Split-Radix Algorithms for Discrete Trigonometric Transforms

Gerlind Plonka and Manfred Tasche

Abstract

In this paper, we derive new split–radix DCT–algorithms of radix–2 length, which are based on real factorization of the corresponding cosine matrices into products of sparse, orthogonal matrices. These algorithms use only permutations, scaling with $\sqrt{2}$, butterfly operations, and plane rotations/rotation–reflections. They can be seen by analogy with the well–known split–radix FFT. Our new algorithms have a very low arithmetical complexity which compares with the best known fast DCT–algorithms. Further, a detailed analysis of the roundoff errors for the new split–radix DCT–algorithm shows its excellent numerical stability which outperforms the real fast DCT–algorithms based on polynomial arithmetic.

Mathematics Subject Classification 2000. 65T50, 65G50, 15A23.

Key words. Split–radix algorithm, split–radix FFT, discrete trigonometric transform, discrete cosine transform, arithmetical complexity, numerical stability, factorization of cosine matrix, direct sum decomposition, sparse orthogonal matrix factors.

1 Introduction

Discrete trigonometric transforms are widely used in processing and compression of signals and images (see [17]). Examples of such transforms are discrete cosine transforms (DCT) and discrete sine transforms (DST) of types I – IV. These transforms are generated by orthogonal cosine and sine matrices, respectively. Especially, the DCT–II and its inverse DCT–III have been shown to be well applicable for image compression. The roots of these transforms go back to trigonometric approximation in the eighteenth century (see [10]). Discrete trigonometric transforms have also found important applications in numerical Fourier methods, approximation via Chebyshev polynomials, quadrature methods of Clenshaw–Curtis type, numerical solution of partial differential equations (fast Poisson solvers), singular integral equations, and Toeplitz–plus–Hankel systems. Since DCTs are the most widely used transforms, we shall concentrate on the construction of real, fast, and numerically stable DCT–algorithms in this paper.
For large transform lengths n, and even for relatively small lengths $n = 8$ and $n = 16$ used in image compression, one needs to have fast DCT–algorithms. There is a close connection between fast DCT–algorithms and factorization of the transform matrix. Let $C_n \in \mathbb{R}^{n\times n}$ be an orthogonal cosine matrix of large radix–2 order n. Assume that we know a factorization of C_n into a product of sparse matrices

$$C_n = M_n^{(m-1)} \ldots M_n^{(0)}, \qquad 1 < m \ll n. \tag{1.1}$$

Then the DCT–transformed vector $C_n\mathbf{x}$ with arbitrary $\mathbf{x} \in \mathbb{R}^n$ can be computed recursively by

$$\mathbf{x}^{(s+1)} := M_n^{(s)}\mathbf{x}^{(s)}, \qquad (\mathbf{x}^{(0)} := \mathbf{x})$$

for $s = 0, \ldots, m-1$ such that $\mathbf{x}^{(m)} = C_n\mathbf{x}$. Since all matrix factors in (1.1) are sparse, the arithmetical complexity of this method will be low such that the factorization (1.1) of C_n generates a fast DCT–algorithm. For an algebraic approach to fast algorithms of discrete trigonometric transforms we refer to [16].
We may distinguish the following three methods to obtain fast DCT–algorithms:

1. *Fast* DCT–*algorithms via* FFT: It is natural to focus first on the computation of DCT by using the FFT (see [15, 25, 7], [17], pp. 49 – 53, and [24], pp. 229 – 245). Such DCT–algorithms are easy to implement using standard FFT routines and possess a good numerical stability (see [2, 23]). However, the real matrix–vector product $C_n\mathbf{x}$ is computed in complex arithmetic. Therefore the arithmetical complexity of these DCT–algorithms is relatively high. The best FFT–based DCT–algorithms require about $2.5\, n \log_2 n$ flops, where a flop is a real arithmetical operation.

2. *Fast* DCT–*algorithms via polynomial arithmetic*: All components of $C_n\mathbf{x}$ can be interpreted as values of one polynomial at n nodes. Reducing the degree of this polynomial by divide–and–conquer technique, one can get a real fast DCT–algorithm with low arithmetical complexity (see [9, 20, 21, 16]). The best DCT–algorithms require about $2\, n \log_2 n$ flops. A polynomial DCT–algorithm generates a factorization (1.1) of C_n with sparse, *non–orthogonal* matrix factors $M_n^{(s)}$, i.e., the factorization (1.1) does not preserve the orthogonality of C_n (see e.g. [19, 2, 23]). This fact leads to a bad numerical stability of these DCT–algorithms [19, 2, 23].

3. *Fast* DCT–*algorithms via direct matrix factorization*: Using simple properties of trigonometric functions one may find direct factorizations of the transform matrix C_n

into a product of real, sparse matrices. The trigonometric approximation algorithm of Runge [18] can be considered as a first example of this approach. Results on direct matrix factorization of C_n into orthogonal, sparse matrices are due to Chen et al. [4] and Wang [26]. Note that various factorizations of C_n use non–orthogonal matrix factors (see [17], pp. 53 – 62, [3, 12, 13]). Many results were published without proofs. A direct "orthogonal" factorization of the cosine matrix of type II and of order 8 was given by Loeffler et al. [14] and is used even today in JPEG standard (cf. Example 2.8). Improving the earlier results in [4, 26], Schreiber has given a constructive proof of a factorization of some cosine matrices of size 2^n into a product of sparse, orthogonal matrices in [19]. Unfortunately, the construction of this factorization is not simple.
However, another important result in [19] (see also [23]) says that a fast DCT–algorithm possesses an excellent numerical stability, if the algorithm is based on a factorization of C_n into sparse, orthogonal matrices. Therefore, in order to get real, fast, and numerically stable DCT–algorithms, one should be especially interested in a factorization (1.1) with sparse, *orthogonal* matrix factors.

In this paper, we shall derive new split–radix DCT–algorithms of radix–2 length, which are based on real factorization of the corresponding cosine matrix into a product of sparse, orthogonal matrices. Here a matrix factor is said to be sparse if each row and column contains at most 2 nonzero entries. The new DCT –algorithms require only permutations, scaling (with $\sqrt{2}$), butterfly operations, and plane rotations/rotation–reflections with small rotation angles. In contrast with many other factorizations, this split–radix approach admits a *recursive* factorization of a cosine matrix into a product of sparse, *orthogonal* matrices. We shall show that the split–radix DCT–algorithms can be seen by analogy with the well–known split–radix FFT. Our new algorithms have a very low arithmetical complexity which compares with the best known fast DCT–algorithms. In particular, the obtained factorization of the cosine matrix of type II and of order 8 is very similar to that in [14] and needs exactly the same number of multiplications and additions. Algorithms with better arithmetical complexity can only be achieved using scaling matrices which are incorporated into the quantization process afterwards (see e.g. [8]).
Further, a detailed analysis of the roundoff errors for the new split–radix DCT–II algorithm shows its excellent numerical stability which outperforms the real fast DCT–II algorithms based on polynomial arithmetic.

The paper is organized as follows: In Section 2 we introduce different types of cosine and sine matrices and derive factorizations of the cosine matrices of types I – IV. All proofs are based on divide–and–conquer technique applied *directly* to a matrix. That is, a given trigonometric matrix can be represented as a *direct sum* of trigonometric matrices of half size (and maybe of different type). While the factorization in Lemma 2.2 has already been used before (see e.g. [18, 4, 26]), the factorization of the cosine matrix of type IV in Lemma 2.4 is new and one of the basic results of this paper. It enables us to derive the corresponding split–radix DCT–algorithms.
In Section 3 we revisit the split–radix FFT and represent it in a new form which is completely based on matrix factorization of the Fourier matrix.
In Sections 4, 5 and 6 we derive some split–radix algorithms for DCT–II, DCT–IV, and DCT–I. A split–radix DCT–III algorithm follows immediately from a split–radix DCT–II algorithm. A comparison of these algorithms with the split–radix FFT shows the close

connection between these approaches. We also compute the arithmetical complexity of the new algorithms.
Section 7 is devoted to a comprehensive analysis of the numerical stability of a split–radix DCT–II algorithm. It can be shown that the new algorithm has a very low arithmetical complexity and a better numerical stability than the DCT–algorithm based on polynomial arithmetic.

2 Trigonometric matrices

Let $n \geq 2$ be a given integer. In the following, we consider *cosine* and *sine matrices of types* I – IV with order n which are defined by

$$\begin{aligned}
C_{n+1}^{I} &:= \sqrt{\tfrac{2}{n}}\left(\epsilon_n(j)\,\cos\tfrac{jk\pi}{n}\right)_{j,k=0}^{n}, \\
C_{n}^{II} &:= \sqrt{\tfrac{2}{n}}\left(\epsilon_n(j)\,\cos\tfrac{j(2k+1)\pi}{2n}\right)_{j,k=0}^{n-1}, \qquad C_{n}^{III} := (C_{n}^{II})^T, \\
C_{n}^{IV} &:= \sqrt{\tfrac{2}{n}}\left(\cos\tfrac{(2j+1)(2k+1)\pi}{4n}\right)_{j,k=0}^{n-1}, \\
S_{n-1}^{I} &:= \sqrt{\tfrac{2}{n}}\left(\sin\tfrac{(j+1)(k+1)\pi}{n}\right)_{j,k=0}^{n-2}, \\
S_{n}^{II} &:= \sqrt{\tfrac{2}{n}}\left(\epsilon_n(j+1)\,\sin\tfrac{(j+1)(2k+1)\pi}{2n}\right)_{j,k=0}^{n-1}, \qquad S_{n}^{III} := (S_{n}^{II})^T, \\
S_{n}^{IV} &:= \sqrt{\tfrac{2}{n}}\left(\sin\tfrac{(2j+1)(2k+1)\pi}{4n}\right)_{j,k=0}^{n-1}.
\end{aligned} \tag{2.1}$$

Here we set $\epsilon_n(0) = \epsilon_n(n) := \sqrt{2}/2$ and $\epsilon_n(j) := 1$ for $j \in \{1, \ldots, n-1\}$. In our notation a subscript of a matrix denotes the corresponding order, while a superscript signifies the "type" of the matrix. First cosine and sine matrices appeared in connection with trigonometric approximation (see [10, 18]). In signal processing, cosine matrices of type II and III were introduced in [1]. The above classification was given in [26] (cf. [17], pp. 12 – 21).
The cosine and sine matrices of type I – IV are orthogonal (see e.g. [17], pp. 13 – 14, [22, 19]). Strang [22] pointed out that the column vectors of each cosine matrix are eigenvectors of a symmetric second difference matrix and therefore orthogonal.
In the following, I_n denotes the identity matrix and $J_n := (\delta(j+k-n+1))_{j,k=0}^{n-1}$ the counteridentity matrix, where δ means the Kronecker symbol. Blanks in a matrix indicate zeros or blocks of zeros. The direct sum of two matrices A, B is defined to be a block diagonal matrix $A \oplus B := \mathrm{diag}\,(A, B)$.
Let $\Sigma_n := \mathrm{diag}\,((-1)^k)_{k=0}^{n-1}$ be the diagonal sign matrix. In the following lemma we describe the intertwining relations of above cosine and sine matrices.

Lemma 2.1 *The cosine and sine matrices in* (2.1) *satisfy the intertwining relations*

$$\begin{aligned}
C_{n+1}^{I}\,J_{n+1} &= \Sigma_{n+1}\,C_{n+1}^{I}, \qquad S_{n-1}^{I}\,J_{n-1} = \Sigma_{n-1}\,S_{n-1}^{I}, \\
C_{n}^{II}\,J_{n} &= \Sigma_{n}\,C_{n}^{II}, \qquad S_{n}^{II}\,J_{n} = \Sigma_{n}\,S_{n}^{II}, \\
C_{n}^{IV}\,J_{n}\,\Sigma_{n} &= J_{n}\,\Sigma_{n}C_{n}^{IV}, \qquad S_{n}^{IV}\,J_{n}\,\Sigma_{n} = J_{n}\,\Sigma_{n}S_{n}^{IV}
\end{aligned} \tag{2.2}$$

and

$$J_n C_n^{II} = S_n^{II} \Sigma_n, \qquad C_n^{IV} J_n = \Sigma_n S_n^{IV}. \tag{2.3}$$

The proof is straightforward and is omitted here (see [19]). The corresponding properties of C_n^{III} and S_n^{III} follow from (2.2) – (2.3) by transposing.
A *discrete trigonometric transform* is a linear mapping generated by a cosine or sine matrix in (2.1). Especially, the *discrete cosine transform of type* II (DCT–II) of length n is generated by C_n^{II}. The *discrete sine transform of type* I (DST–I) of length $n-1$ is generated by S_{n-1}^{I}.
In this paper, we are interested in fast and numerically stable algorithms for discrete trigonometric transforms. As an immediate consequence of Lemma 2.1, we only need to construct algorithms for DCT–I, DCT–II, DCT–IV, and DST–I.

For even $n \geq 4$, P_n and P_{n+1} denote the *even–odd permutation matrices* (or 2*–stride permutation matrices*) defined by

$$\begin{aligned} P_n \mathbf{x} &:= (x_0, x_2, \ldots, x_{n-2}, x_1, x_3, \ldots, x_{n-1})^T, \qquad \mathbf{x} = (x_j)_{j=0}^{n-1}, \\ P_{n+1} \mathbf{y} &:= (y_0, y_2, \ldots, y_n, y_1, y_3, \ldots, y_{n-1})^T, \qquad \mathbf{y} = (y_j)_{j=0}^{n}. \end{aligned}$$

Note that $P_n^{-1} = P_n^T$ is the n_1–stride permutation matrix and $P_{n+1}^{-1} = P_{n+1}^T$ is the (n_1+1)–stride permutation matrix with $n_1 := n/2$.

