Fast and stable algorithms for discrete spherical Fourier transforms

Daniel Potts, Gabriele Steidl, and Manfred Tasche

Abstract. In this paper, we propose an algorithm for the stable and efficient computation of Fourier expansions of square integrable functions on the unit sphere $S \subset \mathbb{R}^3$, as well as for the evaluation of these Fourier expansions at special knots. The heart of the algorithm is an efficient realization of discrete Legendre function transforms based on a modified and stabilized version of the Driscoll–Healy algorithm.

1991 *Mathematics Subject Classification.* Primary 65T99, 33C35, 33C25, 42C10
Key words and phrases. Discrete spherical Fourier transform, spherical harmonics, sampling theorem, discrete Legendre function transform, fast cosine transform, Chebyshev nodes, cascade summation

1 Introduction

Fourier analysis on the sphere $S \subset \mathbb{R}^3$ has practical relevance in tomography, geophysics, seismology, meteorology and crystallography. It can be used in spectral methods for solving partial differential equations on the sphere (see [4], [16]). In [12], the authors utilize spherical Fourier transforms for the decomposition and reconstruction of functions defined on the sphere with respect to spherical frames.
In this paper, we propose an algorithm for the efficient and stable computation of Fourier expansions of square integrable functions on S. The Fourier expansion of a function $f \in L^2(S)$ with band–width N is given by

$$f = \sum_{k=0}^{N} \sum_{n=-k}^{k} (2k+1)\, a_k^n(f)\, Y_k^n \ ,$$

where $a_k^n(f)$ are the Fourier coefficients of f with respect to the orthogonal basis of spherical harmonics Y_k^n. To compute these Fourier coefficients we first prove a sampling theorem which is based on Clenshaw–Curtis quadrature and restricts the evaluation of the integrals $a_k^n(f)$ to the computation of discrete spherical Fourier transforms.
Note that for $N \leq 360$ there exist numerous realizations of discrete spherical Fourier transforms. For the interesting case $N > 360$ we can refer only to [9].
Since the spherical harmonics $Y_k^n(\theta, \varphi)$ are scaled products of complex exponentials $\mathrm{e}^{\mathrm{i}n\varphi}$ and Legendre functions $P_k^{|n|}(\cos\theta)$, the discrete spherical Fourier transform splits into ordinary discrete Fourier transforms for complex exponentials, which can be realized by fast Fourier transform techniques, and discrete Legendre function transforms. The main part of this paper deals with an efficient and stable algorithm for these discrete Legendre function transforms.
For $n = 0$, i.e. in the case of the discrete Legendre transform, we apply the algorithm for the fast polynomial transform introduced in [13]. This algorithm with an arithmetic complexity of $\mathcal{O}(N \log^2 N)$ can be considered as a modified version of the transposed Driscoll–Healy algorithm [6] in which the original fast Fourier transforms were replaced by fast cosine transforms. Due to the consequent application of polynomial arithmetic and cascade summation, our approach seems to be simpler and more straightforward than the original Driscoll–Healy algorithm. By convenient cascade summation a fast polynomial transform can be modified for a fast Legendre function transform (FLFT). Unfortunately, its implementation demonstrates numerical instability for large $n > 16$. The reason for this is that some of the so–called

associated functions $P_k^n(x,c)$ involved in the algorithm become very large for $|x| \approx 1$ while other functions $P_k^n(x,c)$ become relatively small. The multiplications of these large and small values result in unacceptable roundoff errors. To avoid this negative effect, we introduce special stabilization steps in the algorithm. This heuristic stabilization method can be compared with the so–called method of "stable bypass operations" in [9], which was used to stabilize the Driscoll–Healy algorithm on the sphere. The introduction of the exceptional steps closes the gain in the stability of the algorithm at the expense of a loss of runtime efficiency. However, for $n \le 3N/4$, our algorithm performs faster than the Clenshaw algorithm. For $n \approx N/2$, we need half of the CPU-time of the Clenshaw algorithm.
This paper is organized as follows: First we present a sampling theorem for band–limited functions $f \in L^2(S)$. Taking into account that our algorithm for the efficient discrete spherical Fourier transform is mainly based on fast realizations of discrete cosine transforms, Section 3 deals with discrete cosine transforms. A fast algorithm for the discrete Legendre function transform is described in Section 4. Finally, Section 5 contains a stabilized version of the algorithm and numerical results.

2 Band–limited functions on S

Starting with the *Legendre polynomials*

$$P_k(x) := \frac{1}{2^k\,k!}\,\frac{\mathrm{d}^k}{\mathrm{d}x^k}(x^2-1)^k \quad (x \in [-1,1];\ k \in \mathbb{N}_0)\,,$$

we define the *associated Legendre functions* P_k^n $(n \in \mathbb{N}_0;\ k = n, n+1, \ldots)$ by

$$P_k^n(x) := \left(\frac{(k-n)!}{(k+n)!}\right)^{1/2} (1-x^2)^{n/2}\,\frac{\mathrm{d}^n}{\mathrm{d}x^n} P_k(x) \quad (x \in [-1,1])\,.$$

For any fixed $n \in \mathbb{N}_0$, the functions P_k^n $(k = n, n+1, \ldots)$ form a complete orthogonal system in $L^2[-1,1]$ with

$$\frac{1}{2}\int_{-1}^{1} P_k^n(x) P_l^n(x)\,\mathrm{d}x \;=\; \frac{1}{2k+1}\,\delta_{k,l} \quad (n \in \mathbb{N}_0;\ k,l = n, n+1, \ldots)\,.$$

Moreover, the associated Legendre functions fulfil the three–term recurrence relation

$$P_{n-1}^n(x) := 0, \qquad P_n^n(x) := \frac{\left((2n)!\right)^{1/2}}{2^n\,n!}\,(1-x^2)^{n/2}\,,$$

$$P_{k+1}^n(x) = v_k^n \; xP_k^n(x) + w_k^n \; P_{k-1}^n(x) \quad (k = n, n+1, \ldots) \tag{2.1}$$

with

$$v_k^n := \frac{2k+1}{((k-n+1)(k+n+1))^{1/2}}, \qquad w_k^n := -\frac{((k-n)(k+n))^{1/2}}{((k-n+1)(k+n+1))^{1/2}}.$$

