-1} \left(\sum_{l=0}^3 a_{4k+l}^{(0)} P_{4k+l} \right)$$

with

$$\begin{aligned} a_k^{(0)}(x) &:= a_k \quad (k = 0, \dots, N-3), \\ a_{N-2}^{(0)}(x) &:= a_{N-2} + \gamma_{N-1} a_N, \\ a_{N-1}^{(0)}(x) &:= a_{N-1} + \beta_{N-1} a_N + \alpha_{N-1} a_N T_1(x). \end{aligned} \quad (3.5)$$

Now we proceed by cascade summation as shown in Figure 1. By (3.2) with $n = 1$ and $c = 4k + 1$ ($k = 0, \dots, N/4 - 1$) it follows that

$$(a_{4k+2}^{(0)}, a_{4k+3}^{(0)}) \begin{pmatrix} P_{4k+2} \\ P_{4k+3} \end{pmatrix} = (a_{4k+2}^{(0)}, a_{4k+3}^{(0)}) \mathbf{U}_1(\cdot, 4k+1)^\top \begin{pmatrix} P_{4k} \\ P_{4k+1} \end{pmatrix}.$$

Thus

$$P = \sum_{k=0}^{N/4-1} (a_{4k}^{(1)} P_{4k} + a_{4k+1}^{(1)} P_{4k+1})$$

with

$$\begin{pmatrix} a_{4k}^{(1)} \\ a_{4k+1}^{(1)} \end{pmatrix} := \begin{pmatrix} a_{4k}^{(0)} \\ a_{4k+1}^{(0)} \end{pmatrix} + \mathbf{U}_1(\cdot, 4k+1) \begin{pmatrix} a_{4k+2}^{(0)} \\ a_{4k+3}^{(0)} \end{pmatrix}. \quad (3.6)$$

The degree of the polynomial products in (3.6) is at most 3 such that their computation with respect to the Chebyshev polynomials can be realized via Algorithm 2.2 with $M = 4$. Consequently, the evaluation of the Chebyshev coefficients of the polynomials $a_{4k}^{(1)}, a_{4k+1}^{(1)} \in \Pi_3$ ($k = 0, \dots, N/4 - 1$) in *step* 1 requires $11N$ multiplications and $12N$ additions.

We continue in the obvious manner. In *step* τ ($1 < \tau < t$) we compute by (3.2) with $n = 2^\tau - 1$ the Chebyshev coefficients of the polynomials $a_{2^{\tau+1}k}^{(\tau)}, a_{2^{\tau+1}k+1}^{(\tau)} \in \Pi_{2^{\tau+1}-1}$ ($k = 0, \dots, N/2^{\tau+1} - 1$) defined by

$$\begin{pmatrix} a_{2^{\tau+1}k}^{(\tau)} \\ a_{2^{\tau+1}k+1}^{(\tau)} \end{pmatrix} := \begin{pmatrix} a_{2^{\tau+1}k}^{(\tau-1)} \\ a_{2^{\tau+1}k+1}^{(\tau-1)} \end{pmatrix} + \mathbf{U}_{2^\tau-1}(\cdot, 2^{\tau+1}k+1) \begin{pmatrix} a_{2^{\tau+1}k+2^\tau}^{(\tau-1)} \\ a_{2^{\tau+1}k+2^\tau+1}^{(\tau-1)} \end{pmatrix}, \quad (3.7)$$

where we apply Algorithm 2.2 (with $M = 2^{\tau+1}$) for the polynomial products. Assume that the $4N$ values $U_{2^\tau-1}(c_{2^l+1}^{2^\tau+2}, 2^{\tau+1}k+1)$ for $k = 0, \dots, N/2^{\tau+1}$ and $l = 0, \dots, 2^\tau - 1$ were precomputed by Clenshaw algorithm (see [5] or [20], pp. 165 – 172). Then *step* τ requires $(2\tau + 8 + 2^{1-\tau})N$ multiplications and $(6\tau + 5 + 2^{1-\tau})N$ additions and results in

$$P = \sum_{k=0}^{N/2^{\tau+1}-1} (a_{2^{\tau+1}k}^{(\tau)} P_{2^{\tau+1}k} + a_{2^{\tau+1}k+1}^{(\tau)} P_{2^{\tau+1}k+1}).$$