Lemma 2.2 *Let $n \geq 4$ be an even integer.*
(i) *The matrix C_n^{II} can be factorized in the form*

$$C_n^{II} = P_n^T \, (C_{n_1}^{II} \oplus C_{n_1}^{IV}) \, T_n(0) \tag{2.4}$$

with the orthogonal matrix

$$T_n(0) := \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & J_{n_1} \\ I_{n_1} & -J_{n_1} \end{pmatrix} = \tfrac{1}{\sqrt{2}} \, (I_{n_1} \oplus J_{n_1}) \begin{pmatrix} I_{n_1} & J_{n_1} \\ J_{n_1} & -I_{n_1} \end{pmatrix}.$$

(ii) *The matrix C_{n+1}^{I} can be factorized in the form*

$$C_{n+1}^{I} = P_{n+1}^T \, (C_{n_1+1}^{I} \oplus C_{n_1}^{III}) \, T_{n+1}(2) \tag{2.5}$$

with the orthogonal matrix

$$T_{n+1}(2) := \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & & J_{n_1} \\ & \sqrt{2} & \\ I_{n_1} & & -J_{n_1} \end{pmatrix}.$$

(iii) *The matrix S_{n-1}^{I} can be factorized in the form*

$$S_{n-1}^{I} = P_{n-1}^T \, (S_{n_1}^{III} \oplus S_{n_1-1}^{I}) \, T_{n-1}(2). \tag{2.6}$$

Proof. We show (2.4) by divide–and–conquer technique. First we permute the rows of C_n^{II} by multiplying with P_n and write the result as a block matrix:

$$P_n \, C_n^{II} = \tfrac{1}{\sqrt{n_1}} \begin{pmatrix} \left(\epsilon_n(2j) \, \cos \frac{2j(2k+1)\pi}{2n}\right)_{j,k=0}^{n_1-1} & \left(\epsilon_n(2j) \, \cos \frac{2j(n+2k+1)\pi}{2n}\right)_{j,k=0}^{n_1-1} \\ \left(\epsilon_n(2j+1) \, \cos \frac{(2j+1)(2k+1)\pi}{2n}\right)_{j,k=0}^{n_1-1} & \left(\epsilon_n(2j+1) \, \cos \frac{(2j+1)(n+2k+1)\pi}{2n}\right)_{j,k=0}^{n_1-1} \end{pmatrix}.$$

By (2.1) and

$$\cos\tfrac{j(n+2k+1)\pi}{n} = \cos\tfrac{j(n-2k-1)\pi}{n}, \quad \cos\tfrac{(2j+1)(n+2k+1)\pi}{2n} = -\cos\tfrac{(2j+1)(n-2k-1)\pi}{2n}$$

it follows immediately that the four blocks of $P_n\, C_n^{II}$ can be represented by $C_{n_1}^{II}$ and $C_{n_1}^{IV}$:

$$\begin{aligned} P_n\, C_n^{II} &= \tfrac{1}{\sqrt{2}} \begin{pmatrix} C_{n_1}^{II} & C_{n_1}^{II} J_{n_1} \\ C_{n_1}^{IV} & -C_{n_1}^{IV} J_{n_1} \end{pmatrix} = (C_{n_1}^{II} \oplus C_{n_1}^{IV}) \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & J_{n_1} \\ I_{n_1} & -J_{n_1} \end{pmatrix} \\ &= (C_{n_1}^{II} \oplus C_{n_1}^{IV})\, T_n(0). \end{aligned}$$

Since $P_n^{-1} = P_n^T$ and $T_n(0)\, T_n(0)^T = I_n$, the matrices P_n and $T_n(0)$ are orthogonal. The proofs of (ii) and (iii) follow similar lines. □

Remark 2.3 *The trigonometric approximation algorithm of Runge [18] is equivalent to (2.5) and (2.6). Similar factorizations of C_n^{II}, C_{n+1}^{I}, and S_{n-1}^{I} were also been presented in [26], but by using of modified even–odd permutation matrices Q_n and Q_{n+1} defined by*

$$\begin{aligned} Q_n \mathbf{x} &:= (x_0, x_2, \ldots, x_{n-2}, x_{n-1}, x_{n-3}, \ldots, x_1)^T, \qquad \mathbf{x} = (x_j)_{j=0}^{n-1}, \\ Q_{n+1} \mathbf{y} &:= (y_0, y_2, \ldots, y_n, y_{n-1}, y_{n-3}, \ldots, y_1)^T, \qquad \mathbf{y} = (y_j)_{j=0}^{n}. \end{aligned}$$

From (2.4) we obtain the following factorization of C_n^{III} :

$$C_n^{III} = T_n(0)^T\, (C_{n_1}^{III} \oplus C_{n_1}^{IV})\, P_n. \tag{2.7}$$

Now we introduce modified identity matrices

$$I_n' := \operatorname{diag}\, (\epsilon_n(j)^{-1})_{j=0}^{n-1}, \qquad I_n'' := \operatorname{diag}\, (\epsilon_n(j+1)^{-1})_{j=0}^{n-1}.$$

Note that I_n' and I_n'' differ from I_n in only one position. The matrix I_n' has the entry $\sqrt{2}$ in the left upper corner and I_n'' has the entry $\sqrt{2}$ in the right lower corner. Let V_n be the forward shift matrix

$$V_n := (\delta(j-k-1))_{j,k=0}^{n-1},$$

i.e., V_n has nonzero entries 1 only in the upper secondary diagonal. Note that for arbitrary $\mathbf{x} = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$ we have

$$V_n \mathbf{x} = (0\,, x_0\,, x_1, \ldots, x_{n-2})^T, \qquad V_n^T \mathbf{x} = (x_1,\, x_2, \ldots, x_{n-1},\, 0)^T.$$

Further we define cosine and sine vectors by

$$\mathbf{c}_n := \left(\cos\tfrac{(2k+1)\pi}{8n}\right)_{k=0}^{n-1}, \qquad \mathbf{s}_n := \left(\sin\tfrac{(2k+1)\pi}{8n}\right)_{k=0}^{n-1}.$$

Lemma 2.4 *For even $n \geq 4$, the matrix C_n^{IV} can be factorized in the form*

$$C_n^{IV} = P_n^T\, A_n(1)\, (C_{n_1}^{II} \oplus C_{n_1}^{II})\, T_n(1) \tag{2.8}$$

with the modified addition matrix

$$A_n(1) := \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_n' & V_{n_1} \Sigma_{n_1} \\ V_{n_1}^T & -I_{n_1}'' \Sigma_{n_1} \end{pmatrix} (I_{n_1} \oplus J_{n_1})$$

and the cross–shaped twiddle matrix

$$T_n(1) := (I_{n_1} \oplus \Sigma_{n_1}) \begin{pmatrix} \operatorname{diag} \mathbf{c}_{n_1} & J_{n_1} \operatorname{diag}\, (J_{n_1} \mathbf{s}_{n_1}) \\ -J_{n_1} \operatorname{diag} \mathbf{s}_{n_1} & \operatorname{diag}\, (J_{n_1} \mathbf{c}_{n_1}) \end{pmatrix}.$$

The two matrices $A_n(1)$ and $T_n(1)$ are orthogonal.

Proof. We show (2.8) again by divide–and–conquer technique. Therefore we permute the rows of C_n^{IV} by multiplying with P_n and write the result as block matrix:

$$P_n\, C_n^{IV} = \frac{1}{\sqrt{n_1}} \begin{pmatrix} \left(\cos \frac{(4j+1)(2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} & \left(\cos \frac{(4j+1)(n+2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} \\ \left(\cos \frac{(4j+3)(2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} & \left(\cos \frac{(4j+3)(n+2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} \end{pmatrix}.$$

Now we consider the single blocks of $P_n\, C_n^{IV}$ and represent every block by $C_{n_1}^{II}$ and $S_{n_1}^{II}$.
1. By (2.2) and

$$\cos \frac{(4j+1)(2k+1)\pi}{4n} = \cos \frac{j(2k+1)\pi}{n} \cos \frac{(2k+1)\pi}{4n} - \sin \frac{j(2k+1)\pi}{n} \sin \frac{(2k+1)\pi}{4n}$$

it follows that

$$\begin{aligned} \frac{1}{\sqrt{n_1}} \left(\cos \frac{(4j+1)(2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} &= \frac{1}{\sqrt{2}} \left(I'_{n_1}\, C_{n_1}^{II}\, \mathrm{diag}\ \mathbf{c}_{n_1} - V_{n_1}\, S_{n_1}^{II}\, \mathrm{diag}\ \mathbf{s}_{n_1}\right) \\ &= \frac{1}{\sqrt{2}} \left(I'_{n_1}\, C_{n_1}^{II}\, \mathrm{diag}\ \mathbf{c}_{n_1} - V_{n_1}\, \Sigma_{n_1}\, S_{n_1}^{II}\, J_{n_1}\, \mathrm{diag}\ \mathbf{s}_{n_1}\right). \quad (2.9) \end{aligned}$$

2. By (2.2) and

$$\cos \frac{(4j+3)(2k+1)\pi}{4n} = \cos \frac{(j+1)(2k+1)\pi}{n} \cos \frac{(2k+1)\pi}{4n} + \sin \frac{(j+1)(2k+1)\pi}{n} \sin \frac{(2k+1)\pi}{4n}$$

we obtain that

$$\begin{aligned} \frac{1}{\sqrt{n_1}} \left(\cos \frac{(4j+3)(2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} &= \frac{1}{\sqrt{2}} \left(V_{n_1}^T\, C_{n_1}^{II}\, \mathrm{diag}\ \mathbf{c}_{n_1} + I''_{n_1}\, S_{n_1}^{II}\, \mathrm{diag}\ \mathbf{s}_{n_1}\right) \\ &= \frac{1}{\sqrt{2}} \left(V_{n_1}^T\, C_{n_1}^{II}\, \mathrm{diag}\ \mathbf{c}_{n_1} + I''_{n_1}\, \Sigma_{n_1}\, S_{n_1}^{II}\, J_{n_1}\, \mathrm{diag}\ \mathbf{s}_{n_1}\right). \quad (2.10) \end{aligned}$$

3. By (2.2) and

$$\begin{aligned} \cos \frac{(4j+1)(n+2k+1)\pi}{4n} &= (-1)^j \cos \left(\frac{j(2k+1)\pi}{n} + \frac{(n+2k+1)\pi}{4n}\right) \\ &= (-1)^j\, \cos \frac{j(2k+1)\pi}{n} \sin \frac{(n-2k-1)\pi}{4n} - (-1)^j\, \sin \frac{j(2k+1)\pi}{n} \cos \frac{(n-2k-1)\pi}{4n} \end{aligned}$$

it follows that

$$\begin{aligned} &\frac{1}{\sqrt{n_1}} \left(\cos \frac{(4j+1)(n+2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} \\ &\quad = \frac{1}{\sqrt{2}} \left(\Sigma_{n_1}\, I'_{n_1}\, C_{n_1}^{II}\, \mathrm{diag}\ (J_{n_1}\, \mathbf{s}_{n_1}) - \Sigma_{n_1}\, V_{n_1}\, S_{n_1}^{II}\, \mathrm{diag}\ (J_{n_1}\, \mathbf{c}_{n_1})\right) \\ &\quad = \frac{1}{\sqrt{2}} \left(I'_{n_1}\, C_{n_1}^{II}\, J_{n_1} \mathrm{diag}\ (J_{n_1}\, \mathbf{s}_{n_1}) + V_{n_1}\, \Sigma_{n_1}\, S_{n_1}^{II}\, \mathrm{diag}\ (J_{n_1}\, \mathbf{c}_{n_1})\right). \quad (2.11) \end{aligned}$$

Here we have used that $\Sigma_{n_1}\, I'_{n_1} = I'_{n_1}\, \Sigma_{n_1}$ and $-\Sigma_{n_1} V_{n_1} = V_{n_1} \Sigma_{n_1}$.
4. By (2.2) and

$$\begin{aligned} \cos \frac{(4j+3)(n+2k+1)\pi}{4n} &= (-1)^{j+1} \cos \left(\frac{(j+1)(2k+1)\pi}{n} - \frac{(n+2k+1)\pi}{4n}\right) \\ &= (-1)^{j+1}\, \cos \frac{(j+1)(2k+1)\pi}{n} \sin \frac{(n-2k-1)\pi}{4n} + (-1)^{j+1}\, \sin \frac{(j+1)(2k+1)\pi}{n} \cos \frac{(n-2k-1)\pi}{4n} \end{aligned}$$

we obtain that

$$\begin{aligned}
&\tfrac{1}{\sqrt{n_1}} \left(\cos \tfrac{(4j+3)(n+2k+1)\pi}{4n}\right)_{j,k=0}^{n_1-1} \\
&\quad = -\tfrac{1}{\sqrt{2}} \left(\Sigma_{n_1} V_{n_1}^T C_{n_1}^{II} \operatorname{diag}(J_{n_1} \mathbf{s}_{n_1}) + \Sigma_{n_1} I''_{n_1} S_{n_1}^{II} \operatorname{diag}(J_{n_1} \mathbf{c}_{n_1})\right) \\
&\quad = \tfrac{1}{\sqrt{2}} \left(V_{n_1}^T \Sigma_{n_1} C_{n_1}^{II} \operatorname{diag}(J_{n_1} \mathbf{s}_{n_1}) - I''_{n_1} \Sigma_{n_1} S_{n_1}^{II} \operatorname{diag}(J_{n_1} \mathbf{c}_{n_1})\right) \qquad (2.12) \\
&\quad = \tfrac{1}{\sqrt{2}} \left(V_{n_1}^T C_{n_1}^{II} J_{n_1} \operatorname{diag}(J_{n_1} \mathbf{s}_{n_1}) - I''_{n_1} \Sigma_{n_1} S_{n_1}^{II} \operatorname{diag}(J_{n_1} \mathbf{c}_{n_1})\right).
\end{aligned}$$

Using the relations (2.9) – (2.12) we get the following factorization of $P_n\, C_n^{IV}$:

$$P_n\, C_n^{IV} = \tfrac{1}{\sqrt{2}} \begin{pmatrix} I'_{n_1} & V_{n_1}\Sigma_{n_1} \\ V_{n_1}^T & -I''_{n_1}\Sigma_{n_1} \end{pmatrix} (C_{n_1}^{II} \oplus S_{n_1}^{II}) \begin{pmatrix} \operatorname{diag} \mathbf{c}_{n_1} & J_{n_1} \operatorname{diag}(J_{n_1} \mathbf{s}_{n_1}) \\ -J_{n_1} \operatorname{diag} \mathbf{s}_{n_1} & \operatorname{diag}(J_{n_1} \mathbf{c}_{n_1}) \end{pmatrix}.$$

Thus (2.8) follows by the intertwining relation (2.3), namely $S_{n_1}^{II} = J_{n_1} C_{n_1}^{II} \Sigma_{n_1}$. The orthogonality of $A_n(1)$ and $T_n(1)$ can be shown by simple calculation. Note that $T_n(1)$ consists only of n_1 plane rotations/rotation–reflections. □

Corollary 2.5 *Let $n \in \mathbb{N}$, $n \geq 8$ with $n \equiv 0 \bmod 4$ be given. Then the matrices C_n^{II}, C_n^{IV}, and C_{n+1}^{I} can be factorized as follows*

$$\begin{aligned}
C_n^{II} &= P_n^T (P_{n_1}^T \oplus P_{n_1}^T)(I_{n_1} \oplus A_{n_1}(1))(C_{n_2}^{II} \oplus C_{n_2}^{IV} \oplus C_{n_2}^{II} \oplus C_{n_2}^{II})(T_{n_1}(0) \oplus T_{n_1}(1))T_n(0), \\
C_n^{IV} &= P_n^T \left(A_n(1) \left(P_{n_1}^T \oplus P_{n_1}^T\right)(C_{n_2}^{II} \oplus C_{n_2}^{IV} \oplus C_{n_2}^{II} \oplus C_{n_2}^{IV})(T_{n_1}(0) \oplus T_{n_1}(0))\right) T_n(1), \\
C_{n+1}^{I} &= P_{n+1}^T \left(P_{n_1+1}^T \oplus T_{n_1}(0)^T\right) \left(C_{n_2+1}^{I} \oplus C_{n_2}^{III} \oplus C_{n_2}^{III} \oplus C_{n_2}^{IV}\right)(T_{n_1+1}(2) \oplus P_{n_1})\, T_{n+1}(2).
\end{aligned}$$

Now we give a detailed description of the structures of $C_4^{II}, C_4^{IV}, C_8^{II}$, and C_8^{IV}. Here, we want to assume that a matrix–vector product of the form

$$\begin{pmatrix} a_0 & a_1 \\ -a_1 & a_0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

is realized by 2 additions and 4 multiplications (see Remarks 2.10 and 2.11). As we shall see, the above factorization of C_n^{II} for $n = 8$ can be used to construct a fast algorithm which needs only 10 butterfly operations and 3 rotations/rotation–reflections.