We are interested in the Hilbert space $L^2(S)$ of all square integrable functions on the 2–sphere S with the scalar product

$$\langle f, g\rangle := \frac{1}{4\pi} \int_0^\pi \int_0^{2\pi} f(\theta,\varphi)\,\overline{g(\theta,\varphi)}\sin\theta \; \mathrm{d}\varphi \; \mathrm{d}\theta \quad (f,\, g \in L^2(S))$$

and with the corresponding norm $\|\cdot\|$. Set

$$\hat{\mathrm{I}} := \{(k,n) :\; k \in \mathbb{N}_0,\; n = -k, \ldots, k\}.$$

An orthogonal basis of $L^2(S)$ is given by the set $\{Y_k^n :\; (k,n) \in \hat{\mathrm{I}}\}$ of *spherical harmonics*

$$Y_k^n(\theta,\varphi) := P_k^{|n|}(\cos\theta)\,\mathrm{e}^{\mathrm{i}n\varphi}.$$

It is easy to check that

$$\begin{aligned}\langle Y_k^n, Y_l^m\rangle &= \frac{1}{2}\int_{-1}^1 P_k^{|n|}(x)P_l^{|m|}(x)\;\mathrm{d}x \;\cdot\; \frac{1}{2\pi}\int_0^{2\pi} \mathrm{e}^{\mathrm{i}(n-m)\varphi}\;\mathrm{d}\varphi \\ &= \frac{1}{2k+1}\,\delta_{k,l}\,\delta_{n,m} \quad ((k,n),\;(l,m) \in \hat{\mathrm{I}}).\end{aligned}$$

We consider the Fourier expansion of $f \in L^2(S)$ with respect to the spherical harmonics

$$f = \sum_{(k,n)\in\hat{\mathrm{I}}} (2k+1)\,a_k^n(f)\,Y_k^n, \quad a_k^n(f) := \langle f, Y_k^n\rangle.$$

Let $l^2(\hat{\mathrm{I}})$ be the Hilbert space of all complex sequences $(a_k^n)_{(k,n)\in\hat{\mathrm{I}}}$ with

$$\sum_{(k,n)\in\hat{\mathrm{I}}} (2k+1)\,|a_k^n|^2 < \infty.$$

Then we refer to $F :\; L^2(S) \to l^2(\hat{\mathrm{I}})$ defined by

$$F\,f := (\hat{a}_k^n(f))_{(k,n)\in\hat{\mathrm{I}}} \quad (f \in L^2(S))$$

as *spherical Fourier transform.*

For $j \in \mathbb{N}_0$ we set

$$\begin{aligned} \mathrm{I}^j &:= \{(s,t) : s = 0,\ldots,2^{j+1};\ \ t = 0,\ldots,2^{j+1}-1\}, \\ \hat{\mathrm{I}}^j &:= \{(k,n) : k = 0,\ldots,2^j;\ n = -k,\ldots,k\}. \end{aligned}$$

We consider the *sampling spaces of level* j ($j \in \mathbb{N}_0$)

$$V^j := \mathrm{span}\,\{Y_k^n :\ (k,n) \in \hat{\mathrm{I}}^j\}$$

consisting of so–called *band–limited* functions of band–width 2^j. For band–limited functions, it holds the following sampling theorem:

Theorem 2.1. *Let* $f \in V^j$ $(j \in \mathbb{N}_0)$ *with*

$$f = \sum_{(k,n)\in\hat{\mathrm{I}}^j} \hat{a}_k^n\, Y_k^n \tag{2.2}$$

be given. Then we have for $(k,n) \in \hat{\mathrm{I}}^j$

$$\hat{a}_k^n = \frac{1}{2^{j+1}} \sum_{(s,t)\in\mathrm{I}^j} \varepsilon_s^{(j+1)} w_s^{(j+1)} f(p_{s,t}^j)\, Y_k^{-n}(p_{s,t}^j)$$

with

$$p_{s,t}^j := \left(\frac{\pi s}{2^{j+1}}, \frac{\pi t}{2^j}\right),$$

$$\varepsilon_0^{(j)} = \varepsilon_{2^j}^{(j)} := 2^{-1}, \quad \varepsilon_s^{(j)} := 1 \ \ (s = 1,\ldots,2^j-1)$$

and with the Clenshaw–Curtis weights

$$w_s^{(j+1)} := \frac{1}{2^{j+1}} \sum_{u=0}^{2^j} \varepsilon_u^{(j)} \frac{-2}{4u^2-1} \cos\frac{su\pi}{2^j} \quad (s = 0,\ldots,2^{j+1}).$$

Proof: By definition of V^j, it suffices to consider the functions $f(\theta,\varphi) = Y_l^m(\theta,\varphi)$ $((l,m) \in \hat{\mathrm{I}}^j)$. Their Fourier coefficients can be written as

$$a_k^n(f) = \frac{1}{2}\int_{-1}^{1} P_l^{|m|}(x) P_k^{|n|}(x)\,\mathrm{d}x \cdot \frac{1}{2\pi}\int_0^{2\pi} \mathrm{e}^{\mathrm{i}(m-n)\varphi}\,\mathrm{d}\varphi\,. \tag{2.3}$$

Now it holds for m, $n = -2^j,\ldots,2^j$ that

$$\delta_{m,n} = \frac{1}{2\pi}\int_0^{2\pi} \mathrm{e}^{\mathrm{i}(m-n)\varphi}\,\mathrm{d}\varphi = \frac{1}{2^{j+1}} \sum_{t=0}^{2^{j+1}-1} \mathrm{e}^{\pi\mathrm{i}(m-n)t/2^j}\,. \tag{2.4}$$

Hence, for $m \neq n$, we are done. For $m = n$, we verify that $P_l^{|n|} P_k^{|n|}$ is an algebraic polynomial of degree $\leq 2^{j+1}$ such that Clenshaw–Curtis quadrature gives

$$\frac{1}{2} \int_{-1}^{1} P_l^{|n|}(x) P_k^{|n|}(x) \, \mathrm{d}x = \sum_{s=0}^{2^{j+1}} \varepsilon_s^{(j+1)} \, w_s^{(j+1)} P_l^{|n|} \left(c_s^{2^{j+1}}\right) P_k^{|n|} \left(c_s^{2^{j+1}}\right)$$

with the *Chebyshev nodes* $c_s^N := \cos \frac{\pi s}{N}$. Together with (2.3) and (2.4) this completes the proof. ∎

Note that a similar sampling theorem for band–limited functions on S was proved in [6].