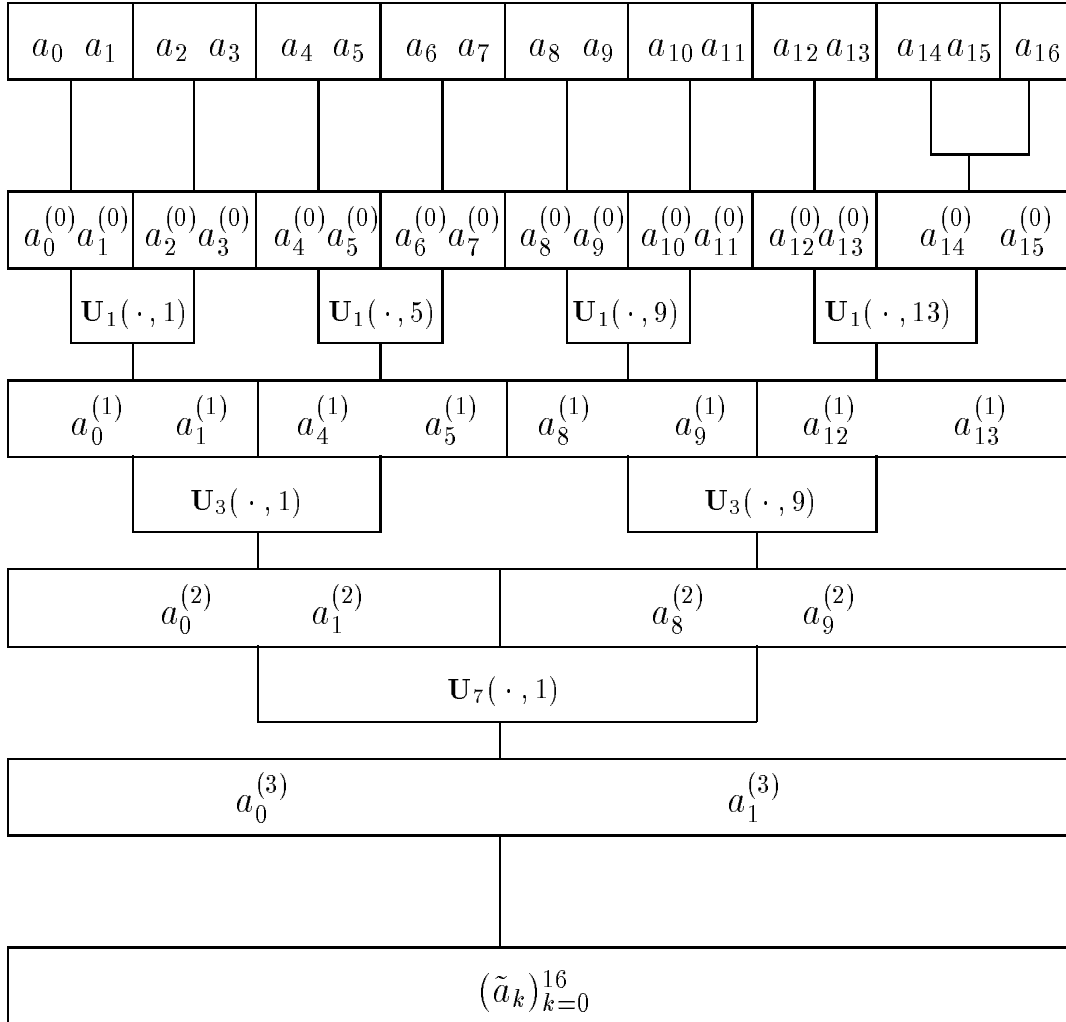


Figure 1. Cascade summation for the computation of the basis exchange in the case $N = 16$