Example 2.6 For $n = 4$ we obtain by Lemma 2.2 that

$$C_4^{II} = P_4^T (C_2^{II} \oplus C_2^{IV})\, T_4(0)$$

with

$$C_2^{II} = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad C_2^{IV} = \begin{pmatrix} \cos\frac{\pi}{8} & \sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & -\cos\frac{\pi}{8} \end{pmatrix}, \quad T_4(0) = \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_2 & J_2 \\ I_2 & -J_2 \end{pmatrix}.$$

Note that $P_4^T = P_4$. This yields

$$C_4^{II} = \tfrac{1}{2} P_4 \left(\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \oplus \sqrt{2} \begin{pmatrix} \cos\frac{\pi}{8} & \sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & -\cos\frac{\pi}{8} \end{pmatrix} \right) \begin{pmatrix} I_2 & J_2 \\ I_2 & -J_2 \end{pmatrix}$$

such that $C_4^{II}\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^4$ can be computed with 8 additions and 4 multiplications. The final scaling with 1/2 is not counted. We see that only 3 butterfly operations and 1 scaled rotation–reflection are required.

Example 2.7 For $n = 4$ we obtain by Lemma 2.4 that

$$\begin{aligned} C_4^{IV} &= P_4^T\, A_4(1)\,(C_2^{II} \oplus C_2^{II})\, T_4(1) \\ &= \tfrac{\sqrt{2}}{2}\, P_4^T\, A_4(1)\,(\sqrt{2}C_2^{II} \oplus \sqrt{2}C_2^{II})\, T_4(1) \end{aligned}$$

with

$$A_4(1) = \tfrac{1}{\sqrt{2}} \begin{pmatrix} \sqrt{2} & & & \\ & 1 & & 1 \\ & 1 & & -1 \\ & & \sqrt{2} & \end{pmatrix}, \quad T_4(1) = \begin{pmatrix} \cos\frac{\pi}{16} & & & \sin\frac{\pi}{16} \\ & \cos\frac{3\pi}{16} & \sin\frac{3\pi}{16} & \\ & -\sin\frac{3\pi}{16} & \cos\frac{3\pi}{16} & \\ \sin\frac{\pi}{16} & & & -\cos\frac{\pi}{16} \end{pmatrix}.$$

Hence, we can compute $C_4^{IV}\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^4$ with 10 additions and 10 multiplications. We see that only 3 butterfly operations, 1 rotation, and 1 rotation–reflection are required.

Example 2.8 For $n = 8$ we obtain by Corollary 2.5 that

$$C_8^{II} = P_8^T (P_4 \oplus P_4)\,(I_4 \oplus A_4(1))\,(C_2^{II} \oplus C_2^{IV} \oplus C_2^{II} \oplus C_2^{II})\,(T_4(0) \oplus T_4(1))\,T_8(0)$$

with

$$T_8(0) = \frac{1}{\sqrt{2}} \begin{pmatrix} I_4 & J_4 \\ I_4 & -J_4 \end{pmatrix}.$$

Note that $B_8 := P_8^T(P_4 \oplus P_4)$ coincides with the bit reversal matrix (see [24], pp. 36 – 43). This yields the factorization

$$C_8^{II} = \tfrac{\sqrt{2}}{4} B_8 (I_4 \oplus A_4(1))(\sqrt{2}C_2^{II} \oplus \sqrt{2}C_2^{IV} \oplus \sqrt{2}C_2^{II} \oplus \sqrt{2}C_2^{II})(\sqrt{2}T_4(0) \oplus \sqrt{2}T_4(1)) \begin{pmatrix} I_4 & J_4 \\ I_4 & -J_4 \end{pmatrix}$$

which is very similar to that of Loeffler et al. [14]. Note that

$$\sqrt{2}\, C_2^{II} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

is a butterfly matrix. If we compute $C_8^{II}\mathbf{x}$ for arbitrary $\mathbf{x} \in \mathbb{R}^8$, then the algorithm based on the above factorization requires only 26 additions and 14 multiplications (not including the final scaling by $\sqrt{2}/4$). Further we see that only 10 (scaled) butterfly operations, 1 scaled rotation, and 2 scaled rotation–reflections are used.

Example 2.9 For $n = 8$ we obtain by Corollary 2.5 that

$$\begin{aligned} C_8^{IV} &= P_8^T\, A_8(1)\,(P_4 \oplus P_4)\,(C_2^{II} \oplus C_2^{IV} \oplus C_2^{II} \oplus C_2^{IV})\,(T_4(0) \oplus T_4(0))\,T_8(1) \\ &= \tfrac{\sqrt{2}}{4} P_8^T\, \sqrt{2} A_8(1)\,(P_4 \oplus P_4)\,(\sqrt{2}C_2^{II} \oplus \sqrt{2}C_2^{IV} \oplus \sqrt{2}C_2^{II} \oplus \sqrt{2}C_2^{IV}) \\ &\quad \cdot (\sqrt{2}\, T_4(0) \oplus \sqrt{2}\, T_4(0))\, T_8(1) \end{aligned}$$

with the cross–shaped twiddle matrix

$$T_8(1) = \begin{pmatrix} \cos\frac{\pi}{32} & & & & & & & \sin\frac{\pi}{32} \\ & \cos\frac{3\pi}{32} & & & & & \sin\frac{3\pi}{32} & \\ & & \cos\frac{5\pi}{32} & & & \sin\frac{5\pi}{32} & & \\ & & & \cos\frac{7\pi}{32} & \sin\frac{7\pi}{32} & & & \\ & & & -\sin\frac{7\pi}{32} & \cos\frac{7\pi}{32} & & & \\ & & \sin\frac{5\pi}{32} & & & -\cos\frac{5\pi}{32} & & \\ & -\sin\frac{3\pi}{32} & & & & & \cos\frac{3\pi}{32} & \\ \sin\frac{\pi}{32} & & & & & & & -\cos\frac{\pi}{32} \end{pmatrix}$$

and the modified addition matrix

$$A_8(1) = \tfrac{1}{\sqrt{2}} \begin{pmatrix} \sqrt{2} & & & & & & & \\ & 1 & & & 1 & & & \\ & & 1 & & & -1 & & \\ & & & 1 & & & 1 & \\ & 1 & & & -1 & & & \\ & & 1 & & & 1 & & \\ & & & 1 & & & -1 & \\ & & & & & & & \sqrt{2} \end{pmatrix} (I_4 \oplus J_4).$$

Remark 2.10 *In [9, 14] one can find the identity*

$$\begin{pmatrix} a_0 & a_1 \\ -a_1 & a_0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} a_0 + a_1 & & \\ & a_1 & \\ & & a_0 - a_1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}.$$

If the terms $a_0 + a_1$ and $a_0 - a_1$ are precomputed, then this formula suggests an algorithm with 3 additions and 3 multiplications. Using this method, the DCT–II *of length 8 in Example 2.8 needs only 11 multiplications and 29 additions. However, the numerical stability of this method is worse than the usual algorithm with 2 additions and 4 multiplications (see [27]).*

Remark 2.11 *For a rotation matrix*

$$\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \qquad (\varphi \in (-\pi, \pi)),$$

there is a second way to realize the matrix vector multiplication with only 3 multiplications and 3 additions. Namely, using 3 lifting steps [5], one finds the factorization

$$\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} = \begin{pmatrix} 1 & \tan(\varphi/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -\sin\varphi & 1 \end{pmatrix} \begin{pmatrix} 1 & \tan(\varphi/2) \\ 0 & 1 \end{pmatrix}.$$

The above matrix factors are not orthogonal. However, it has been shown (see [27]) that for small rotation angle φ the roundoff error is less than for classical rotation. The same method is also applicable to the rotation–reflection matrix

$$\begin{pmatrix} \cos\varphi & \sin\varphi \\ \sin\varphi & -\cos\varphi \end{pmatrix} \qquad (\varphi \in (-\pi, \pi)).$$

3 Split-radix FFT

The unitary *Fourier matrix*

$$F_n := \tfrac{1}{\sqrt{n}} \, (w_n^{jk})_{j,k=0}^{n-1}$$

with $w_n := \exp(-2\pi \mathrm{i}/n)$ is closely related with cosine and sine matrices. A standard FFT is based on factorization of the Fourier matrix into a product of sparse, (almost) unitary matrices. A complex matrix A_n is called *almost unitary*, if $A_n \, \overline{A_n^T} = \alpha \, I_n$ with some $\alpha > 0$. A real, almost unitary matrix is called *almost orthogonal*.

Lemma 3.1 *For even $n \geq 4$, the Fourier matrix F_n can be factorized as follows*

$$F_n = P_n^T \, (F_{n_1} \oplus F_{n_1}) U_n, \tag{3.1}$$
$$F_n = U_n \, (F_{n_1} \oplus F_{n_1}) P_n^T \tag{3.2}$$

with the unitary matrix

$$U_n := \left(I_{n_1} \oplus \operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1}\right) \frac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & I_{n_1} \\ I_{n_1} & -I_{n_1} \end{pmatrix}.$$

Proof. We only sketch the proof of (3.1). Multiplying F_n with the permutation matrix P_n we obtain the block matrix

$$P_n \, F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} (w_n^{2jk})_{j,k=0}^{n_1-1} & (w_n^{2j(n_1+k)})_{j,k=0}^{n_1-1} \\ (w_n^{(2j+1)k})_{j,k=0}^{n_1-1} & (w_n^{(2j+1)(n_1+k)})_{j,k=0}^{n_1-1} \end{pmatrix}.$$

The four blocks can be expressed in the form

$$\begin{aligned} P_n \, F_n &= \frac{1}{\sqrt{2}} \begin{pmatrix} F_{n_1} & F_{n_1} \\ F_{n_1} \operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1} & -F_{n_1} \operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1} \end{pmatrix} \\ &= (F_{n_1} \oplus F_{n_1}) \frac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & I_{n_1} \\ \operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1} & -\operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1} \end{pmatrix} \\ &= (F_{n_1} \oplus F_{n_1}) \left(I_{n_1} \oplus \operatorname{diag}\,(w_n^k)_{k=0}^{n_1-1}\right) \frac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & I_{n_1} \\ I_{n_1} & -I_{n_1} \end{pmatrix}. \end{aligned}$$

Obviously we have $U_n \, \overline{U_n^T} = I_n$. This completes the proof of (3.1). □

Remark 3.2 *In signal processing, the even–odd permutation of rows of F_n (as realized in the above proof of (3.1)) is called "decimation–in–frequency", since the frequency–dependent row indices are decimated into even and odd indices. The even–odd permutation of columns of F_n (corresponding to (3.2)) is called "decimation–in–time", since the time–dependent column indices are splitted (see [24], pp. $67 - 68$).*

In the case $n = 2^t$ $(t \geq 2)$, recursive application of (3.1) provides the Gentleman–Sande factorization of F_n (see [24], pp. 65 – 67). Recursive application of the transposed factorization yields the well–known Cooley–Tukey factorization (see [24], pp. 17 – 21). The corresponding FFT are also known as radix–2 algorithms. A better arithmetical complexity can be achieved using the so–called split–radix algorithm. The idea of split–radix FFT, due to [6, 7], can be described as a clever synthesis of radix–2 and radix–4 FFTs (see [24], pp. 111 – 119).

In this section we want to derive a new approach to the split–radix FFT which is completely based on the matrix factorization of F_n. In the next sections, we will apply the same ideas to derive new split–radix DCT–algorithms.

Let $n = 2^t$ $(t \geq 2)$. We set $n_s := n \, 2^{-s} = 2^{t-s}$ for $s = 0, \ldots, t-1$. Further, we introduce the *odd Fourier matrix*

$$F_n^{(1)} := \frac{1}{\sqrt{n}} \left(w_{2n}^{(2j+1)k}\right)_{j,k=0}^{n-1}.$$

This matrix is unitary, too. Note that F_n is also called the even Fourier matrix. We shall show that F_n can be splitted into $F_{n_1} \oplus F_{n_1}^{(1)}$ in a first step, then $F_{n_1} \oplus F_{n_1}^{(1)}$ can be splitted into $F_{n_2} \oplus F_{n_2}^{(1)} \oplus F_{n_2} \oplus F_{n_2}$ in a second step and so on. This procedure follows from

Lemma 3.3 *For even $n \geq 4$, the matrices F_n and $F_n^{(1)}$ can be factorized as follows*

$$F_n = P_n^T \, (F_{n_1} \oplus F_{n_1}^{(1)}) \, W_n(0), \tag{3.3}$$

$$F_n^{(1)} = P_n^T \, (F_{n_1} \oplus F_{n_1}) \, W_n(1) \tag{3.4}$$

with the unitary matrices

$$W_n(0) := \tfrac{1}{\sqrt{2}} \begin{pmatrix} I_{n_1} & I_{n_1} \\ I_{n_1} & -I_{n_1} \end{pmatrix}, \quad W_n(1) := (D_{n_1} \oplus D_{n_1}^3) \, W_n(0) \, (I_{n_1} \oplus (-\mathrm{i} I_{n_1})),$$

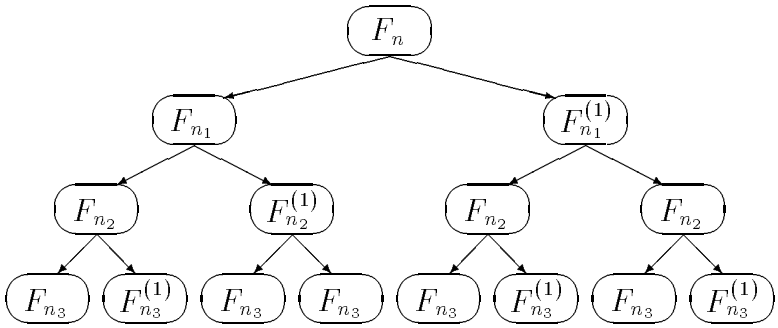
where $D_{n_1} := \operatorname{diag} (w_{2n}^k)_{k=0}^{n_1-1}$.

The proof is similar to that of Lemma 3.1 and is omitted here.