We are interested in the efficient solution of the following two problems:
1. For given values $f(p_{s,t}^j)$ $((s,t) \in \mathrm{I}^j)$ of $f \in V^j$ compute the Fourier–coefficients $\hat{a}_k^n \in \mathbb{C}$ $((k,n) \in \hat{\mathrm{I}}^j)$ given in Theorem 2.1.
2. For given Fourier–coefficients $\hat{a}_k^n \in \mathbb{C}$ $((k,n) \in \hat{\mathrm{I}}^j)$ compute the values $f(p_{s,t}^j)$ $((s,t) \in \mathrm{I}^j)$ of f defined by (2.2).

By definition of Y_k^n and by Theorem 2.1, we suggest the following algorithms for 1. and 2., respectively.

Algorithm 2.2 (Discrete spherical Fourier transform)
Input: For fixed $j \in \mathbb{N}_0$, let $f(p_{s,t}^j) \in \mathbb{C}$ $((s,t) \in \mathrm{I}^j)$ be given.

1. For every $s = 0, \ldots, 2^{j+1}$ form by fast Fourier transform

$$f_{s,n}^j := \frac{1}{2^{j+1}} \sum_{t=0}^{2^{j+1}-1} f(p_{s,t}^j) \, \mathrm{e}^{-\mathrm{i} n \pi t / 2^j} \quad (n = -2^j, \ldots, 2^j) \, . \tag{2.5}$$

2. For every $n = -2^j, \ldots, 2^j$ compute

$$\hat{a}_k^n := \sum_{s=0}^{2^{j+1}} \varepsilon_s^{(j+1)} w_s^{(j+1)} \, f_{s,n}^j \, P_k^{|n|} \left(c_s^{2^{j+1}}\right) \quad (k = |n|, \ldots, 2^j) \, . \tag{2.6}$$

Output: $\hat{a}_k^n \in \mathbb{C}$ $((k,n) \in \hat{\mathrm{I}}^j)$.

Algorithm 2.3 (Inverse discrete spherical Fourier transform)
Input: For fixed $j \in \mathbb{N}_0$, let $\hat{a}_k^n \in \mathbb{C}$ $((k,n) \in \hat{\mathrm{I}}^j)$ be given.

1. For every $n = -2^j, \ldots, 2^j$ compute

$$g^j_{s,n} := \sum_{k=|n|}^{2^j} \hat{a}^n_k \, P^{|n|}_k \, (c^{2^{j+1}}_s) \quad (s = 0, \ldots, 2^{j+1}) \,. \tag{2.7}$$

2. For every $s = 0, \ldots, 2^{j+1}$ form by fast Fourier transform

$$f(p^j_{s,t}) := \sum_{n=-2^j}^{2^j} g^j_{s,n} \, \mathrm{e}^{\mathrm{i}n\pi t/2^j} \quad (t = 0, \ldots, 2^{j+1} - 1) \,. \tag{2.8}$$

Output: $f(p^j_{s,t}) \in \mathbb{C}$ $((s,t) \in \mathrm{I}^j)$.

The fast Fourier transforms in (2.5) and (2.8) require $\mathcal{O}((j+1)2^{j+1})$ arithmetical operations for fixed n. Hence, it remains to develop an algorithm for the fast evaluation of (2.6) and (2.7), respectively.
A fast algorithm for (2.7) implies the factorization of the transform matrix

$$\mathbf{P}^{|n|} := \left(P^{|n|}_k (c^{2^{j+1}}_s) \right)_{s=0,k=|n|}^{2^{j+1},2^j}$$

into a product of sparse matrices. Consequently, once a fast algorithm for (2.7) is known, a fast algorithm for the "inverse" problem (2.6) with the transform matrix $(\mathbf{P}^{|n|})^{\mathrm{T}}$ is available too by transposing the sparse matrix product. Therefore, we restrict our attention to the fast computation of (2.7).

3 Discrete cosine transforms

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

$$\begin{aligned} \mathbf{C}_{N+1} &:= \left(c^N_{jk}\right)^N_{j,k=0} \,, & \mathbf{D}_{N+1} &:= \operatorname{diag}\left(\varepsilon^N_j\right)^N_{j=0} \,, \\ \tilde{\mathbf{C}}_N &:= \left(c^{2N}_{j(2k+1)}\right)^{N-1}_{j,k=0} \,, & \tilde{\mathbf{D}}_N &:= \operatorname{diag}\left(\varepsilon^N_j\right)^{N-1}_{j=0} \end{aligned}$$

with $\varepsilon^N_0 = \varepsilon^N_N := 1/2$, $\varepsilon^N_k := 1$ $(k = 1, \ldots, N-1)$ be given. Then the following transforms are referred as *discrete cosine transforms* (DCT) *of type* I – III, respectively:

DCT–I$\,(N+1)$: $\mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$ 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$\,(N)$: $\mathbb{R}^N \to \mathbb{R}^N$ 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$\,(N)$: $\mathbb{R}^N \to \mathbb{R}^N$ with

$$\hat{\mathbf{b}} \;:=\; \tilde{\mathbf{C}}_N^{\mathrm{T}}\,\tilde{\mathbf{D}}_N\mathbf{b}\,,$$

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 $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$ arithmetical operations. For DCT–III and DCT–II we prefer the fast algorithms in [14] because of their low arithmetical complexity and since the corresponding data permutations allow a simple, efficient implementation (see [9]). Fast algorithms for DCT–I based on [14] can be found in [2] (see also [15]). Concerning the inverse DCT's, it is easy to check (see [2]):