After the *step* $t - 1$, our cascade summation arrives at

$$P = a_0^{(t-1)} P_0 + a_1^{(t-1)} P_1.$$

Now $P_0(x) = 1$, $P_1(x) = \alpha_1 x + \beta_1$ and

$$x T_0(x) = T_1(x), \quad x T_n(x) = \frac{1}{2} (T_{n+1}(x) + T_{n-1}(x)) \quad (n = 1, 2, \dots).$$

Hence, if

$$a_1^{(t-1)} = \sum_{n=0}^{N-1} a_{1,n}^{(t-1)} T_n,$$

then

$$a_1^{(t)} := a_1^{(t-1)} P_1 = \sum_{n=0}^N a_{1,n}^{(t)} T_n$$

with

$$(a_{1,n}^{(t)})_{n=0}^N = (\alpha_1 \mathbf{T}_{N+1}^T + \beta_1 \mathbf{I}_{N+1})(a_{1,n}^{(t-1)})_{n=0}^N, \quad (3.8)$$

where we set $a_{1,N}^{(t-1)} := 0$ and where \mathbf{T}_{N+1} is the tridiagonal $(N + 1, N + 1)$ -matrix

$$\mathbf{T}_{N+1} := \begin{pmatrix} 0 & 1 & & & \\ 1/2 & 0 & 1/2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/2 & 0 & 1/2 \\ & & & 1 & 0 \end{pmatrix}. \quad (3.9)$$

This leads to

$$P = a_0^{(t-1)} + a_1^{(t)}$$

and the final addition of the Chebyshev coefficients of $a_0^{(t-1)}$ and $a_1^{(t)}$ yields the desired Chebyshev coefficients of P , i.e.

$$(\tilde{a}_n)_{n=0}^N = (a_{0,n}^{(t-1)})_{n=0}^N + (a_{1,n}^{(t)})_{n=0}^N. \quad (3.10)$$

We summarize:

Algorithm 3.2 (Fast polynomial transform)

Input: $N = 2^t$, $M = 2^s$ ($s, t \in \mathbb{N}$; $s \geq t$),

$$a_k \in \mathbb{R} \quad (k = 0, \dots, N),$$

$$\mathbf{U}_{2^{\tau-1}}(c_{2^{l+1}}^{2^{\tau+2}}, 2^{\tau+1}k + 1) \quad (\tau = 1, \dots, t-1; k = 0, \dots, 2^{t-\tau-1}; \\ l = 0, \dots, 2^{\tau+1} - 1).$$

Step 0. Compute $a_k^{(0)}(x)$ ($k = 0, \dots, 2^t - 1$) by (3.5).

For $\tau = 1, \dots, t-1$ do

Step τ . For every $k = 0, \dots, 2^{t-\tau-1} - 1$ form (3.7) by Algorithm 2.2 for the fast polynomial multiplications.

Step t . Compute \tilde{a}_n ($n = 0, \dots, N$) by (3.8) and (3.10).

Compute (3.4) by fast DCT-I($M+1$).

Output: $P(c_j^M)$ ($j = 0, \dots, M$).

In summary, we have to store the $4N(\log N - 1)$ precomputed elements of the matrices \mathbf{U} . Counting the arithmetical operations in each step, we verify that the whole basis exchange algorithm requires $N \log^2 N + 7N \log N + O(N)$ multiplications and $3N \log^2 N + 2N \log N + O(N)$ additions.

Finally, the computation of (3.4) by the fast DCT-I($M+1$) takes $\frac{1}{2} M \log M + O(M)$ multiplications and $\frac{3}{2} M \log M + O(M)$ additions. The whole fast polynomial transform becomes more efficient than the Clenshaw algorithm for $N \geq 128$.

A fast algorithm for (1.5), i.e. for the multiplication with \mathbf{P}^T , can be obtained immediately by “reversing” Algorithm 3.2. With other words, we have simply to reverse the direction of the arrows in the flowgraph of Algorithm 3.2. There already exists an $O(N \log^2 N)$ algorithm for the problem (1.4) as well as (1.5) which was originally formulated by Driscoll and Healy [6] with respect to Legendre polynomials and was generalized to arbitrary polynomials satisfying a three-term recurrence relation in [7, 10]. The following section briefly describes the relation between our algorithm and the Driscoll–Healy algorithm.