For $n = 2^t$ $(t \geq 3)$, we conclude from Lemma 3.3 that

$$F_n = P_n^T \, (P_{n_1}^T \oplus P_{n_1}^T) \, (F_{n_2} \oplus F_{n_2}^{(1)} \oplus F_{n_2} \oplus F_{n_2}) \, (W_{n_1}(0) \oplus W_{n_1}(1)) \, W_n(0).$$

Recursive application of Lemma 3.3 provides the split–radix factorization of F_n on which the split–radix FFT is based. The factorization steps can be illustrated in the following diagram:



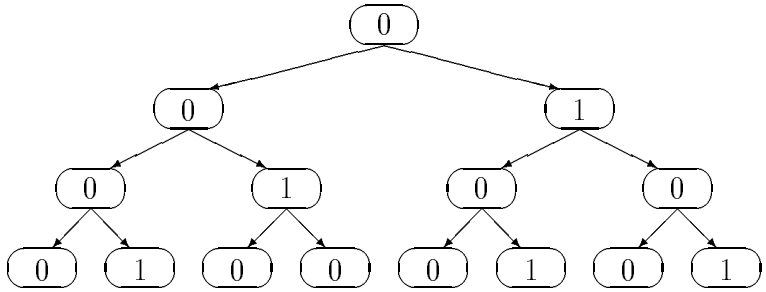
step $s = 0$

step $s = 1$

step $s = 2$

step $s = 3$

Now we shall answer the following problem: On which positions $k \in \{1, \ldots, 2^s\}$ in step $s \in \{0, \ldots, t-1\}$ occur F_{n_s} and $F_{n_s}^{(1)}$, respectively?
We introduce the binary vectors $\boldsymbol{\beta}_s := (\beta_s(1), \ldots, \beta_s(2^s))$. If F_{n_s} stands at position k in step s, then let $\beta_s(k) := 0$. We put $\beta_s(k) := 1$, if $F_{n_s}^{(1)}$ stands at position k in step s (see [24], pp. 115 – 119). By Lemma 3.3, from $\beta_s(k) := 0$ it follows immediately that $\beta_{s+1}(2k-1) = 0$ and $\beta_{s+1}(2k) = 1$. Further, from $\beta_s(k) := 1$ it follows that $\beta_{s+1}(2k-1) = \beta_{s+1}(2k) = 0$. Now, our diagram looks as follows:



$\boldsymbol{\beta}_0 = (0)$

$\boldsymbol{\beta}_1 = (0, 1)$

$\boldsymbol{\beta}_2 = (0, 1, 0, 0)$

$\boldsymbol{\beta}_3 = (0, 1, 0, 0, 0, 1, 0, 1)$

Lemma 3.4 *Let $t \in \mathbb{N}$ with $t \geq 2$ be given and $\boldsymbol{\beta}_0 := (0)$. Then*

$$\boldsymbol{\beta}_{s+1} = (\boldsymbol{\beta}_s, \, \tilde{\boldsymbol{\beta}}_s), \qquad s = 0, \ldots, t-2, \tag{3.5}$$

where $\tilde{\boldsymbol{\beta}}_s$ equals $\boldsymbol{\beta}_s$ with the exception that the last bit position is reversed. Further,

$$\|\boldsymbol{\beta}_s\|_1 = \sum_{k=1}^{2^s} \beta_s(k) = \tfrac{1}{3}\,(2^s - (-1)^s) \tag{3.6}$$

is the number of ones in the binary vector $\boldsymbol{\beta}_s$.

Proof. The assertion (3.5) follows by induction. For $s = 0$ we have $\boldsymbol{\beta}_0 = (0)$ and $\boldsymbol{\beta}_1 = (0, 1)$ by Lemma 3.3 such that $\boldsymbol{\beta}_1 = (\boldsymbol{\beta}_0, \tilde{\boldsymbol{\beta}}_0)$.
Assume that for arbitrary $s \in \{1, \ldots, t-2\}$,

$$\boldsymbol{\beta}_s = (\boldsymbol{\beta}_{s-1},\, \tilde{\boldsymbol{\beta}}_{s-1}).$$

Then we have by definition

$$\tilde{\boldsymbol{\beta}}_s = (\boldsymbol{\beta}_{s-1},\, \boldsymbol{\beta}_{s-1}). \tag{3.7}$$

In step $s+1$, the first part of vector $\boldsymbol{\beta}_s$, namely $\boldsymbol{\beta}_{s-1}$, turns into $(\boldsymbol{\beta}_{s-1},\, \tilde{\boldsymbol{\beta}}_{s-1})$ by our assumption. The second part of $\boldsymbol{\beta}_s$, namely $\tilde{\boldsymbol{\beta}}_{s-1}$, changes into $(\boldsymbol{\beta}_{s-1},\, \boldsymbol{\beta}_{s-1})$ by assumption. Consequently, we obtain $\boldsymbol{\beta}_{s+1} = (\boldsymbol{\beta}_{s-1},\, \tilde{\boldsymbol{\beta}}_{s-1},\, \boldsymbol{\beta}_{s-1},\, \boldsymbol{\beta}_{s-1}) = (\boldsymbol{\beta}_s,\, \tilde{\boldsymbol{\beta}}_s)$.
The 1–norm of $\boldsymbol{\beta}_s$ equals the number of ones in $\boldsymbol{\beta}_s$. By (3.3) we know that $\beta_s(k) = 0$ implies $\beta_{s+1}(2k-1) = 0$ and $\beta_{s+1}(2k) = 1$. By (3.4) we see that $\beta_s(k) = 1$ implies $\beta_{s+1}(2k-1) = 0$ and $\beta_{s+1}(2k) = 0$. Thus we have

$$\|\boldsymbol{\beta}_s\|_1 + \|\boldsymbol{\beta}_{s+1}\|_1 = 2^s, \quad \|\boldsymbol{\beta}_0\|_1 = 0.$$

Using classical difference equation theory we obtain (3.6). □

We introduce permutation matrices

$$P_n(s) := P_{n_s}^T \oplus \ldots \oplus P_{n_s}^T, \qquad s = 0, \ldots, t-2,$$

and for each pointer $\boldsymbol{\beta}_s$

$$W_n(\boldsymbol{\beta}_s) := W_{n_s}(\beta_s(1)) \oplus \ldots \oplus W_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-1,$$

with $W_n(0)$ and $W_n(1)$ as in Lemma 3.3. Note that $B_n := P_n(0) \ldots P_n(t-2)$ is the bit reversal matrix (see [24], pp. 36 – 43). Further, $W_n(\boldsymbol{\beta}_{t-1})$ is a diagonal block matrix, where each block in the diagonal is either

$$F_2 = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{or} \quad F_2^{(1)} = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & -\mathrm{i} \\ 1 & \mathrm{i} \end{pmatrix}.$$

Recursive application of Lemma 3.3 provides the split–radix factorization of F_n.

Theorem 3.5 *Let $n = 2^t$ $(t \geq 3)$. Then the Fourier matrix F_n can be factorized into the following product of sparse unitary matrices*

$$F_n = B_n\, W_n(\boldsymbol{\beta}_{t-1}) W_n(\boldsymbol{\beta}_{t-2}) \ldots W_n(\boldsymbol{\beta}_0).$$

In order to reduce the number of multiplications, the factor $\sqrt{2}/2$ in each matrix $W_n(\boldsymbol{\beta}_s)$ is moved to the end of calculation. From

$$F_n = \tfrac{1}{\sqrt{n}}\, B_n\, \sqrt{2}\, W_n(\boldsymbol{\beta}_{t-1})\, \sqrt{2}\, W_n(\beta_{t-2}) \ldots\, \sqrt{2}\, W_n(\beta_0)$$

we obtain the following split–radix FFT:

Algorithm 3.6 [Split–radix FFT]
Input: $n = 2^t$ $(t \geq 3)$, $\mathbf{x} \in \mathbb{C}^n$.

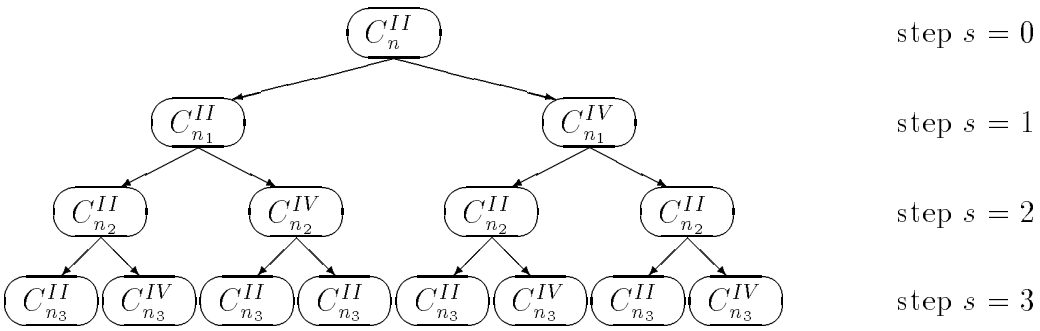
1. For $s = 3, \ldots, t$ precompute the roots of unity $w_{2^s}^k$ $(k = 1, \ldots, 2^{s-2} - 1)$.
2. For $s = 0, \ldots, t-2$ form $\boldsymbol{\beta}_{s+1} := (\boldsymbol{\beta}_s, \tilde{\boldsymbol{\beta}}_s)$ with $\boldsymbol{\beta}_0 = (0)$.
3. Put $\mathbf{x}^{(0)} := \mathbf{x}$ and compute for $s = 0, \ldots, t-1$
$$\mathbf{x}^{(s+1)} := \sqrt{2}\, W_n(\boldsymbol{\beta}_s)\mathbf{x}^{(s)}.$$
4. Permute $\mathbf{x}^{(t)} := B_n\, \mathbf{x}^{(t-1)}$.
5. Multiply with scaling factor $\mathbf{y} := \frac{1}{\sqrt{n}}\, \mathbf{x}^{(t)}$.

Output: $\mathbf{y} = F_n \mathbf{x}$.

The split–radix FFT is known to be one of the FFT with lowest arithmetical complexity, in particular, it has a lower complexity than radix–2 and radix–4 FFTs (see e.g. [24], p. 119).

4 Split-radix DCT-II algorithms

In this section, we present split–radix algorithms for the DCT–II of radix–2 length. Recursive application of (2.4) and (2.8) provides the split–radix factorization of C_n^{II}. The corresponding complete factorization can be derived similarly as worked out in Section 3 for the Fourier matrix. Note that there does not exist a factorization of C_n^{II} or C_n^{IV} which is analogously to that of Lemma 3.1. Therefore a split–radix DCT–II algorithm seems to be very natural. A split–radix DCT–III algorithm follows immediately from a split–radix factorization of C_n^{II} by transposing.
Let $n = 2^t$ $(t \geq 2)$ be given. Further, let $n_s := 2^{t-s}$ $(s = 0, \ldots, t-1)$. In the first factorization step, C_n^{II} is splitted into $C_{n_1}^{II} \oplus C_{n_1}^{IV}$ by (2.4). Then in the second step, we use (2.4) and (2.8) in order to split $C_{n_1}^{II} \oplus C_{n_1}^{IV}$ into $C_{n_2}^{II} \oplus C_{n_2}^{IV} \oplus C_{n_2}^{II} \oplus C_{n_2}^{II}$. In the case $n_2 > 2$ we continue this procedure. Finally, we obtain the split–radix factorization of C_n^{II}. The first factorization steps are illustrated by the following diagram:



As in Section 3, we introduce binary vectors $\boldsymbol{\beta}_s = (\beta_s(1), \ldots, \beta_s(2^s))$ for $s \in \{0, \ldots, t-1\}$. We put $\beta_s(k) := 0$ if $C_{n_s}^{II}$ stands at position $k \in \{1, \ldots, 2^s\}$ in step s, and $\beta_s(k) := 1$ if $C_{n_s}^{IV}$ stands at position k in step s. These pointers $\boldsymbol{\beta}_s$ have the same properties as in Section 3.

For each pointer $\boldsymbol{\beta}_s$ we define the modified addition matrix

$$A_n(\boldsymbol{\beta}_s) := A_{n_s}(\beta_s(1)) \oplus \ldots \oplus A_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-2$$

with $A_{n_s}(0) := I_{n_s}$ and $A_{n_s}(1)$ as in Lemma 2.4, further the modified twiddle matrix

$$T_n(\boldsymbol{\beta}_s) := T_{n_s}(\beta_s(1)) \oplus \ldots \oplus T_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-1$$

with $T_{n_s}(0)$ and $T_{n_s}(1)$ as in Lemma 2.2 and Lemma 2.4, and finally the cosine matrix

$$C_n(\boldsymbol{\beta}_s) := C_{n_s}(\beta_s(1)) \oplus \ldots \oplus C_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-1$$

with $C_{n_s}(0) := C_{n_s}^{II}$ and $C_{n_s}(1) := C_{n_s}^{IV}$. Let the permutation matrices $P_n(s)$ be given as in Section 3.
Note that $C_n(\boldsymbol{\beta}_0) = C_n^{II}$. We shall see in the following that the cosine matrix $C_n(\boldsymbol{\beta}_s)$ appears as intermediate result in our recursive factorization of C_n^{II}. The matrix $C_n(\boldsymbol{\beta}_{t-1}) = T_n(\boldsymbol{\beta}_{t-1})$ is a diagonal block matrix, where each block in the diagonal is either C_2^{II} or C_2^{IV}. By construction, all matrices $P_n(s)$, $A_n(\boldsymbol{\beta}_s)$ and $T_n(\boldsymbol{\beta}_s)$ are sparse and orthogonal.

Theorem 4.1 *Let $n = 2^t$ $(t \geq 2)$. Then C_n^{II} can be factorized into the following product of sparse orthogonal matrices*

$$C_n^{II} = (P_n(0)A_n(\boldsymbol{\beta}_0)) \ldots (P_n(t-2)A_n(\boldsymbol{\beta}_{t-2}))T_n(\boldsymbol{\beta}_{t-1}) \ldots T_n(\boldsymbol{\beta}_0). \tag{4.1}$$

Proof. The split–radix factorization (4.1) follows immediately from

$$C_n(\boldsymbol{\beta}_s) = P_n(s)A_n(\boldsymbol{\beta}_s)C_n(\boldsymbol{\beta}_{s+1})T_n(\boldsymbol{\beta}_s), \qquad s = 0, \ldots, t-2. \tag{4.2}$$

By definition of $P_n(s)$, $A_n(\boldsymbol{\beta}_s)$, $C_n(\boldsymbol{\beta}_s)$ and $T_n(\boldsymbol{\beta}_s)$, these formulas (4.2) are direct consequences of Lemma 2.2 and Lemma 2.4. □

In order to reduce the number of multiplications, we move the factors $1/\sqrt{2}$ in the matrices $T_n(\boldsymbol{\beta}_s)$, $s = 0, \ldots, t-1$ to the end of the calculation and obtain with the scaled, almost orthogonal matrices $S_n(\boldsymbol{\beta}_s) := \sqrt{2}\, T_n(\boldsymbol{\beta}_s)$ the factorization

$$C_n^{II} = \tfrac{1}{\sqrt{n}} (P_n(0)A_n(\boldsymbol{\beta}_0)) \ldots (P_n(t-2)A_n(\boldsymbol{\beta}_{t-2}))S_n(\boldsymbol{\beta}_{t-1}) \ldots S_n(\boldsymbol{\beta}_0). \tag{4.3}$$

This implies the following fast algorithm for the DCT–II of radix–2 length:

Algorithm 4.2 [Split–radix DCT–II algorithm]
Input: $n = 2^t$ $(t \geq 2)$, $\mathbf{x} \in \mathbb{R}^n$.

1. Precompute $\sqrt{2}/2$ and for $s = 0, \ldots, t-2$ and $k = 0, \ldots, 2^s - 1$ the values $\sqrt{2}\cos\frac{(2k+1)\pi}{2^{s+3}}$ and $\sqrt{2}\sin\frac{(2k+1)\pi}{2^{s+3}}$.

2. For $s = 0, \ldots, t-2$ form $\boldsymbol{\beta}_{s+1} := (\boldsymbol{\beta}_s, \tilde{\boldsymbol{\beta}}_s)$ with $\boldsymbol{\beta}_0 := (0)$.