Lemma 3.1. *It holds that*

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

Hence $(\mathbf{C}_{N+1}\mathbf{D}_{N+1})^{-1} = \frac{2}{N}\,\mathbf{C}_{N+1}\mathbf{D}_{N+1}$ and $(\tilde{\mathbf{C}}_N)^{-1} = \frac{2}{N}\,\tilde{\mathbf{C}}_N^{\mathrm{T}}\tilde{\mathbf{D}}_N$ 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 $j = 0, \ldots, M-1$, where $M = 2^s$ $(s \in \mathbb{N})$ with $M/2 \le m + n < M$ is chosen. Then the Chebyshev coefficients b_k $(k = 0, \ldots, m+n)$ in

$$R \;:=\; P\,Q \;=\; \sum_{k=0}^{n+m} b_k\, T_k$$

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

Algorithm 3.2 (Fast polynomial multiplication)

Input: $M = 2^s$ $(s \in \mathbb{N})$ with $M/2 \le m + n < M$,
$Q(c_{2j+1}^{2M}) \in \mathbb{R}$ $(j = 0, \ldots, M-1)$ with $Q \in \Pi_m$,
$a_k \in \mathbb{R}$ $(k = 0, \ldots, n)$.

1. Compute
$$(P(c_{2j+1}^{2M}))_{j=0}^{M-1} \;:=\; \tilde{\mathbf{C}}_M^{\mathrm{T}}\,(a_k)_{k=0}^{M-1}$$
by fast DCT–III (M) of $(a_k)_{k=0}^{M-1}$ with $a_k := 0$ $(k = n+1, \ldots, M-1)$.
2. Evaluate the M products
$$R(c_{2j+1}^{2M}) \;:=\; P(c_{2j+1}^{2M})\,Q(c_{2j+1}^{2M}) \quad (j = 0, \ldots, M-1)\,.$$
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 $(k = 0, \ldots, m+n)$.

The fast DCT–III (2^s) computed by [14] 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.

4 Fast Legendre function transforms

The simplest realization of (2.7) by the Clenshaw algorithm utilizes the three–term recurrence relation (2.1) and requires $\mathcal{O}((2^j - n)2^j)$ arithmetical operations for fixed n. In the sequel, we propose a new algorithm for the evaluation of (2.7) which is faster than Clenshaw's algorithm for sufficiently large $2^j - n$. Especially, for $n \le 16$ it requires only $\mathcal{O}(2^j j^2)$ arithmetical operations. We restrict our attention to even $n \in \mathbb{N}$. Then

$$P \;:=\; \sum_{k=n}^{2^j} \hat{a}_k^n \; P_k^n \tag{4.1}$$

is a polynomial of degree $\le 2^j$ which must be evaluated at the Chebyshev nodes $c_s^{2^{j+1}}$ $(s = 0, \ldots, 2^{j+1})$. If n is odd, then

$$P(x) = (1 - x^2)^{1/2}\, Q(x)$$

with the polynomial

$$Q(x) = \sum_{k=n}^{2^j} \hat{a}_k^n \; (1-x^2)^{\frac{n-1}{2}} \; \frac{\mathrm{d}^n}{\mathrm{d}x^n} \; P_k(x) \; \in \Pi_{2^j-1}$$

of degree $\le 2^j - 1$. Here our algorithm evaluates $Q(c_s^{2^{j+1}})$ $(s = 0, \ldots, 2^{j+1})$ in a similar manner as for even n. Finally, we have to compute $P(c_s^{2^{j+1}}) = Q(c_s^{2^{j+1}}) \sin \frac{\pi s}{2^{j+1}}$.

In the following, we develop an algorithm for the fast evaluation of (4.1) at the Chebyshev nodes for fixed *even* $n \in \mathbb{N}_0$. We introduce the polynomials $\tilde{P}_k^n$ by

$$\tilde{P}_{-1}^n(x) := 0, \qquad \tilde{P}_0^n(x) := \frac{\left((2n)!\right)^{1/2}}{2^n \, n!} \,,$$

$$\tilde{P}_{k+1}^n(x) := (\alpha_k^n \, x + \beta_k^n) \; \tilde{P}_k^n(x) + \gamma_k^n \; \tilde{P}_{k-1}^n(x) \quad (k = 0, 1, \ldots) \tag{4.2}$$

with

$$\alpha_k^n := \begin{cases} (-1)^{k+1} & k < n\,, \\ v_k^n & k \ge n\,, \end{cases}$$
$$\beta_k^n := \begin{cases} 1 & k < n\,, \\ 0 & k \ge n\,, \end{cases}$$
$$\gamma_k^n := \begin{cases} 0 & k \le n\,, \\ w_k^n & k > n\,. \end{cases}$$

By definition, it holds that $\tilde{P}_k^n = P_k^n$ $(k \ge n)$. For simplicity we drop the tilde. To realize the cascade summation in the following algorithm in a convenient way, we consider

$$P := \sum_{k=0}^{2^j} \hat{a}_k^n \, P_k^n \tag{4.3}$$

with $\hat{a}_k^n := 0 \quad (k = 0, \ldots, n-1)$ instead of (4.1).
Shifting the index $k \in \mathbb{N}_0$ of the coefficients in (4.2) by $c \in \mathbb{N}_0$, we obtain the *associated functions* $P_k^n(\,\cdot\,, c)$ of P_k^n defined by

$$\begin{aligned} P_{-1}^n(x,c) &:= 0\,, \quad P_0^n(x,c) := 1\,, \\ P_{k+1}^n(x,c) &:= (\alpha_{k+c}^n x + \beta_{k+c}^n)\, P_k^n(x,c) + \gamma_{k+c}^n P_{k-1}^n(x,c) \quad (k = 0, 1, \ldots)\,. \end{aligned}$$

Now induction yields (see [3])

Lemma 4.1 *For c, $n \in \mathbb{N}_0$, it holds*

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

Lemma 4.1 implies

$$\begin{pmatrix} P_{c+k}^n \\ P_{c+k+1}^n \end{pmatrix} = \mathbf{U}_k^n(\cdot, c)^{\mathrm{T}} \begin{pmatrix} P_{c-1}^n \\ P_c^n \end{pmatrix} \tag{4.4}$$

with

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

The main part of our algorithm realizes the basis exchange from $\{P_k^n\}_{k=0}^{2^j}$ to the basis of Chebyshev polynomials $\{T_k\}_{k=0}^{2^j}$ and produces the real coefficients $\tilde{a}_k$ in

$$P = \sum_{k=0}^{2^j} \tilde{a}_k \, T_k\,. \tag{4.5}$$

Knowing the coefficients $\tilde{a}_k$, the values $P(c_s^{2^{j+1}})$ $(s = 0, \ldots, 2^{j+1})$ can be computed for example via fast DCT–I $(2^{j+1}+1)$ in $\mathcal{O}(j\, 2^j)$ arithmetical operations in a final step (see [15])