4 Modified Driscoll–Healy algorithm

In the following, we modify the Driscoll–Healy algorithm by replacing the original fast Fourier transforms by fast cosine transforms which seem to be more natural in the context of the algorithm. As consequence the modified algorithm is simpler, requires fewer arithmetical operations and avoids the arithmetic with complex numbers.

where

$$P := \sum_{j=0}^N \varepsilon_j^N a_j T_j$$

denotes the polynomial associated with \mathbf{P} . Based on the fact that the Chebyshev knots c_k^N ($k = 0, \dots, N$) are the zeros of $(1 - x^2)U_{N-1}(x)$, the relation between circulant matrices related to DCT-I and polynomials is determined by the isomorphism of the algebras

$$\mathcal{P} \simeq \mathbb{R}[x] / ((1 - x^2)U_{N-1}(x)) .$$

Here U_{N-1} is the $(N - 1)$ -th *Chebyshev polynomial of second kind*.

Let the polynomials P_l satisfy the three-term recurrence relation (3.1). Consider (1.5) with $M = N = 2^t$ ($t \in \mathbb{N}$), i.e.

$$\tilde{b}_l := \sum_{j=0}^N \varepsilon_j^N d_j P_l(c_j^N) \quad (l = 0, \dots, N)$$

with given $d_j := (\varepsilon_j^N)^{-1} b_j \in \mathbb{R}$. Following the lines of [6], we define

$$z(k, l) := \sum_{j=0}^N \varepsilon_j^N d_j c_{jk}^N P_l(c_j^N) \quad (k, l = 0, \dots, N)$$

and $\mathbf{z}_l := (z(k, l))_{k=0}^N$. We are interested in the efficient computation of $z(0, l) = \tilde{b}_l$ ($l = 0, \dots, N$). By Lemma 2.1 we observe that

$$\begin{aligned} \mathbf{z}_0 &= \mathbf{C}_{N+1}^{\mathbf{I}} (d_j)_{j=0}^N , \\ \mathbf{z}_l &= \mathbf{C}_{N+1}^{\mathbf{I}} ((P_l(c_j^N))_{j=0}^N \circ (d_j)_{j=0}^N) \end{aligned} \quad (4.2)$$

$$= \frac{2}{N} \mathbf{C}_{N+1}^{\mathbf{I}} P_l(\mathbf{\Gamma}_{N+1}) \mathbf{C}_{N+1}^{\mathbf{I}} \mathbf{z}_0 . \quad (4.3)$$

where \circ denotes the componentwise multiplication. Especially, we obtain with $P_1(x) = \alpha_1 x + \beta_1$ by (4.1) and (4.3) that

$$\begin{aligned} \mathbf{z}_1 &= \frac{2}{N} \mathbf{C}_{N+1}^{\mathbf{I}} (\alpha_1 \mathbf{\Gamma}_{N+1} + \beta_1 \mathbf{I}_{N+1}) \mathbf{C}_{N+1}^{\mathbf{I}} \mathbf{z}_0 \\ &= (\alpha_1 \mathbf{T}_{N+1} + \beta_1 \mathbf{I}_{N+1}) \mathbf{z}_0 \end{aligned}$$

(compare with (3.8)). Using (4.2) and (3.2), we compute $\mathbf{z}_{N/2}$ and $\mathbf{z}_{N/2+1}$ by

$$\begin{pmatrix} \mathbf{z}_{N/2} \\ \mathbf{z}_{N/2+1} \end{pmatrix} = \frac{2}{N} (\mathbf{C}_{N+1}^{\mathbf{I}} \oplus \mathbf{C}_{N+1}^{\mathbf{I}}) \mathbf{U}_{N/2-1} (\mathbf{\Gamma}_{N+1}, 1)^{\mathbf{T}} (\mathbf{C}_{N+1}^{\mathbf{I}} \oplus \mathbf{C}_{N+1}^{\mathbf{I}}) \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \end{pmatrix}$$

and form the truncated vectors

$$\mathbf{z}_l^{(1)} := (z(k, l))_{k=0}^{N/2} \quad (l = 0, 1, N/2, N/2 + 1).$$

This is the result of *step 1* of our modified Driscoll–Healy algorithm. Compare with *step $t - 1$* of Algorithm 3.2. Note that $\mathbf{U}_{N/2-1}(\mathbf{\Gamma}_{N+1}, 1)$ is a block-diagonal $(2N + 2, 2N + 2)$ -matrix.