3. Put $\mathbf{x}^{(0)} := \mathbf{x}$ and compute for $s = 0, \ldots, t-1$

$$\mathbf{x}^{(s+1)} := S_n(\boldsymbol{\beta}_s)\mathbf{x}^{(s)}.$$

4. For $s = 0, \ldots, t-2$ compute

$$\mathbf{x}^{(t+s+1)} := P_n(t-s-2)\, A_n(\boldsymbol{\beta}_{t-s-2})\; \mathbf{x}^{(t+s)}.$$

5. Multiply with scaling factor $\mathbf{y} := \frac{1}{\sqrt{n}}\mathbf{x}^{(2t-1)}$.

Output: $\mathbf{y} = C_n^{II}\mathbf{x}$.

Now we determine the arithmetical complexity of this split–radix DCT–II algorithm. As usual, the final scaling by $n^{-1/2}$ is not counted. For an arbitrary real matrix A_n of order n, let $\alpha(A_n)$ and $\mu(A_n)$ denote the number of additions and multiplications for computing $A_n\,\mathbf{x}$ with arbitrary $\mathbf{x} \in \mathbb{R}^n$.

Theorem 4.3 *Let $n = 2^t$ $(t \geq 2)$ be given. Then the split–radix Algorithm 4.2 for the* DCT–II *of length n possesses the following arithmetical complexity*

$$\begin{aligned} \alpha(C_n^{II}) &= \tfrac{4}{3}\,nt - \tfrac{8}{9}\,n - \tfrac{1}{9}(-1)^t + 1, \\ \mu(C_n^{II}) &= nt - \tfrac{4}{3}\,n + \tfrac{1}{3}\,(-1)^t + 1. \end{aligned}$$

Proof. By (4.3) it follows that

$$\alpha(C_n^{II}) = \sum_{s=0}^{t-1} \alpha(S_n(\boldsymbol{\beta}_s)) + \sum_{s=0}^{t-2} \alpha(A_n(\boldsymbol{\beta}_s)), \tag{4.4}$$

$$\mu(C_n^{II}) = \sum_{s=0}^{t-2} \mu(S_n(\boldsymbol{\beta}_s)) + \sum_{s=0}^{t-2} \mu(A_n(\boldsymbol{\beta}_s)). \tag{4.5}$$

By definition of $A_n(\boldsymbol{\beta}_s)$ and $S_n(\boldsymbol{\beta}_s)$ we obtain

$$\alpha(S_n(\boldsymbol{\beta}_s)) = n, \qquad \mu(S_n(\boldsymbol{\beta}_s)) = 2n_s\, \|\boldsymbol{\beta}_s\|_1,$$

$$\alpha(A_n(\boldsymbol{\beta}_s)) = \mu(A_n(\boldsymbol{\beta}_s)) = (n_s - 2)\, \|\boldsymbol{\beta}_s\|_1.$$

By (3.6) we have $\|\boldsymbol{\beta}_s\|_1 = \frac{1}{3}\,(2^s - (-1)^s)$. Inserting these values into (4.4) and (4.5) yields the assertions of the theorem. □

We want to derive a second algorithm which possesses even lower arithmetical complexity. We slightly change the derived orthogonal matrix product (4.1) in the following way. Instead of $A_n(\boldsymbol{\beta}_s)$, consider the modified addition matrices

$$A'_n(\boldsymbol{\beta}_s) = A'_{n_s}(\beta_s(1)) \oplus \ldots \oplus A'_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-2$$

with $A'_{n_s}(0) := A_{n_s}(0) = I_{n_s}$ and $A'_{n_s}(1) := \sqrt{2}\, A_{n_s}(1)$. Hence $A_n(\boldsymbol{\beta}_s)$ and $A'_n(\boldsymbol{\beta}_s)$ are connected by

$$A'_n(\boldsymbol{\beta}_s) = D_n(\boldsymbol{\beta}_s)\, A_n(\boldsymbol{\beta}_s)$$

with a diagonal matrix

$$D_n(\boldsymbol{\beta}_s) := (\sqrt{2})^{\beta_s(1)} I_{n_s} \oplus \ldots \oplus (\sqrt{2})^{\beta_s(2^s)} I_{n_s}, \quad s = 0, \ldots, t-2. \tag{4.6}$$

Further, consider the modified twiddle matrices

$$S'_n(\boldsymbol{\beta}_s) := S'_{n_s}(\beta_s(1)) \oplus \ldots \oplus S'_{n_s}(\beta_s(2^s)), \qquad s = 0, \ldots, t-2$$

with $S'_{n_s}(0) := \sqrt{2}\, T_{n_s}(0)$ and $S'_{n_s}(1) := T_{n_s}(1)$ such that

$$S'_n(\boldsymbol{\beta}_s) = (D_n(\boldsymbol{\beta}_s))^{-1}\, S_n(\boldsymbol{\beta}_s).$$

As before, the matrices $A'_n(\boldsymbol{\beta}_s)$ and $S'_n(\boldsymbol{\beta}_s)$ are sparse matrices with at most 2 nonzero entries in each row. More precisely, after suitable permutations, $A'_n(\boldsymbol{\beta}_s)$, $s = 1, \ldots, t-2$, contains only butterfly matrices (with entries ± 1) and diagonal matrices (with entries 1 and $\sqrt{2}$). The matrices $S'_n(\boldsymbol{\beta}_s)$, $s = 0, \ldots, t-2$ only contains butterfly matrices (with entries ± 1), rotation matrices, and rotation–reflection matrices. Note that $A'_n(\boldsymbol{\beta}_0) = A_n(0) = I_n$ and $S'_n(\boldsymbol{\beta}_0) = \sqrt{2}\, T_n(0)$.
With the changed matrices we find from (4.1) the factorization

$$C_n^{II} = \frac{1}{(\sqrt{2})^{t-1}} P_n(0) A'_n(\boldsymbol{\beta}_0) \ldots P_n(t-2) A'_n(\boldsymbol{\beta}_{t-2}) T_n(\boldsymbol{\beta}_{t-1}) S'_n(\boldsymbol{\beta}_{t-2}) \ldots S'_n(\boldsymbol{\beta}_0),$$

since for each $s = 0, \ldots, t-2$ the product of diagonal matrices $D_n(\boldsymbol{\beta}_s)^{-1} \ldots D_n(\boldsymbol{\beta}_0)^{-1}$ commutes with $T_n(\boldsymbol{\beta}_{s+1})$ and with $A_n(\boldsymbol{\beta}_s)$. Observe that the inner matrix $T_n(\boldsymbol{\beta}_{t-1})$ is not modified. For the algorithm, we multiply this matrix with $\sqrt{2}$ and finally obtain

$$C_n^{II} = \tfrac{1}{\sqrt{n}}\, P_n(0)\, A'_n(\boldsymbol{\beta}_0) \ldots P_n(t-2)\, A'_n(\boldsymbol{\beta}_{t-2}) S_n(\boldsymbol{\beta}_{t-1})\, S'_n(\boldsymbol{\beta}_{t-2}) \ldots S'_n(\boldsymbol{\beta}_0). \tag{4.7}$$

This factorization leads to the following modified split–radix DCT–II algorithm:

Algorithm 4.4 [Modified split–radix DCT–II algorithm]
Input: $n = 2^t$ $(t \geq 2)$, $\mathbf{x} \in \mathbb{R}^n$.

1. Precompute $\sqrt{2}$, $\sqrt{2}\cos\frac{\pi}{8}$, $\sqrt{2}\sin\frac{\pi}{8}$, and for $s = 1, \ldots, t-2$; $k = 0, \ldots, 2^s - 1$ precompute the values $\cos\frac{(2k+1)\pi}{2^{s+3}}$ and $\sin\frac{(2k+1)\pi}{2^{s+3}}$.

2. For $s = 0, \ldots, t-2$ form $\boldsymbol{\beta}_{s+1} := (\boldsymbol{\beta}_s, \tilde{\boldsymbol{\beta}}_s)$ with $\boldsymbol{\beta}_0 := (0)$.

3. Put $\mathbf{x}^{(0)} := \mathbf{x}$ and compute for $s = 0, \ldots, t-2$

$$\mathbf{x}^{(s+1)} := S'_n(\boldsymbol{\beta}_s)\mathbf{x}^{(s)}.$$

4. Compute $\mathbf{x}^{(t)} := S_n(\boldsymbol{\beta}_{t-1})\mathbf{x}^{(t-1)}$.

5. For $s = 0, \ldots, t-2$ compute

$$\mathbf{x}^{(t+1+s)} := P_n(t-s-2)\, A'_n(\boldsymbol{\beta}_{t-s-2})\, \mathbf{x}^{(t+s)}.$$

6. Multiply with scaling factor $\mathbf{y} := \frac{1}{\sqrt{n}}\, \mathbf{x}^{(2t-1)}$.

Output: $\mathbf{y} = C_n^{II}\mathbf{x}$.

The arithmetical complexity of this modified split–radix DCT–II algorithm can be determined analogously as in Theorem 4.3. By (4.7) it follows that

$$\begin{aligned}\alpha(C_n^{II}) &= \alpha(S_n(\boldsymbol{\beta}_{t-1})) + \sum_{s=0}^{t-2} \alpha(S'_n(\boldsymbol{\beta}_s)) + \sum_{s=0}^{t-2} \alpha(A'_n(\boldsymbol{\beta}_s)),\\ \mu(C_n^{II}) &= \mu(S_n(\boldsymbol{\beta}_{t-1})) + \sum_{s=0}^{t-2} \mu(S'_n(\boldsymbol{\beta}_s)) + \sum_{s=0}^{t-2} \mu(A'_n(\boldsymbol{\beta}_s)).\end{aligned}$$

By definition of $A'_n(\boldsymbol{\beta}_s)$ and $S'_n(\boldsymbol{\beta}_s)$ we obtain for $s = 0, \ldots, t-2$

$$\begin{aligned}\alpha(S'_n(\boldsymbol{\beta}_s)) &= n, \qquad \mu(S'_n(\boldsymbol{\beta}_s)) = 2n_s \,\|\boldsymbol{\beta}_s\|_1,\\ \alpha(A'_n(\boldsymbol{\beta}_s)) &= (n_s - 2)\,\|\boldsymbol{\beta}_s\|_1, \qquad \mu(A'_n(\boldsymbol{\beta}_s)) = 2\,\|\boldsymbol{\beta}_s\|_1\\ \alpha(S_n(\boldsymbol{\beta}_{t-1})) &= n, \qquad \mu(S_n(\boldsymbol{\beta}_{t-1})) = 4\,\|\boldsymbol{\beta}_{t-1}\|_1.\end{aligned}$$

By (3.6) we have $\|\boldsymbol{\beta}_s\|_1 = \frac{1}{3}(2^s - (-1)^s)$. Inserting these values into the above equations yields for Algorithm 4.4

$$\begin{aligned}\alpha(C_n^{II}) &= \tfrac{4}{3}\,nt - \tfrac{8}{9}\,n - \tfrac{1}{9}(-1)^t + 1,\\ \mu(C_n^{II}) &= \tfrac{2}{3}\,nt - \tfrac{1}{9}\,n + \tfrac{1}{9}(-1)^t - 1.\end{aligned}$$

Remark 4.5 *For comparison, the algorithm presented by Wang [26] (which already outperforms the algorithm in [4]) for the* DCT–II *of length* $n = 2^t$ *needs* $\frac{3}{4}\,nt - n + 3$ *multiplications and* $\frac{7}{4}\,nt - 2n + 3$ *additions. The complexity of our new Algorithm* 4.4 *is even comparible with fast* DCT–II *algorithms based on polynomial arithmetic which need* $\frac{1}{2}\,nt$ *multiplications and* $\frac{3}{2}\,nt - n + 1$ *additions (see e.g. [12, 13, 20, 21]). Namely, using the method of Remark* 2.10 *(or Remark* 2.11*) computing a (scaled) rotation matrix/rotation–reflection matrix with only* 3 *multiplications and* 3 *additions, we obtain for Algorithm* 4.4

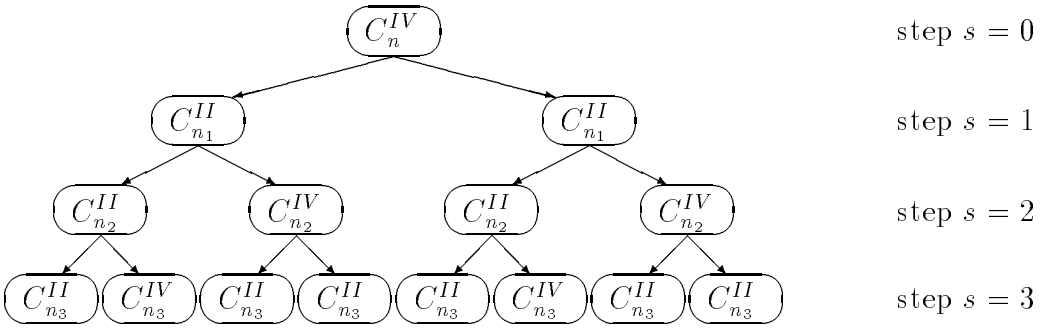
$$\begin{aligned}\alpha(S'_n(\boldsymbol{\beta}_s)) &= \tfrac{1}{2}\,n_s\,\|\boldsymbol{\beta}_s\|_1 + n, \qquad \mu(S'_n(\boldsymbol{\beta}_s)) = \tfrac{3}{2}\,n_s\,\|\boldsymbol{\beta}_s\|_1, \qquad s = 0, \ldots, t-2,\\ \alpha(T_n(\boldsymbol{\beta}_{t-1})) &= \|\boldsymbol{\beta}_{t-1}\|_1 + n, \qquad \mu(T_n(\boldsymbol{\beta}_{t-1}) = 3\,\|\boldsymbol{\beta}_{t-1}\|_1.\end{aligned}$$

Hence we get $\alpha(C_n^{II}) = \frac{3}{2}nt - n + 1$ *and* $\mu(C_n^{II}) = \frac{1}{2}nt - 1$.
The goal of [24], p. 229 *was to develop* FFT*–based* DCT–II *algorithms that require* $2.5\,n\,\log_2 n$ *flops, assuming that the transform length* n *is a power of* 2. *We have shown that this is also possible with* $2\,n\,\log_2 n$ *flops using only operations of simple structure, namely permutations, scaling with* $\sqrt{2}$*, butterfly operations, and plane rotations/rotation–reflections with small rotation angles contained in* $(0, \pi/4)$.

5 Split-radix DCT-IV algorithms

The same split–radix technique as in Sections 3 and 4 can be applied to the DCT–IV of radix–2 length.
Let $n = 2^t$ $(t \geq 2)$ be given. In the first step we can split C_n^{IV} into $C_{n_1}^{II} \oplus C_{n_1}^{II}$ by (2.8). Then in the second step, we use (2.4) in order to split $C_{n_1}^{II} \oplus C_{n_1}^{II}$ into $C_{n_1}^{II} \oplus C_{n_1}^{IV} \oplus C_{n_1}^{II} \oplus C_{n_1}^{IV}$.

In the case $n_2 \geq 4$, we continue the procedure. This method is illustrated by the following diagram:



Analogously as in Sections 3 and 4, we introduce binary vectors $\boldsymbol{\gamma}_s := (\gamma_s(1), \ldots, \gamma_s(2^s))$, $s \in \{0, \ldots, t-1\}$. We put $\gamma_s(k) := 0$ if $C_{n_s}^{II}$ stands at position $k \in \{1, \ldots, 2^s\}$ of step s, and $\gamma_s(k) := 1$ if $C_{n_s}^{IV}$ stands at position k of step s. These pointers possess different properties as in Lemma 3.4.