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

Let us turn to the basis exchange. Set $N = 2^j$. In the *initial step* we use (4.2) and the fact that $T_1(x) = x$ to obtain

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

with

$$\begin{aligned} a_k^{(0)}(x) \;&:=\; \hat{a}_k^n \quad (k = 0, \ldots, N-3)\,,\\ a_{N-2}^{(0)}(x) \;&:=\; \hat{a}_{N-2}^n \,+\, \gamma_N \hat{a}_N^n\,,\\ a_{N-1}^{(0)}(x) \;&:=\; \hat{a}_{N-1}^n \,+\, \beta_N \hat{a}_N^n \,+\, \alpha_N \hat{a}_N^n\, T_1(x)\,. \end{aligned} \tag{4.6}$$

Now we proceed by cascade summation as shown in Figure 1. By (4.4) with $k = 1$ and $c = 4l + 1$ $(l = n, \ldots, N/4 - 1)$ it follows that

$$(a_{4l+2}^{(0)},\, a_{4l+3}^{(0)}) \begin{pmatrix} P_{4l+2}^n \\ P_{4l+3}^n \end{pmatrix} \;=\; (a_{4l+2}^{(0)},\, a_{4l+3}^{(0)})\, \mathbf{U}_1^n(\,\cdot\,, 4l+1)^{\mathrm{T}} \begin{pmatrix} P_{4l}^n \\ P_{4l+1}^n \end{pmatrix}.$$

Thus

$$P \;=\; \sum_{l=0}^{N/4-1} (\, a_{4l}^{(1)}\, P_{4l}^n \,+\, a_{4l+1}^{(1)}\, P_{4l+1}^n)$$

with

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

The degree of the polynomial products in (4.7) is at most 3 such that their computation with respect to the the basis of Chebyshev polynomials can be realized by Algorithm 3.2 with $M = 4$.
We continue in the obvious manner. In *step* τ $(1 \le \tau < j)$ we compute by (4.4) with $k = 2^\tau - 1$ the Chebyshev coefficients of the polynomials $a_{2^{\tau+1}l}^{(\tau)}$, $a_{2^{\tau+1}l+1}^{(\tau)} \in \Pi_{2^{\tau+1}-1}$ $(l = 0, \ldots, N/2^{\tau+1} - 1)$ defined by

$$\begin{pmatrix} a_{2^{\tau+1}l}^{(\tau)} \\ a_{2^{\tau+1}l+1}^{(\tau)} \end{pmatrix} := \begin{pmatrix} a_{2^{\tau+1}l}^{(\tau-1)} \\ a_{2^{\tau+1}l+1}^{(\tau-1)} \end{pmatrix} \,+\, \mathbf{U}_{2^\tau-1}^n(\,\cdot\,, 2^{\tau+1}l+1) \begin{pmatrix} a_{2^{\tau+1}l+2^\tau}^{(\tau-1)} \\ a_{2^{\tau+1}l+2^\tau+1}^{(\tau-1)} \end{pmatrix}, \tag{4.7}$$

where we apply the Algorithm 3.2 (with $M = 2^{\tau+1}$) for the polynomial products.
After the *step* $j-1$, our cascade summation arrives at

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

Now $P_0^n(x) = \frac{((2n)!)^{1/2}}{2^n \, n!}$, $P_1^n(x) = (\alpha_1^n x + \beta_1^n) \, P_0^n(x)$ and

$$x \, T_0(x) = T_1(x) , \quad x \, T_k(x) = \frac{1}{2} (T_{k+1}(x) + T_{k-1}(x)) \quad (k = 1, 2, \ldots) .$$

Hence, if

$$a_1^{(j-1)} = \sum_{k=0}^{N-1} a_{1,k}^{(j-1)} T_k ,$$

then

$$a_1^{(j)} := a_1^{(j-1)} P_1^n = \sum_{k=0}^{N} a_{1,k}^{(j)} T_k$$

with

$$(a_{1,k}^{(j)})_{k=0}^N = (\alpha_1^n \, \mathbf{T}_{N+1}^{\mathrm{T}} + \beta_1^n \, \mathbf{I}_{N+1})(a_{1,k}^{(j-1)})_{k=0}^N , \tag{4.8}$$

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

$$\mathbf{T}_{N+1} := \begin{pmatrix} 0 & 1 & & & \\ 1/2 & 0 & 1/2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/2 & 0 & 1/2 \\ & & & 1 & 0 \end{pmatrix} . \tag{4.9}$$

This leads to

$$P = \frac{((2n)!)^{1/2}}{2^n \, n!} \left(a_0^{(j-1)} + a_1^{(j)} \right)$$

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

$$(\tilde{a}_k)_{k=0}^N = \frac{((2n)!)^{1/2}}{2^n \, n!} \left((a_{0,k}^{(j-1)})_{k=0}^N + (a_{1,k}^{(j)})_{k=0}^N \right) . \tag{4.10}$$