Now we continue in a similar manner as in [6, 7] but with respect to circulant matrices related to DCT–I. Take into consideration that the description of the following steps involves some more ideas than *step 1*. The resulting modified Driscoll–Healy algorithm does not agree with the “transposed” version of Algorithm 3.2, but it can be considered as a “transposed” version of a modified Algorithm 3.2 in which the fast polynomial multiplications are realized by DCT–I instead of DCT–II and DCT–III as mentioned in Remark 2.3.

However, in our opinion the derivation of the (modified) Driscoll–Healy algorithm is less straightforward than the development of Algorithm 3.2.

5 Numerical tests

Algorithm 3.2 was implemented in C and tested on a Sun SPARCstation 20 for various ultraspherical polynomials. The corresponding fast cosine transforms were described in detail in [18, 2].

Example 5.1. We consider the ultraspherical polynomials P_n^λ ($\lambda > -1/2$) given by

$$\begin{aligned} P_{-1}^\lambda(x) &:= 0, & P_0^\lambda(x) &:= 1, \\ P_n^\lambda(x) &:= \frac{2(n + \lambda - 1)}{n} x P_{n-1}^\lambda(x) - \frac{n + 2\lambda - 2}{n} P_{n-2}^\lambda(x) \quad (n = 1, 2, \dots). \end{aligned}$$

These polynomials are orthogonal with respect to the weight function (1.1). For $\lambda = \frac{n-2}{2}$, the ultraspherical polynomials are the zonal spherical polynomials of S^{n-1} with respect to $\text{SO}(n)/\text{SO}(n-1)$. For the 2-sphere S^2 , i.e. for $\lambda = 1/2$, the ultraspherical polynomials are the Legendre polynomials. For given $a_k \in \mathbb{R}$ ($k = 0, \dots, N$) we compute

$$\hat{a}_j = \sum_{k=0}^N a_k P_k^\lambda(c_j^N) \quad (j = 0, \dots, N) \quad (5.1)$$

by the Clenshaw algorithm (CA) in double precision arithmetic, the Clenshaw algorithm realized in Maple with high precision arithmetic of 64 digits

(CA₆₄) and by our fast polynomial transform (FPT) in double precision arithmetic. Table 1 compares the results for different transform lengths N ranging between 256 and 2048 and various parameters λ . The third column of the table contains the given coefficients a_k , while the fourth and last columns contain the relative error $\varepsilon(\text{CA})$ of the Clenshaw algorithm defined by

$$\varepsilon(\text{CA}) := \max_{0 \leq j \leq N} |\hat{a}_j(\text{CA}) - \hat{a}_j(\text{CA}_{64})| / \max_{0 \leq j \leq N} |\hat{a}_j(\text{CA}_{64})|$$

and the relative error of Algorithm 3.2 given by

$$\varepsilon(\text{FPT}) := \max_{0 \leq j \leq N} |\hat{a}_j(\text{FPT}) - \hat{a}_j(\text{CA}_{64})| / \max_{0 \leq j \leq N} |\hat{a}_j(\text{CA}_{64})|.$$

Here $\hat{a}_j(\text{CA})$, $\hat{a}_j(\text{CA}_{64})$ and $\hat{a}_j(\text{FPT})$ denote the corresponding results of (5.1) using CA, CA₆₄ and FPT, respectively.