Lemma 5.1 *Let $t \in \mathbb{N}$ $(t \geq 2)$ and $\boldsymbol{\gamma}_0 := (1)$. Then*

$$\boldsymbol{\gamma}_{s+1} = (\tilde{\boldsymbol{\gamma}}_s, \tilde{\boldsymbol{\gamma}}_s), \qquad s = 0, \ldots, t-2,$$

where $\tilde{\boldsymbol{\gamma}}_s$ equals $\boldsymbol{\gamma}_s$ with the exception that the last bit position is reversed. Further,

$$\|\boldsymbol{\gamma}_s\|_1 = \sum_{k=1}^{2^s} \gamma_s(k) = \tfrac{1}{3}(2^s + 2(-1)^s).$$

The proof is similar to that of Lemma 3.4 and omitted here. Now, for each pointer $\boldsymbol{\gamma}_s$ we define $A_n(\boldsymbol{\gamma}_s)$ and $T_n(\boldsymbol{\gamma}_s)$ (or their modified versions) in the same way as $A_n(\boldsymbol{\beta}_s)$ and $T_n(\boldsymbol{\beta}_s)$ in Section 4.

Theorem 5.2 *Let $n = 2^t$ $(t \geq 2)$. Then the matrix C_n^{IV} can be factorized into the following product of sparse orthogonal matrices*

$$C_n^{IV} = P_n(0) A_n(\boldsymbol{\gamma}_0) \ldots P_n(t-2) A_n(\boldsymbol{\gamma}_{t-2}) T_n(\boldsymbol{\gamma}_{t-1}) \ldots T_n(\boldsymbol{\gamma}_0).$$

The proof directly follows from Lemma 2.2 and Lemma 2.4.
The factorization of C_n^{IV} in Theorem 5.2 (slightly changed by diagonal matrices as in Section 4) implies fast DCT–IV algorithms analogously to Algorithm 4.2 or Algorithm 4.4.
For computing $C_n^{IV}\,\mathbf{x}$ for $\mathbf{x} \in \mathbb{R}^n$ using Lemma 2.4 and Algorithm 4.4 we derive an arithmetical complexity

$$\begin{aligned} \alpha(C_n^{IV}) &= 2\,\alpha(C_{n_1}^{II}) + \alpha(\sqrt{2}\,A_n(1)) + \alpha(T_n(1)) \\ &= \tfrac{4}{3}\,nt - \tfrac{2}{9}\,n + \tfrac{2}{9}\,(-1)^t, \end{aligned}$$

$$\begin{aligned} \mu(C_n^{IV}) &= 2\,\mu(C_{n_1}^{II}) + \mu(\sqrt{2}\,A_n(1)) + \mu(T_n(1)) \\ &= \tfrac{2}{3}\,nt + \tfrac{11}{9}\,n - \tfrac{2}{9}\,(-1)^t + 1. \end{aligned}$$

We see that DCT–IV of radix–2 length n can be computed by $(1 + 2\,\log_2 n)\,n$ flops. With the method of Remark 2.10 (or Remark 2.11), the number of multiplications can even be reduced further.

6 Split-radix DCT-I algorithms

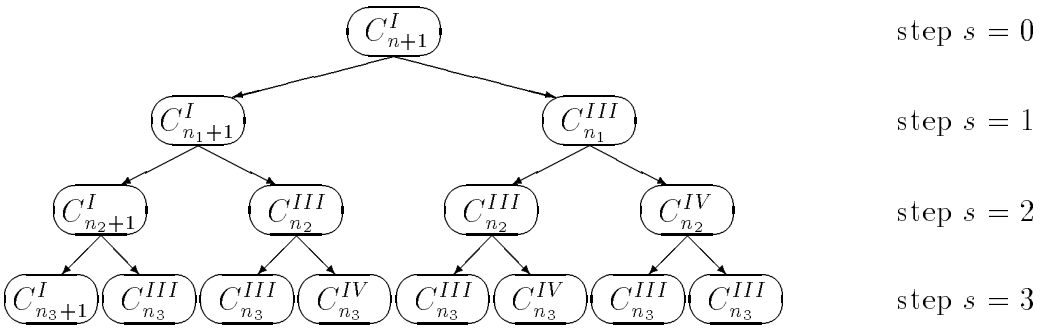
A split–radix DCT–I algorithm is based on (2.5), (2.7), and (2.8). Let $n = 2^t$ $(t \geq 2)$ be given. Further, let $n_s := 2^{t-s}$ $(s = 0, \ldots, t-1)$. In the first factorization step, C^I_{n+1} is splitted into $C^I_{n_1+1} \oplus C^{III}_{n_1}$ by (2.5). Then in the second step, we use (2.5) and (2.7) in order to split $C^I_{n_1+1} \oplus C^{III}_{n_1}$ into $C^I_{n_2+1} \oplus C^{III}_{n_2} \oplus C^{III}_{n_2} \oplus C^{IV}_{n_2}$. In the case $n_2 > 2$ we continue this procedure. For C^{IV}_n we use the factorization

$$C^{IV}_n = T_n(1)^T \, (C^{III}_{n_1} \oplus C^{III}_{n_1}) \, A_n(1)^T \, P_n$$

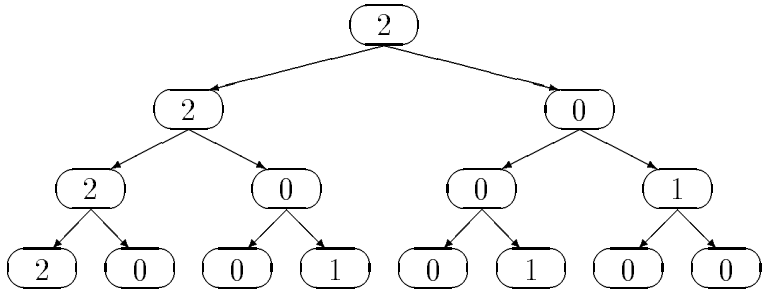
which follows from (2.8) by transposing. Finally, we obtain the split–radix factorization of C^I_{n+1}. Note that (2.5) is in some sense also true for $n = 2$:

$$C^I_3 = P_3^T \, (C^{II}_2 \oplus 1) \, T_3(2),$$

i.e., the DCT–I of length 3 can be computed by 2 (scaled) butterfly operations.
The first factorization steps can be illustrated by the following diagram:



Now we have to indicate on which position $k \in \{1, \ldots, 2^k\}$ in step $s \in \{0, \ldots, t-1\}$ stands $C^I_{n_s+1}$, $C^{III}_{n_s}$, and $C^{IV}_{n_s}$, respectively. We introduce triadic vectors $\boldsymbol{\delta}_s = (\delta_s(1), \ldots, \delta_s(2^s))$ for $s \in \{0, \ldots, t-1\}$ as pointers. Since $C^I_{n_s+1}$ stands at first position of each step s, we set $\delta_s(1) := 2$. We put $\delta_s(k) := 0$ if $C^{III}_{n_s}$ stands at position $k \in \{2, \ldots, 2^s\}$ in step $s \in \{1, \ldots, t-1\}$, and $\delta_s(k) := 1$ if $C^{IV}_{n_s}$ stands at position k in step s.



These pointers $\boldsymbol{\delta}_s$ have similar properties as $\boldsymbol{\beta}_s$ in Section 3.

Lemma 6.1 *Let $t \in \mathbb{N}$ with $t \geq 2$ be given and $\boldsymbol{\delta}_0 := (2)$. Then*

$$\boldsymbol{\delta}_{s+1} = (\boldsymbol{\delta}_s, \, \boldsymbol{\beta}_s), \qquad s = 0, \ldots, t-2, \tag{6.1}$$

where $\boldsymbol{\beta}_s$ is determined by the recursion (3.5). *Further,*

$$\|\boldsymbol{\delta}_s\|_1 - 2 = \tfrac{1}{3}\, 2^s - \tfrac{1}{2} + \tfrac{1}{6}\, (-1)^s$$

is the number of ones in the triadic vector $\boldsymbol{\delta}_s$.

The proof is similar to that of Lemma 3.4 and omitted here for shortness.

For each pointer $\boldsymbol{\delta}_s$ we define the matrix

$$B_{n+1}(\boldsymbol{\delta}_s) := P_{n_s+1}^T \oplus B_{n_s}(\boldsymbol{\delta}_s(2)) \oplus \ldots \oplus B_{n_s}(\boldsymbol{\delta}_s(2^s)), \qquad s = 0, \ldots, t-2,$$

with $B_{n_s}(0) := T_{n_s}(0)^T$ and $B_{n_s}(1) := T_{n_s}(1)^T$ as in Lemma 2.2 and Lemma 2.4. Further we introduce the matrix

$$U_{n+1}(\boldsymbol{\delta}_s) := T_{n_s+1}(2) \oplus U_{n_s}(\boldsymbol{\delta}_s(2)) \oplus \ldots \oplus U_{n_s}(\boldsymbol{\delta}_s(2^s)), \qquad s = 0, \ldots, t-2,$$

with $U_{n_s}(0) := P_{n_s}$ and $U_{n_s}(1) := A_{n_s}(1)^T P_{n_s}$ as in Lemma 2.2 and Lemma 2.4, and finally the cosine matrix

$$C_{n+1}(\boldsymbol{\delta}_s) := C_{n_s+1}^I \oplus C_{n_s}(\boldsymbol{\delta}_s(2)) \oplus \ldots \oplus C_{n_s}(\boldsymbol{\delta}_s(2^s)), \qquad s = 0, \ldots, t-1,$$

with $C_{n_s}(0) := C_{n_s}^{III}$ and $C_{n_s}(1) := C_{n_s}^{IV}$.
Note that $C_n(\boldsymbol{\delta}_0) = C_{n+1}^I$. We shall see in the following that the cosine matrix $C_{n+1}(\boldsymbol{\delta}_s)$ appears as intermediate result in our recursive factorization of C_{n+1}^I. By construction, all matrices $B_n(\boldsymbol{\delta}_s)$ and $U_n(\boldsymbol{\delta}_s)$ are sparse and orthogonal.

Theorem 6.2 *Let $n = 2^t$ $(t \geq 2)$. Then the matrix C_{n+1}^I can be factorized into the following product of sparse orthogonal matrices*

$$C_{n+1}^I = B_{n+1}(\boldsymbol{\delta}_0) \ldots B_{n+1}(\boldsymbol{\delta}_{t-2})\, C_{n+1}(\boldsymbol{\delta}_{t-1})\, U_{n+1}(\boldsymbol{\delta}_{t-2}) \ldots U_{n+1}(\boldsymbol{\delta}_0). \tag{6.2}$$

The proof directly follows from Lemma 2.2 and Lemma 2.4. The matrix $C_{n+1}(\boldsymbol{\delta}_{t-1})$ in (6.2) is a block matrix consisting only of C_3^I (in the first block), C_2^{III} and C_2^{IV}. The factorization of C_{n+1}^I in Theorem 6.2 implies a fast DCT–I algorithm with

$$\begin{aligned} \alpha(C_{n+1}^I) &= \tfrac{4}{3}\, nt - \tfrac{14}{9}\, n + t + \tfrac{1}{18}(-1)^t + \tfrac{7}{2}, \\ \mu(C_{n+1}^I) &= \tfrac{5}{3}\, nt - \tfrac{22}{9}\, n + t - \tfrac{1}{18}(-1)^t + \tfrac{9}{2}. \end{aligned}$$

We see that DCT–I of length $n+1$ can be computed by $3\, n\, \log_2 n$ flops if n is a power of 2. This split–radix DCT–I algorithm uses only permutations, scaled butterfly operations, and plane rotations/rotation–reflections and works *without* additional scaling. Therefore we obtain a higher arithmetical complexity. For comparison, the FFT–based DCT–I algorithm in [24], pp. 238 – 239 requires $2.5\, n\, \log_2 n$ flops, but it possesses a worse numerical stability, since it requires divisions by small sine values.

7 Numerical stability of split-radix DCT algorithms

In the following we use Wilkinson's standard method for the binary floating point arithmetic for real numbers (see [11], p. 44). If $x \in \mathbb{R}$ is represented by the floating point number $\mathrm{fl}(x)$, then

$$\mathrm{fl}(x) = x\,(1+\delta) \qquad (|\delta| \leq u),$$

where u denotes the *unit roundoff* or *machine precision* as long as we disregard underflow and overflow. For arbitrary $x_0, x_1 \in \mathbb{R}$ and any arithmetical operation $\circ \in \{+, -, \times, /\}$, the exact value $y = x_0 \circ x_1$ and the computed value $\tilde{y} = \mathrm{fl}(x_0 \circ x_1)$ are related by

$$\mathrm{fl}(x_0 \circ x_1) = (x_0 \circ x_1)(1+\delta^\circ) \qquad (|\delta^\circ| \leq u). \tag{7.1}$$

In the IEEE arithmetic of single precision (24 bits for the mantissa with 1 sign bit, 8 bits for the exponent), we have $u \approx 2^{-24} \approx 5.96 \cdot 10^{-8}$. For arithmetic double precision (53 bits for the mantissa with 1 sign bit, 11 bits for the exponent), we have $u \approx 2^{-53} \approx 1.11 \cdot 10^{-16}$ (see [11], p. 45).
Usually the total roundoff error in the result of an algorithm is composed of a number of such errors. To make the origin of relative errors δ_k° clear in this notation, we use superscripts for the operation $\circ$ and subscripts for the operation step k.
In this section we show that, under weak assumptions, a split–radix DCT–II algorithm possesses a remarkable numerical stability. We consider Algorithm 4.2 in detail. The other algorithms can be analysed similarly.

The roundoff errors of Algorithm 4.2 are caused by multiplications with the matrices $S_n(\boldsymbol{\beta}_s)$, $s = 0, \ldots, t-1$, and $A_n(\boldsymbol{\beta}_s)$, $s = 1, \ldots, t-2$. These matrices have a very simple structure. After suitable permutations, every matrix is block–diagonal with blocks of order ≤ 2. All blocks of order 1 are equal to 1. Every block of order 2 is either a (scaled) butterfly matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix},$$

or a scaled rotation matrix/rotation–reflection matrix

$$\begin{pmatrix} a_0 & a_1 \\ -a_1 & a_0 \end{pmatrix}, \quad \begin{pmatrix} a_0 & a_1 \\ a_1 & -a_0 \end{pmatrix}$$

with

$$a_0 = \sqrt{2}\,\cos\frac{(2k+1)\pi}{2^{s+3}}, \quad a_1 = \sqrt{2}\sin\frac{(2k+1)\pi}{2^{s+3}}, \qquad s = 0, \ldots, t-2;\ k = 0, \ldots, 2^s - 1.$$

First we analyse the roundoff errors of simple matrix–vector products, where the matrix of order 2 is a (scaled) butterfly matrix or scaled rotation matrix. Before starting the detailed analysis, we show the following useful estimate.

Lemma 7.1 *For all $a, b, c, d \in \mathbb{R}$, we have*

$$(|ac| + |bd| + |ac - bd|)^2 + (|ad| + |bc| + |ad + bc|)^2 \leq \tfrac{16}{3}(a^2 + b^2)(c^2 + d^2)$$

where the constant $16/3$ is best possible.