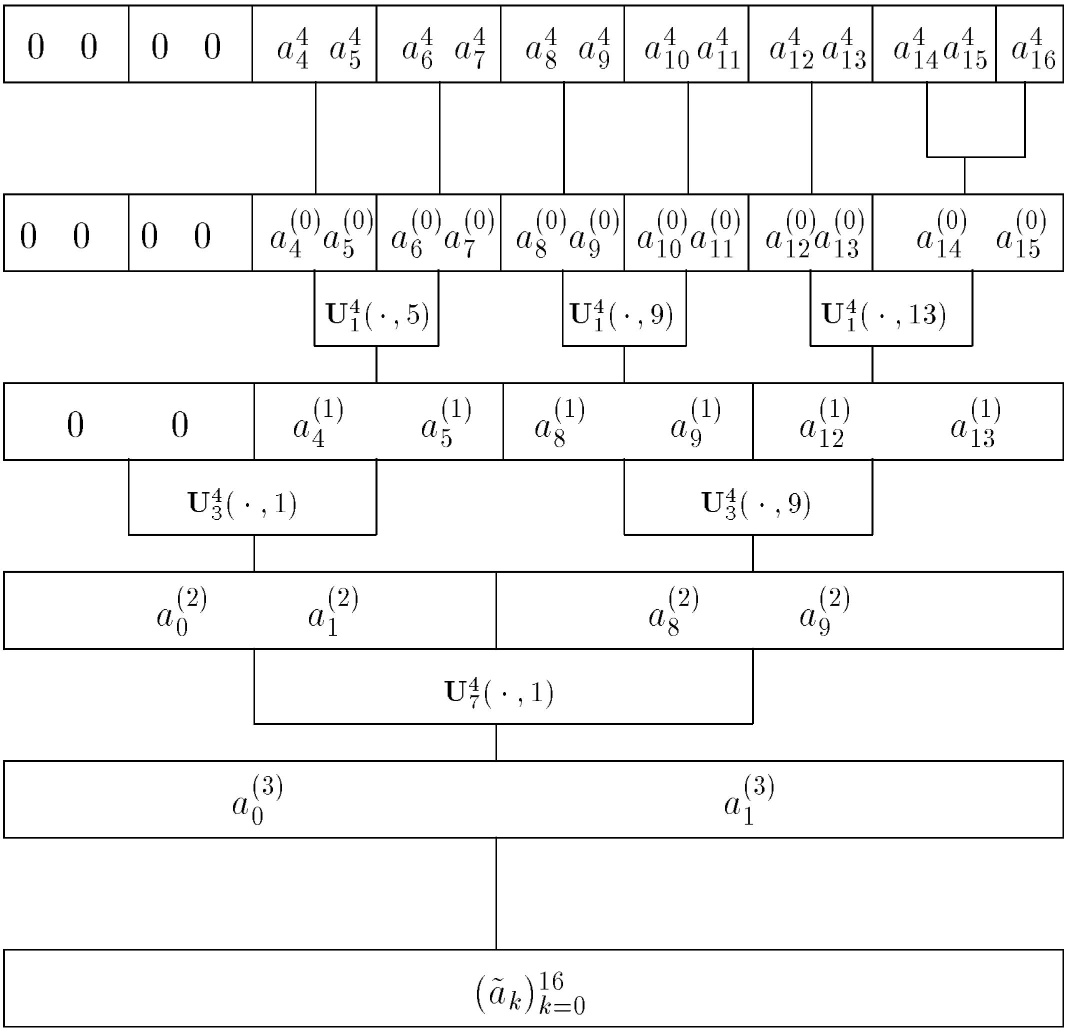

Figure 1: Cascade summation for the computation of the basis exchange for $N = 16$ and $n = 4$

We summarize:

Algorithm 4.2 (Fast Legendre function transform (FLFT))

Input: $n,\, j \in \mathbb{N}$ with $n < N := 2^j$,

$\hat{a}_k^n \in \mathbb{R}$ $(k = n, \ldots, N)$,

$\mathbf{U}_{2^\tau-1}^n(c_{2s+1}^{(\tau+2)},\, 2^{\tau+1}l+1)$ $(\tau = 1, \ldots, j-1;\ l = 0, \ldots, 2^{j-\tau-1};$

$s = 0, \ldots, 2^{\tau+1}-1)$.

Step 0. Compute $s := \lfloor \log_2 n \rfloor$ and $a_k^{(0)}$ $(k = 0, \ldots, N-1)$ by (4.6).
For $\tau = 1, \ldots, j-1$ do

For every $k = \lfloor 2^{s-\tau-1} \rfloor, \ldots, 2^{j-\tau-1}-1$ do
Step $\tau.k$. Form (4.7) by fast polynomial multiplications.

Step j. Compute $\tilde{a}_n$ $(n = 0, \ldots, N)$ by (4.8) and (4.10).

Output: $\tilde{a}_n \ \in \mathbb{R}$ $(n = 0, \ldots, N)$.

Algorithm 4.2 requires $\mathcal{O}(N \log^2 N)$ arithmetical operations. Straightforward calulation of (4.1) needs $\mathcal{O}(N(N-n))$ operations.

5 Stabilized fast Legendre function transform

For fixed n and given $\hat{a}_k^n \in \mathbb{R}$ $(k = n, \ldots, 2^j)$ we have computed

$$f_l = \sum_{k=n}^{2^j} \hat{a}_k^n \, P_k^n(c_l^{2^j}) \quad (l = 0, \ldots, 2^j) \tag{5.1}$$

by the Clenshaw algorithm (CA) [5] in double precision arithmetic, by the Clenshaw algorithm realized in Maple with high precision arithmetic of 64 digits (CA_{64}), and by our fast Legendre function transform (FLFT) in double precision arithmetic. The algorithm was implemented in C and tested on a Sun SPARCstation 20. Table 1 compares the results for $N = 1024$ and various parameters n. The second column of Table 1 contains the given coefficients $\hat{a}_k^n$, while the third and last columns contain the relative error $\varepsilon(\mathrm{CA})$ of the Clenshaw algorithm defined by

$$\varepsilon(\mathrm{CA}) := \max_{0 \le l \le 2^j} |f_l(\mathrm{CA}) - f_l(\mathrm{CA}_{64})| / \max_{0 \le l \le 2^j} |f_l(\mathrm{CA}_{64})|$$

and the relative error of Algorithm 4.2 given by

$$\varepsilon(\mathrm{FLFT}) := \max_{0\le l\le 2^j} |f_l(\mathrm{FLFT}) - f_l(\mathrm{CA}_{64})| / \max_{0\le l\le 2^j} |f_l(\mathrm{CA}_{64})| \,.$$

Here $f_l(\mathrm{CA})$, $f_l(\mathrm{CA}_{64})$ and $f_l(\mathrm{FLFT})$ denote the corresponding results of (5.1) using CA, CA_{64}, and FLFT, respectively.