N	λ	a_k	$\varepsilon(\text{CA})$	$\varepsilon(\text{FPT})$
256	0.5	$1/(k+1)$	$3.88E-16$	$3.77E-13$
512	0.5	$1/(k+1)$	$1.59E-14$	$5.73E-12$
1024	0.5	$1/(k+1)$	$4.21E-13$	$8.98E-12$
2048	0.5	$1/(k+1)$	$2.11E-12$	$3.19E-11$
256	1.5	$1/(k+1)$	$1.88E-13$	$8.36E-13$
512	1.5	$1/(k+1)$	$6.12E-13$	$1.29E-11$
1024	1.5	$1/(k+1)$	$1.26E-12$	$8.00E-11$
256	5	$1/(k+1)$	$1.15E-13$	$2.72E-13$
512	5	$1/(k+1)$	$5.15E-13$	$4.37E-12$
1024	5	$1/(k+1)$	$1.04E-12$	$5.18E-12$
256	2	1	$2.44E-13$	$7.52E-13$
512	2	1	$8.61E-13$	$6.61E-12$
1024	2	1	$1.71E-12$	$4.82E-12$

Table 1

Note that both Clenshaw algorithm and fast polynomial transform realize the problem (5.1) with almost the same precision, but our method is much faster than the Clenshaw algorithm as shown in Example 5.2.

Example 5.2. As in Example 5.1 we use ultraspherical polynomials. It is well-known that the Clenshaw algorithm requires N^2 multiplications and $3N^2$ additions. The Algorithm 3.2 is significantly faster for large $N \geq 128$. The third and fourth columns of Table 2 list the CPU-times $t(\text{CA})$ and

$t(\text{FPT})$ (in seconds) for the Clenshaw algorithm and for Algorithm 3.2. The last column contains the relative error

$$\tilde{\varepsilon}(\text{FPT}) := \max_{0 \leq j \leq N} |\hat{a}_j(\text{FPT}) - \hat{a}_j(\text{CA})| / \max_{0 \leq j \leq N} |\hat{a}_j(\text{CA})|,$$

Here the original coefficients a_k ($k = 0, \dots, N$) are randomly distributed in the interval $[-0.5, 0.5]$.

N	λ	$t(\text{CA})$	$t(\text{FPT})$	$\tilde{\varepsilon}(\text{FPT})$
128	0.5	0.05	0.04	$3.59E - 14$
256	0.5	0.21	0.07	$4.35E - 12$
512	0.5	0.82	0.19	$4.93E - 12$
1024	0.5	3.27	0.39	$5.78E - 11$
2048	0.5	13.70	0.85	$2.09E - 10$
4096	0.5	55.41	1.92	$1.04E - 09$
8192	0.5	220.05	4.26	$5.04E - 08$
4096	2.5	55.43	1.91	$1.72E - 09$
4096	4.0	55.42	1.91	$6.41E - 10$
4096	5.0	55.42	1.92	$3.35E - 10$

Table 2

6 Conclusions

A motivation to consider the discrete polynomial transforms (1.4) and (1.5) with respect to the $2N + 1$ Chebyshev nodes c_j^{2N} ($j = 0, \dots, 2N$) arises from the Clenshaw–Curtis quadrature which seems to be very useful for the computation of the Fourier coefficients in (1.2) in the case of Legendre polynomials P_n , i.e. $w = 1$. However, other quadrature rules may be of interest. So Gaussian quadrature reduces the number of required nodes in (1.3) from $2N + 1$ to $N + 1$. Hence a natural question is how to compute

$$\sum_{k=0}^N a_k P_k(x_j^M) \quad (j = 0, \dots, M; M \geq N)$$

for arbitrary $x_j^M \in [-1, 1]$ and $a_k \in \mathbb{R}$ in an efficient way. The heart of our method, the basis exchange in Algorithm 3.2 is independent of the choice of knots x_j^M . So it remains to perform the final step of our algorithm, the computation of

$$\sum_{k=0}^N \tilde{a}_k T_k(x_j^M) \quad (j = 0, \dots, M), \quad (6.1)$$

in a fast way. One possibility for realizing (6.1) in $O(M \log^2 M)$ arithmetical operations based on a (heuristic) stabilization of the Borodin–Munro algorithm was suggested in [14]. We prefer the application of the fast adaptive multipole method (see [8]) which computes (6.1) in $O(M \log 1/\varepsilon)$ arithmetical operations, where ε denotes the desired precision. This approach seems to be interesting in connection with fast Fourier transforms on spheres and on distance transitive graphs, too (see [7]). The results will be presented in a forthcoming paper.