Proof. Without loss of generality, we can assume that $a, b, c, d \geq 0$ and $ac \geq bd$. Then the above inequality reads as follows

$$(ac)^2 + (ad + bc)^2 \leq \tfrac{4}{3}(a^2 + b^2)(c^2 + d^2).$$

This inequality is equivalent to

$$0 \leq (ad - bc)^2 + (ac - 2bd)^2$$

and obviously true. For $a = c = \sqrt{2}$ and $b = d = 1$ we have equality. □

Lemma 7.2 (i) *For the butterfly operation* $y_0 := x_0 + x_1$, $y_1 := x_0 - x_1$ *with* $\tilde{y}_0 := \mathrm{fl}(x_0 + x_1)$ *and* $\tilde{y}_1 := \mathrm{fl}(x_0 - x_1)$, *the roundoff error can be estimated by*

$$(\tilde{y}_0 - y_0)^2 + (\tilde{y}_1 - y_1)^2 \leq 2\,u^2\,(x_0^2 + x_1^2). \tag{7.2}$$

(ii) *If the scaling factor* $a \notin \{0, \pm 1\}$ *is precomputed by* $\tilde{a} = a + \Delta a$ *with* $|\Delta a| \leq c_1\,u$, *then for the scaled butterfly operation* $y_0 := a(x_0 + x_1)$, $y_1 := a(x_0 - x_1)$ *with* $\tilde{y}_0 := \mathrm{fl}(\tilde{a}(x_0 + x_1))$ *and* $\tilde{y}_1 := \mathrm{fl}(\tilde{a}(x_0 - x_1))$, *the roundoff error can be estimated by*

$$(\tilde{y}_0 - y_0)^2 + (\tilde{y}_1 - y_1)^2 \leq (2\sqrt{2}\,|a| + \sqrt{2}\,c_1 + \mathcal{O}(u))^2\,u^2\,(x_0^2 + x_1^2). \tag{7.3}$$

(iii) *If the different entries* $a_k \notin \{0, \pm 1\}$ *with* $a^2 := a_0^2 + a_1^2 > 0$ *are precomputed by* $\tilde{a}_k = a_k + \Delta a_k$ *with* $|\Delta a_k| \leq c_2 u$ *for* $k = 0, 1$, *then for the scaled rotation*

$$y_0 := a_0 x_0 + a_1 x_1, \quad y_1 := -a_1 x_0 + a_0 x_1$$

with $\tilde{y}_0 := \mathrm{fl}(\tilde{a}_0 x_0 + \tilde{a}_1 x_1)$, $\tilde{y}_1 := \mathrm{fl}(-\tilde{a}_1 x_0 + \tilde{a}_0 x_1)$, *the roundoff error can be estimated by*

$$(\tilde{y}_0 - y_0)^2 + (\tilde{y}_1 - y_1)^2 \leq (\tfrac{4}{\sqrt{3}}\,|a| + \sqrt{2}\,c_2 + \mathcal{O}(u))^2\,u^2\,(x_0^2 + x_1^2). \tag{7.4}$$

Proof. (i) By (7.1) we have

$$\begin{aligned} \tilde{y}_0 &= (x_0 + x_1)\,(1 + \delta_0^+) = y_0 + (x_0 + x_1)\delta_0^+, \\ \tilde{y}_1 &= (x_0 - x_1)\,(1 + \delta_1^+) = y_1 + (x_0 - x_1)\,\delta_1^+ \end{aligned}$$

with $|\delta_k^+| \leq u$ for $k = 0, 1$ such that by

$$|\tilde{y}_0 - y_0| \leq |x_0 + x_1|\,u, \qquad |\tilde{y}_1 - y_1| \leq |x_0 - x_1|\,u,$$

we obtain (7.2).
(ii) Putting $z_0 := \tilde{a}(x_0 + x_1)$, $z_1 := \tilde{a}(x_0 - x_1)$, it follows from (7.1) that

$$\begin{aligned} \tilde{y}_0 &= \tilde{a}(x_0 + x_1)(1 + \delta_0^+)(1 + \delta_0^\times) = z_0 + \tilde{a}(x_0 + x_1)(\delta_0^+ + \delta_0^\times + \delta_0^+\delta_0^\times), \\ \tilde{y}_1 &= \tilde{a}(x_0 - x_1)(1 + \delta_1^+)(1 + \delta_1^\times) = z_1 + \tilde{a}(x_0 - x_1)(\delta_1^+ + \delta_1^\times + \delta_1^+\delta_1^\times) \end{aligned}$$

with $|\delta_k^+| \leq u$, $|\delta_k^\times| \leq u$, $k = 0, 1$. Thus, by

$$|\tilde{y}_0 - z_0| \leq |\tilde{a}(x_0 + x_1)|(2u + u^2), \qquad |\tilde{y}_1 - z_1| \leq |\tilde{a}(x_0 - x_1)|(2u + u^2)$$

we get the estimate

$$|\tilde{y}_0 - z_0|^2 + |\tilde{y}_1 - z_1|^2 \leq 2\,\tilde{a}^2\,u^2\,(2 + u)^2\,(x_0^2 + x_1^2)$$

with $\tilde{a}^2 = a^2 + \mathcal{O}(u)$, i.e.

$$\left\| \begin{pmatrix} \tilde{y}_0 - z_0 \\ \tilde{y}_1 - z_1 \end{pmatrix} \right\|_2 \leq (2\sqrt{2}\,|a| + \mathcal{O}(u))\,u \left\| \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right\|_2.$$

By

$$\begin{pmatrix} z_0 - y_0 \\ z_1 - y_1 \end{pmatrix} = \Delta a \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

we obtain

$$\left\| \begin{pmatrix} z_0 - y_0 \\ z_1 - y_1 \end{pmatrix} \right\|_2 \leq \sqrt{2}\, c_1\, u \, \left\| \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right\|_2$$

and finally by triangle inequality

$$\left\| \begin{pmatrix} \tilde{y}_0 - y_0 \\ \tilde{y}_1 - y_1 \end{pmatrix} \right\|_2 \leq (2\sqrt{2}\,|a| + \sqrt{2}\, c_1 + \mathcal{O}(u))\, u \, \left\| \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right\|_2 .$$

(iii) Introducing $z_0 := \tilde{a}_0 x_0 + \tilde{a}_1 x_1$, $z_1 := -\tilde{a}_1 x_0 + \tilde{a}_0 x_1$, it follows from (7.1) that

$$\begin{aligned} \tilde{y}_0 &= [\tilde{a}_0 x_0 (1 + \delta_0^{\times}) + \tilde{a}_1 x_1 (1 + \delta_1^{\times})]\,(1 + \delta_0^{+}), \\ \tilde{y}_1 &= [-\tilde{a}_1 x_0 (1 + \delta_2^{\times}) + \tilde{a}_0 x_1 (1 + \delta_3^{\times})]\,(1 + \delta_1^{+}) \end{aligned}$$

with $|\delta_j^{\times}| \leq u$ for $j = 0, \ldots, 3$ and $|\delta_k^{+}| \leq u$ for $k = 0, 1$. Hence we obtain

$$\begin{aligned} |\tilde{y}_0 - z_0| &\leq (|\tilde{a}_0 x_0| + |\tilde{a}_1 x_1| + |\tilde{a}_0 x_0 + \tilde{a}_1 x_1|)\, u + (|\tilde{a}_0 x_0| + |\tilde{a}_1 x_1|)\, u^2, \\ |\tilde{y}_1 - z_1| &\leq (|\tilde{a}_1 x_0| + |\tilde{a}_0 x_1| + |\tilde{a}_1 x_0 - \tilde{a}_0 x_1|)\, u + (|\tilde{a}_1 x_0| + |\tilde{a}_0 x_1|)\, u^2 \end{aligned}$$

and hence

$$\begin{aligned} |\tilde{y}_0 - z_0|^2 + |\tilde{y}_1 - z_1|^2 \;\leq\; & [(|\tilde{a}_0 x_0| + |\tilde{a}_1 x_1| + |\tilde{a}_0 x_0 + \tilde{a}_1 x_1|)^2 \\ & + (|\tilde{a}_1 x_0| + |\tilde{a}_0 x_1| + |\tilde{a}_1 x_0 - \tilde{a}_0 x_1|)^2]\, u^2 (1 + u)^2. \end{aligned}$$

Applying Lemma 7.1, we find

$$\begin{aligned} |\tilde{y}_0 - z_0|^2 + |\tilde{y}_1 - z_1|^2 &\leq \tfrac{16}{3}\,(\tilde{a}_0^2 + \tilde{a}_1^2)(x_0^2 + x_1^2) u^2 (1 + u)^2 \\ &= \tfrac{16}{3}\,(a^2 + \mathcal{O}(u)) u^2 (x_0^2 + x_1^2), \end{aligned}$$

since $\tilde{a}_0^2 + \tilde{a}_1^2 = a_0^2 + a_1^2 + \mathcal{O}(u) = a^2 + \mathcal{O}(u)$ by assumption. Therefore we obtain

$$\left\| \begin{pmatrix} \tilde{y}_0 - z_0 \\ \tilde{y}_1 - z_1 \end{pmatrix} \right\|_2 \leq \left(\tfrac{4}{\sqrt{3}}\,|a| + \mathcal{O}(u) \right)\, u \, \left\| \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right\|_2 .$$

By

$$\begin{pmatrix} z_0 - y_0 \\ z_1 - y_1 \end{pmatrix} = \begin{pmatrix} \Delta a_0 & \Delta a_1 \\ \Delta a_1 & -\Delta a_0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \tag{7.5}$$

we conclude that

$$\left\| \begin{pmatrix} z_0 - y_0 \\ z_1 - y_1 \end{pmatrix} \right\|_2 \leq \sqrt{2}\, c_2\, u \, \left\| \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right\|_2 ,$$

since the matrix in (7.5) is almost orthogonal and therefore its spectral norm equals

$$\sqrt{(\Delta a_0)^2 + (\Delta a_1)^2} \leq \sqrt{2}\, c_2\, u.$$

Using the triangle inequality, we obtain (7.4). □

For an arbitrary input vector $\mathbf{x} \in \mathbb{R}^n$, let $\mathbf{y} := C_n^{II}\mathbf{x} \in \mathbb{R}^n$ be the exact transformed vector. Further let $\tilde{\mathbf{y}} \in \mathbb{R}^n$ be the output vector computed by Algorithm 4.2 using floating point arithmetic with unit roundoff u. Since C_n^{II} is nonsingular, $\tilde{\mathbf{y}}$ can be represented in the

form $\tilde{\mathbf{y}} = C_n^{II}(\mathbf{x} + \Delta\mathbf{x})$ with $\Delta\mathbf{x} \in \mathbb{R}^n$. An algorithm for computing $C_n^{II}\,\mathbf{x}$ is called *normwise backward stable* (see [11], p. 142), if there is a positive constant k_n such that

$$\|\Delta\mathbf{x}\|_2 \leq (k_n u + \mathcal{O}(u^2))\,\|\mathbf{x}\|_2 \tag{7.6}$$

for all vectors $\mathbf{x} \in \mathbb{R}^n$ and $k_n u \ll 1$. The constant k_n measures the quality of numerical stability. Since C_n^{II} is orthogonal, we conclude that $\|\Delta\mathbf{x}\|_2 = \|C_n^{II}(\Delta\mathbf{x})\|_2 = \|\tilde{\mathbf{y}} - \mathbf{y}\|_2$ and $\|\mathbf{x}\|_2 = \|C_n^{II}\mathbf{x}\|_2 = \|\mathbf{y}\|_2$. Hence we also have *normwise forward stability* by

$$\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 \leq (k_n u + \mathcal{O}(u^2))\,\|\mathbf{y}\|_2,$$

if (7.6) is satisfied.
Now let us look closer at the computation steps in Algorithm 4.2. In step 1 of Algorithm 4.2, for every $s = 0, \ldots, t-2$, all values

$$\sqrt{2}\,\cos\frac{(2k+1)\pi}{2^{s+3}}, \qquad \sqrt{2}\,\sin\frac{(2k+1)\pi}{2^{s+3}}, \qquad k = 0, \ldots, 2^s - 1 \tag{7.7}$$

are precomputed. If cosine and sine are internally computed to higher precision and the results afterwards are rounded towards the next machine number, then we obtain very accurate values of (7.7) with an error constant $c_2 = 1$ (see Lemma 7.2, (iii)). We use the matrices $\tilde{S}_n(\boldsymbol{\beta}_s)$, $s = 1, \ldots, t-1$, with precomputed entries (7.7) instead of $S_n(\boldsymbol{\beta}_s) = \sqrt{2}\,T_n(\boldsymbol{\beta}_s)$. Assume that the value $\sqrt{2}/2$ is precomputed with an error constant c_1 (see Lemma 7.2, (ii)). We use the matrices $\tilde{A}_n(\boldsymbol{\beta}_s)$, $s = 1, \ldots, t-2$, with the precomputed scaling factors $\sqrt{2}/2$ instead of $A_n(\boldsymbol{\beta}_s)$.
The vectors $\boldsymbol{\beta}_s$ in Algorithm 4.2, 2. are generated without producing roundoff errors.
Let $\tilde{\mathbf{x}}^{(0)} = \mathbf{x}^{(0)} := \mathbf{x}$. We denote the vectors computed in Algorithm 4.2, 3. by

$$\tilde{\mathbf{x}}^{(s+1)} := \mathrm{fl}(\tilde{S}_n(\boldsymbol{\beta}_s)\,\tilde{\mathbf{x}}^{(s)}), \qquad s = 0, \ldots, t-1.$$

Further, we introduce the error vectors $\mathbf{e}^{(s+1)} \in \mathbb{R}^n$ by

$$\tilde{\mathbf{x}}^{(s+1)} = S_n(\boldsymbol{\beta}_s)\tilde{\mathbf{x}}^{(s)} + \mathbf{e}^{(s+1)}. \tag{7.8}$$

Note that $\mathbf{e}^{(s+1)}$ describes the precomputation error and roundoff error of one step in Algorithm 4.2, 3. The matrix–vector product $\tilde{S}_n(\beta_0)\tilde{\mathbf{x}}^{(0)} = S_n(\beta_0)\mathbf{x}$ involves only butterfly operations such that by Lemma 7.2, (i)

$$\|\mathbf{e}^{(1)}\|_2 \leq \sqrt{2}u\,\|\mathbf{x}\|_2. \tag{7.9}$$

Every matrix–vector product $\tilde{S}_n(\boldsymbol{\beta}_s)\,\tilde{\mathbf{x}}^{(s)}$, $s = 1, \ldots, t-1$, consists of butterfly operations and rotations/rotation–reflections scaled by $\sqrt{2}$ such that by Lemma 7.2, (i) and (iii) we obtain

$$\|\mathbf{e}^{(s+1)}\|_2 \leq (\tfrac{4\sqrt{2}}{\sqrt{3}} + \sqrt{2}\,c_2 + \mathcal{O}(u))\,u\,\|\tilde{\mathbf{x}}^{(s)}\|_2, \qquad s = 1, \ldots, t-1. \tag{7.10}$$