n	$\hat{a}_k^n$	$\varepsilon(\mathrm{CA})$	$\varepsilon(\mathrm{FLFT})$
0	1	$1.95E-12$	$2.18E-11$
8	1	$1.02E-12$	$6.13E-11$
16	1	$6.83E-13$	$5.34E-13$
24	1	$7.62E-13$	$5.28E-08$
32	1	$4.10E-13$	$7.16E-06$
48	1	$2.02E-13$	$2.58E-01$
64	1	$3.26E-13$	$1.00E-00$

Table 1: Relative error of the CA versus the FLFT for $N = 1024$

Note that the extensive error analysis of the Driscoll–Healy algorithm for the case $n = 0$ in [6] which demonstrates the stability of the algorithm for Legendre polynomials can be developed for our algorithm in a similar way. While the relative error is acceptable for $0 \le n \le 16$, it becomes quite large if n further increases. Similar results appear for other transform lengths N. The instability results from the fact that for some *special tripels* $(n, k, c) \in \{(n, 2^\tau - 1, 2^{\tau+1}l + 1) : n = 0, \ldots, N;\ \tau = 1, \ldots, j-1;\ l = 0, \ldots, N/2^{\tau+1}\}$ the absolute values of some entries of the matrices $\mathbf{U}_k^n(x, c)$ become very large for $|x| \approx 1$ while the entries of $\mathbf{U}_k^n(x, 1)$ (or more general of $\mathbf{U}_k^n(x, c)$ with $c < n$) become very small. The multiplication of the resulting small and large values introduces large roundoff errors. Consequently, the simplest idea consists in replacing the ordinary cascade summation step by "special" stabilization steps whenever the values $|P_k^n(x, c)|$ involved in the algorithm cross some threshold. This straightforward idea was firstly formulated in [9] as so-called "stability bypass operations" in connection with the Driscoll–Healy algorithm. To avoid the multiplications with large values, we replace the multiplications with $\mathbf{U}_k^n(\,\cdot\,, c)$ by those with $\mathbf{U}_{k+c-1}^n(\,\cdot\,, 1)$. By

$$\mathbf{U}_{k+c-1}^n(\,\cdot\,, 1) \;=\; \mathbf{U}_{n-1}^n(\,\cdot\,, 1)\, \mathbf{U}_{c-n-2}^n(\,\cdot\,, n+1)\, \mathbf{U}_k^n(\,\cdot\,, c)$$

with

$$\mathbf{U}_{n-1}^n(x, 1) \;=\; \begin{pmatrix} 0 & 0 \\ (1-x^2)^{(n-2)/2}(1+x) & (1-x^2)^{n/2} \end{pmatrix}$$

the entries of $\mathbf{U}^n_{k+c-1}(x,1)$ are significantly smaller than those of $\mathbf{U}^n_k(x,c)$ for $|x| \approx 1$. With other words, instead of step $\tau.\ k$ in Algorithm 4.2 we compute

$$\begin{pmatrix} a^{(\tau)}_{0,\text{new}} \\ \\ a^{(\tau)}_{1,\text{new}} \end{pmatrix} := \begin{pmatrix} a^{(\tau)}_0 \\ \\ a^{(\tau)}_1 \end{pmatrix} + \mathbf{U}^n_{2^\tau(2k+1)-1}(\,\cdot\,,1) \begin{pmatrix} a^{(\tau-1)}_{2^{\tau+1}k+2^\tau} \\ \\ a^{(\tau-1)}_{2^{\tau+1}k+2^\tau+1} \end{pmatrix}$$

by fast polynomial multiplications. Unfortunately, the number of arithmetical operations required by an exceptional step is quite larger than those of the corresponding ordinary step $\tau.k$ since we need DCT of length $2^{1+\lfloor \log_2(2^\tau+c-1)\rfloor}$ instead of length $2^{\tau+1}$. Moreover, the number of exceptional steps increases with n while $n \le N/2$.

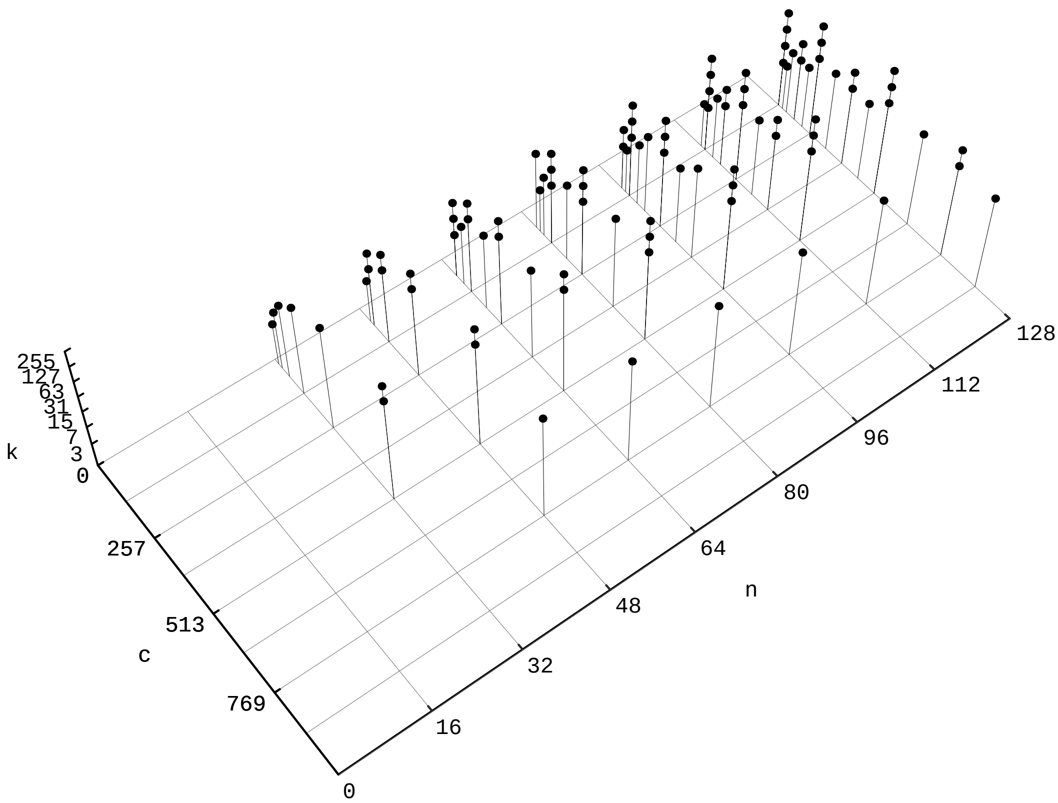

Figure 2: Location of stabilization steps (n,k,c) for $N = 1024$, $16 < n \le 128$ and threshold 10^4