Acknowledgement. The authors thank the referee for helpful comments and for pointing out the references [10] and [12].

References

- [1] B.K. Alpert and V. Rokhlin, A fast algorithm for the evaluation of Legendre expansions, *SIAM J. Sci. Statist. Comput.* **12** (1991), 158 – 179.
- [2] G. Baszenski and M. Tasche, Fast polynomial multiplication and convolutions related to the discrete cosine transform, *Linear Algebra Appl.* **252** (1997), 1 – 25.
- [3] S. Belmehdi, On the associated orthogonal polynomials, *J. Comput. Appl. Math.* **32** (1991), 311 – 319.
- [4] T.S. Chihara, *An Introduction to Orthogonal Polynomials*, Gordon and Breach, New York, 1978.
- [5] C.W. Clenshaw, A note on the summation of Chebyshev series, *Math. Comp.* **9** (1955), 118 – 120.
- [6] J.R. Driscoll and D.M. Healy, Computing Fourier transforms and convolutions on the 2–Sphere, *Adv. in Appl. Math.* **15** (1994), 202 – 240.
- [7] J.R. Driscoll, D.M. Healy and D. Rockmore, Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs, *SIAM J. Sci. Comput.*, in print.
- [8] A. Dutt, M. Gu and V. Rokhlin, Fast algorithms for polynomial interpolation, integration and differentiation, Research Report, Yale University, New Haven, CT, 1993.

- [9] W. Gautschi, The condition of Vandermonde-like matrices involving orthogonal polynomials, *Linear Algebra Appl.* **52/53** (1983), 293 – 300.
- [10] D.M. Healy, S. Moore and D. Rockmore, Efficiency and stability issues in the numerical convolution of Fourier transforms and convolutions on the 2-sphere, Technical Report, Dartmouth College, 1994.
- [11] N.J. Higham, Fast solution of Vandermonde-like systems involving orthogonal polynomials, *IMA J. Numer. Anal.* **8** (1988), 473 – 486.
- [12] D. Maslen, A polynomial approach to orthogonal polynomial transforms, Research Report, Max-Planck-Institute of Mathematics, Bonn, 1994.
- [13] S.S.B. Moore, Efficient stabilization methods for fast polynomial transforms, Thesis, Dartmouth College, 1994.
- [14] S.S.B. Moore, D.M. Healy and D.N. Rockmore, Symmetry stabilization for fast discrete monomial transforms and polynomial evaluation, *Linear Algebra Appl.* **192** (1993), 249 – 299.
- [15] S.A. Orszag, Fast eigenfunction transforms, in: *Science and Computers* (G.C. Rota, ed.), Academic Press, New York, 1986, 23 – 30.
- [16] V. Pan, Fast evaluation and interpolation at the Chebyshev sets of points, *Appl. Math. Lett.* **34** (1989), 255 – 258.
- [17] K.R. Rao and P. Yip, *Discrete Cosine Transform*, Academic Press, Boston, 1990.
- [18] G. Steidl, Fast radix- p discrete cosine transform, *Appl. Algebra in Engrg. Comm. Comput.* **3** (1992), 39 – 46.
- [19] G. Steidl and M. Tasche, A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms, *Math. Comp.* **56** (1991), 281 – 296.
- [20] J. Wimp, *Computation with Recurrence Relations*, Pitman Press, Boston, 1984.

Fachbereich Mathematik
 Universität Rostock
 D-18051 Rostock
E-mail address: daniel.potts@stud.uni-rostock.de

Fakultät für Mathematik und Informatik
Universität Mannheim
D-68131 Mannheim
E-mail address: steidl@kiwi.math.uni-mannheim.de

Fachbereich Mathematik
Universität Rostock
D-18051 Rostock
E-mail address: tasche@mathematik.uni-rostock.d400.de