Now we introduce the vectors computed in Algorithm 4.2, 4. by

$$\tilde{\mathbf{x}}^{(t+s+1)} := \mathrm{fl}(P_n(t-s-2)\tilde{A}_n(\boldsymbol{\beta}_{t-s-2})\,\tilde{\mathbf{x}}^{(t+s)}), \qquad s = 0, \ldots, t-2,$$

and the corresponding error vectors $\mathbf{e}^{(t+s+1)} \in \mathbb{R}^n$ by

$$\tilde{\mathbf{x}}^{(t+s+1)} := P_n(t-s-2)A_n(\boldsymbol{\beta}_{t-s-2})\,\tilde{\mathbf{x}}^{(t+s)} + \mathbf{e}^{(t+s+1)}. \tag{7.11}$$

The vector $\mathbf{e}^{(t+s+1)}$ describes the precomputation error and roundoff error of one step in Algorithm 4.2, 4. Permutations do not produce roundoff errors. Every matrix–vector product $\tilde{A}_n(\boldsymbol{\beta}_{t-s-2})\,\tilde{\mathbf{x}}^{(t+s)}$, $s = 0, \ldots, t-3$, consists of identities and scaled butterfly operations (with precomputed scaling factor $\sqrt{2}/2$) such that by Lemma 7.2, (ii) we can estimate

$$\|\mathbf{e}^{(t+s+1)}\|_2 \le (2 + \sqrt{2}\,c_1 + \mathcal{O}(u))\,u\|\tilde{\mathbf{x}}^{(t+s)}\|_2, \qquad s = 0, \ldots, t-3. \tag{7.12}$$

Note that by $\tilde{A}_n(\boldsymbol{\beta}_0) = I_n$ we have $\mathbf{e}^{(2t-1)} = \mathbf{0}$.
The final part of Algorithm 4.2 is the scaling $\mathbf{y} := \frac{1}{\sqrt{n}}\,\mathbf{x}^{(2t-1)}$. Let $\tilde{\mathbf{y}} := \mathrm{fl}(\frac{1}{\sqrt{n}}\,\tilde{\mathbf{x}}^{(2t-1)})$. By (7.1) we find the estimate

$$\begin{aligned} \|\tilde{\mathbf{y}} - \mathbf{y}\|_2 &\le \|\tilde{\mathbf{y}} - \tfrac{1}{\sqrt{n}}\,\tilde{\mathbf{x}}^{(2t-1)}\|_2 + \tfrac{1}{\sqrt{n}}\,\|\tilde{\mathbf{x}}^{(2t-1)} - \mathbf{x}^{(2t-1)}\|_2 \\ &\le \tfrac{u}{\sqrt{n}}\,\|\tilde{\mathbf{x}}^{(2t-1)}\|_2 + \tfrac{1}{\sqrt{n}}\,\|\tilde{\mathbf{x}}^{(2t-1)} - \mathbf{x}^{(2t-1)}\|_2. \end{aligned} \tag{7.13}$$

We are now ready to estimate the total roundoff error $\|\tilde{\mathbf{y}} - \mathbf{y}\|_2$ of Algorithm 4.2 under the assumption that $\sqrt{2}/2$ and the trigonometric values (7.7) in the factor matrices are precomputed with error bounds $c_1 u$ and $c_2 u$.

Theorem 7.3 *Let $n = 2^t$ $(t \ge 3)$. Assume that $\sqrt{2}/2$ is precomputed with absolute error bound $c_1 u$ and the values in (7.7) are precomputed with absolute error bound $c_2 u$, respectively. Then the split–radix Algorithm 4.2 is normwise backward stable with the constant*

$$k_n = \left(\tfrac{4}{\sqrt{3}} + 2 + \sqrt{2}c_1 + c_2\right)(\log_2 n - 1) - \sqrt{2}\,c_1.$$

Proof. First we estimate the roundoff error $\|\tilde{\mathbf{x}}^{(2t-1)} - \mathbf{x}^{(2t-1)}\|_2$. Applying (7.8) and (7.11) repeatedly, we obtain

$$\begin{aligned} \tilde{\mathbf{x}}^{(2t-1)} = \mathbf{x}^{(2t-1)} &+ P_n(0)A_n(\boldsymbol{\beta}_0)\ldots P_n(t-2)A_n(\boldsymbol{\beta}_{t-2})S_n(\boldsymbol{\beta}_{t-1})\ldots S_n(\boldsymbol{\beta}_1)\mathbf{e}^{(1)} + \ldots \\ &+ P_n(0)A_n(\boldsymbol{\beta}_0)\ldots P_n(t-2)A_n(\boldsymbol{\beta}_{t-2})S_n(\boldsymbol{\beta}_{t-1})\mathbf{e}^{(t-1)} \\ &+ P_n(0)A_n(\boldsymbol{\beta}_0)\ldots P_n(t-2)A_n(\boldsymbol{\beta}_{t-2})\mathbf{e}^{(t)} + \ldots \\ &+ P_n(0)A_n(\boldsymbol{\beta}_0)\mathbf{e}^{(2t-2)}. \end{aligned} \tag{7.14}$$

The matrices $S_n(\boldsymbol{\beta}_s)$, $s = 0, \ldots, t-1$, are almost orthogonal with the spectral norm $\|S_n(\boldsymbol{\beta}_s)\|_2 = \sqrt{2}$. The matrices $P_n(s)\,A_n(\boldsymbol{\beta}_s)$, $s = 0, \ldots, t-2$, are orthogonal such that $\|P_n(s)\,A_n(\boldsymbol{\beta}_s)\|_2 = 1$. By (7.8) and (7.11) we can estimate

$$\begin{aligned} \|\tilde{\mathbf{x}}^{(s+1)}\| &\le \sqrt{2}\,\|\tilde{\mathbf{x}}^{(s)}\|_2 + \|\mathbf{e}^{(s+1)}\|_2, \qquad s = 0, \ldots, t-1, \\ \|\tilde{\mathbf{x}}^{(t+s+1)}\| &\le \|\tilde{\mathbf{x}}^{(t+s)}\|_2 + \|\mathbf{e}^{(t+s+1)}\|_2, \qquad s = 0, \ldots, t-2. \end{aligned}$$

Thus by (7.10) and (7.12) we see that

$$\begin{aligned} \|\tilde{\mathbf{x}}^{(s+1)}\| &\le (\sqrt{2} + \mathcal{O}(u))\,\|\tilde{\mathbf{x}}^{(s)}\|_2, \qquad s = 0, \ldots, t-1, \\ \|\tilde{\mathbf{x}}^{(t+s+1)}\| &\le (1 + \mathcal{O}(u))\,\|\tilde{\mathbf{x}}^{(t+s)}\|_2, \qquad s = 0, \ldots, t-2. \end{aligned}$$

Since $\tilde{\mathbf{x}}^{(0)} = \mathbf{x}$ this implies

$$\|\tilde{\mathbf{x}}^{(s+1)}\| \le (2^{s/2} + \mathcal{O}(u))\,\|\mathbf{x}\|_2, \qquad s = 0, \ldots, t-1, \tag{7.15}$$

$$\|\tilde{\mathbf{x}}^{(t+s+1)}\| \le (2^{t/2} + \mathcal{O}(u))\,\|\mathbf{x}\|_2, \qquad s = 0, \ldots, t-2. \tag{7.16}$$

From (7.10), (7.12), (7.15), and (7.16) it follows that

$$\begin{aligned} \|\mathbf{e}^{(s+1)}\|_2 &\leq 2^{s/2}(\tfrac{4}{\sqrt{3}} + c_2 + \mathcal{O}(u))\, u\, \|\mathbf{x}\|_2, \qquad s = 1, \ldots, t-1, \\ \|\mathbf{e}^{(t+s+1)}\|_2 &\leq 2^{t/2}(2 + \sqrt{2}\, c_1 + \mathcal{O}(u))\, u\, \|\mathbf{x}\|_2, \qquad s = 0, \ldots, t-3. \end{aligned}$$

We obtain from (7.14) that

$$\begin{aligned} \|\tilde{\mathbf{x}}^{(2t-1)} - \mathbf{x}^{(2t-1)}\|_2 &\leq \|S_n(\boldsymbol{\beta}_{t-1}) \ldots S_n(\boldsymbol{\beta}_1)\|_2 \|\mathbf{e}^{(1)}\| + \ldots \|S_n(\boldsymbol{\beta}_{t-1})\|_2 \|\mathbf{e}^{(t-1)}\|_2 \\ &\quad + \|\mathbf{e}^{(t)}\|_2 + \ldots + \|\mathbf{e}^{(2t-2)}\|_2 \\ &\leq (\sqrt{2})^{t-1} \|\mathbf{e}^{(1)}\|_2 + \ldots + \sqrt{2}\, \|\mathbf{e}^{(t-1)}\|_2 + \|\mathbf{e}^{(t)}\|_2 + \ldots + \|\mathbf{e}^{(2t-2)}\|_2 \end{aligned}$$

and hence by (7.9)

$$\begin{aligned} &\|\tilde{\mathbf{x}}^{(2t-1)} - \mathbf{x}^{(2t-1)}\|_2 \\ &\qquad \leq 2^{t/2}\, u \left((\frac{4}{\sqrt{3}} + 2 + \sqrt{2}\, c_1 + c_2)(t-1) - 1 - \sqrt{2}\, c_1 + \mathcal{O}(u) \right) \|\mathbf{x}\|_2. \end{aligned} \tag{7.17}$$

The final part of Algorithm 4.2 is the scaling $\mathbf{y} = 2^{-t/2}\, \mathbf{x}^{(2t-1)}$. Let $\tilde{\mathbf{y}} = \mathrm{fl}(2^{-t/2}\, \tilde{\mathbf{x}}^{(2t-1)})$. For even t this does not produce a additional roundoff error. By (7.13), (7.16) and (7.17) we get the final estimate

$$\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 \leq u \, \left((\tfrac{4}{\sqrt{3}} + 2 + \sqrt{2}\, c_1 + c_2)(t-1) - \sqrt{2}\, c_1 + \mathcal{O}(u) \right) \|\mathbf{x}\|_2.$$

This completes the proof. □

Remark 7.4 *In [2], it has been shown that for computing* $\mathbf{y} = C_n^{II}\mathbf{x}$ *one has normwise backward stability with*
(i) $k_n = \sqrt{2}\, n^{3/2}$ *for classical matrix–vector computation,*
(ii) $k_n = (4\sqrt{2} + 2) \log_2 n + \sqrt{2}$ *for an* FFT*–based* DCT–II *algorithm, and*
(iii) $k_n = (2\sqrt{3})(n-1)$ *for a real fast algorithm based on polynomial arithmetic.*
In these algorithms the nontrivial entries of the factor matrices were assumed to be precomputed exactly. As shown in Theorem 7.3*, the new Algorithm* 4.2 *is extremely stable with a constant which is comparable with the constant for the* FFT*–based* DCT–II *algorithm.*

Remark 7.5 *Considering the numerical stability of Algorithm* 4.4*, we obtain a constant* $k_n = \mathcal{O}(\sqrt{n}\, \log_2 n)$*. Hence Algorithm* 4.4 *is less stable than Algorithm* 4.2*. This is due to the fact that all matrices* $A'_n(\boldsymbol{\beta}_s)$ *and* $S'_n(\boldsymbol{\beta}_s)$*,* $s = 1, \ldots, t-2$*, as well as* $S_n(\boldsymbol{\beta}_{t-1})$ *are not longer orthogonal, but have the spectral norm* $\sqrt{2}$*.*

References

[1] H. Ahmed, T. Natarajan, and K. R. Rao, Discrete cosine transform, IEEE Trans. Comput. **23** (1974), 90 – 93.

[2] G. Baszenski, U. Schreiber, and M. Tasche, Numerical stability of fast cosine transforms, Numer. Funct. Anal. Optim. **21** (2000), 25 – 46.

[3] V. Britanak, A unified discrete cosine and discrete sine transform computation, Signal Process. **43** (1995), 333 – 339.

[4] W. H. Chen, C. H. Smith, and S. Fralick, A fast computational algorithm for the discrete cosine transform, IEEE Trans. Comm. **25** (1977), 1004 – 1009.

[5] I. Daubechies and W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. **4** (1998), 247 – 269.

[6] P. Duhamel and H. Hollmann, Split radix FFT algorithms, Electron. Lett. **20** (1984), 14 – 16.

[7] P. Duhamel, Implementation of the split–radix FFT algorithms for complex, real, and real–symmetric data, IEEE Trans. Acoust. Speech Signal Process. **34** (1986), 285 – 295.

[8] E. Feig, A scaled DCT algorithm, Proc. SPIE **1244** (1990), 2 – 13.

[9] E. Feig and S. Winograd, Fast algorithms for the discrete cosine transform, IEEE Trans. Signal Process. **40** (1992), 2174 – 2193.

[10] M. T. Heideman, D. H. Johnson, and C. S. Burrus, Gauss and the history of the fast Fourier transform, Arch. Hist. Exact Sci. **34** (1985), 265 – 277.

[11] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

[12] H. S. Hou, A fast recursive algorithm for computing the discrete cosine transform, IEEE Trans. Acoust. Speech Signal Process. **35** (1987), 1455 – 1461.

[13] B. Lee, A new algorithm to compute the discrete cosine transform, IEEE Trans. Acoust. Speech Signal Process. **32** (1984), 1243 – 1245.

[14] L. Loeffler, A. Ligtenberg, and G. S. Moschytz, Practicle fast 1–d DCT algorithms with 11 multiplications, Proc. IEEE ICASSP (1989), 989 – 991.

[15] M. J. Narasimha and A. M. Peterson, On computing the discrete cosine transform, IEEE Trans. Commun. **26** (1978), 934 – 936.

[16] M. Püschel and J. M. Moura, The algebraic approach to the discrete cosine and sine transforms and their fast algorithms, Preprint, Carnegie Mellon Univ., Pittsburgh, 2001.

[17] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Boston, 1990.

[18] C. Runge, On the decomposition of empirically given periodic functions into sine waves (in German), Z. Math. Phys. **48** (1903), 443 – 456, and **52** (1905), 117 – 123.

[19] U. Schreiber, Fast and numerically stable trigonometric transforms (in German), Thesis, Univ. of Rostock, 1999.

[20] G. Steidl, Fast radix–p discrete cosine transform, Appl. Algebra Engrg. Comm. Comput. **3** (1992), 39 – 46.

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

[22] G. Strang, The discrete cosine transform, SIAM Rev. **41** (1999), 135 – 147.

[23] M. Tasche and H. Zeuner, Roundoff error analysis for fast trigonometric transforms, in: *Handbook of Analytic–Computational Methods in Applied Mathematics*, G. Anastassiou (ed.), Chapman & Hall/CRC, Boca Rota, 2000, pp. 357 – 406.

[24] C. F. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[25] M. Vetterli and H. J. Nussbaumer, Simple FFT and DCT algorithms with reduced number of operations, Signal Process. **6** (1984), 267 – 278.

[26] Z. Wang, Fast algorithms for the discrete W transform and the discrete Fourier transform, IEEE Trans. Acoust. Speech Signal Process. **32** (1984), 803 – 816.

[27] H. Zeuner, A general theory of stochastic roundoff error analysis with applications to DFT and DCT, J. Comput. Anal. Appl., to appear.

INSTITUTE OF MATHEMATICS, GERHARD–MERCATOR–UNIVERSITY OF DUISBURG,
D – 47048 DUISBURG, GERMANY

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF ROSTOCK,
D – 18051 ROSTOCK, GERMANY

E–*mail addresses*: plonka@math.uni-duisburg.de
manfred.tasche@mathematik.uni-rostock.de