Example. For $N = 1024$, Figure 2 indicates the tripels (n, k, c) $(16 < n \leq 128)$ with the property that some entries of $\mathbf{U}_k^n(\cdot, c)$ have absolute values $> 10^4$. For $16 < n \leq 32$ only some entries of $\mathbf{U}_7^n(\,\cdot\,, 17)$, $\mathbf{U}_{15}^n(\,\cdot\,, 33)$, $\mathbf{U}_{31}^n(\,\cdot\,, 65)$, $\mathbf{U}_{63}^n(\,\cdot\,, 129)$, $\mathbf{U}_{127}^n(\,\cdot\,, 257)$, $\mathbf{U}_{127}^n(\,\cdot\,, 513)$, $\mathbf{U}_{255}^n(\,\cdot\,, 513)$ can become $> 10^4$ such that we have to introduce at most 7 exceptional steps. For $32 < n \leq 48$ we have to stabilize 10 steps and so on.
Table 2 demonstrates that our stabilized FLFT with threshold 10^4 realizes (4.1) with almost double precision accuracy.

n	$\hat{a}_k^n$	$\varepsilon(\mathrm{CA})$	$\varepsilon(\mathrm{FLFT})$
24	1	$7.62E-13$	$8.06E-12$
32	1	$4.10E-13$	$1.38E-10$
48	1	$2.02E-13$	$1.09E-10$
64	1	$3.26E-13$	$4.45E-10$
80	1	$2.83E-13$	$3.09E-10$
80	$1/(k+1)$	$2.71E-13$	$7.47E-10$
96	$1/(k+1)$	$1.70E-13$	$7.48E-10$
112	$1/(k+1)$	$2.07E-13$	$4.17E-10$
224	$1/(k+1)$	$7.67E-14$	$4.34E-10$
768	$1/(k+1)$	$4.48E-14$	$1.42E-10$

Table 2: Relative error of the CA versus the stabilized FLFT for $N = 1024$ and threshold 10^4

By choosing higher thresholds, we can decrease the number of stabilization steps and hence the number of arithmetical operations at the cost of a lower accuracy. For example, a threshold of 10^6 implies an error of $\approx 10^{-7}$ in the above calculations.
Table 3 lists the CPU–times $t(\mathrm{CA})$ and $t(\mathrm{FLFT})$ (in seconds) for the Clenshaw algorithm and for the stabilized FLFT while Figure 3 shows the number of multiplications required by the stabilized FLFT (star) and by the Clenshaw algorithm (cross). Here the coefficients $\hat{a}_k^n$ $(k = n, \ldots, 2^{10})$ are randomly distributed in the interval $[-0.5, 0.5]$. The last column of Table 3 contains the relative error

$$\tilde{\varepsilon}(\mathrm{FLFT}) := \max_{0 \leq l \leq 2^j} |f_l(\mathrm{FLFT}) - f_l(\mathrm{CA})| / \max_{0 \leq l \leq 2^j} |f_l(\mathrm{CA})| \,.$$

n	$t(\mathrm{CA})$	$t(\mathrm{FLFT})$	$\tilde{\varepsilon}(\mathrm{FLFT})$
0	4.15	0.38	$2.21E-11$
128	4.09	0.74	$8.74E-11$
256	3.52	1.18	$1.67E-10$
384	3.33	1.38	$1.84E-10$
512	2.45	1.63	$7.38E-10$
640	2.09	1.53	$6.43E-11$
768	1.49	1.42	$1.11E-10$

Table 3: CPU-time of the CA versus the stabilized FLFT for $N = 1024$ and theshold 10^4

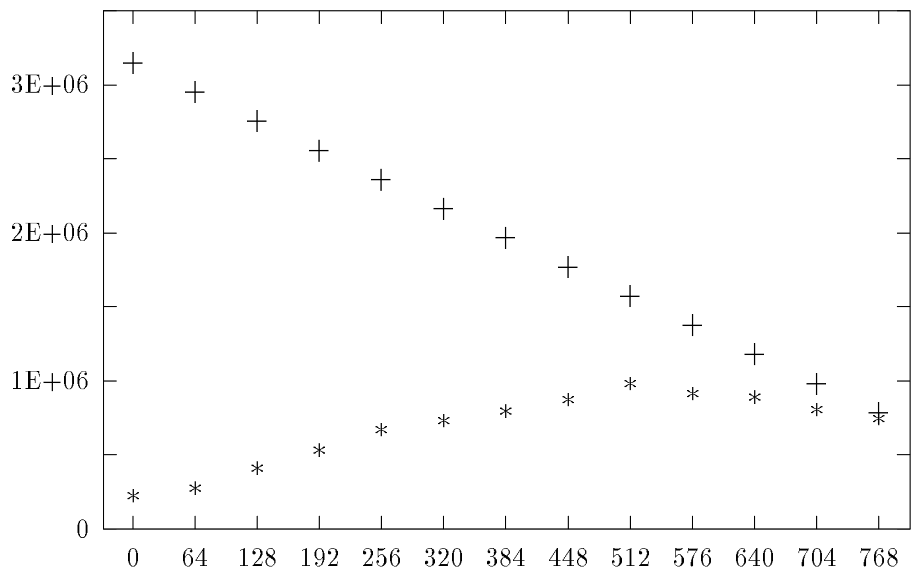

Figure 3: Number of multiplications of the CA (cross) and the stabilized FLFT (star) for $N = 1024$, $n = 64l$ $(l = 0, \ldots, 12)$ and threshold 10^4

Finally, Figure 4 compares the number of arithmetical operations of the CA and the stabilized FLFT for different transform lengths N, $n = 0, N/4, N/2, 3N/4$ and the thresholds 10^4 and 10^6.

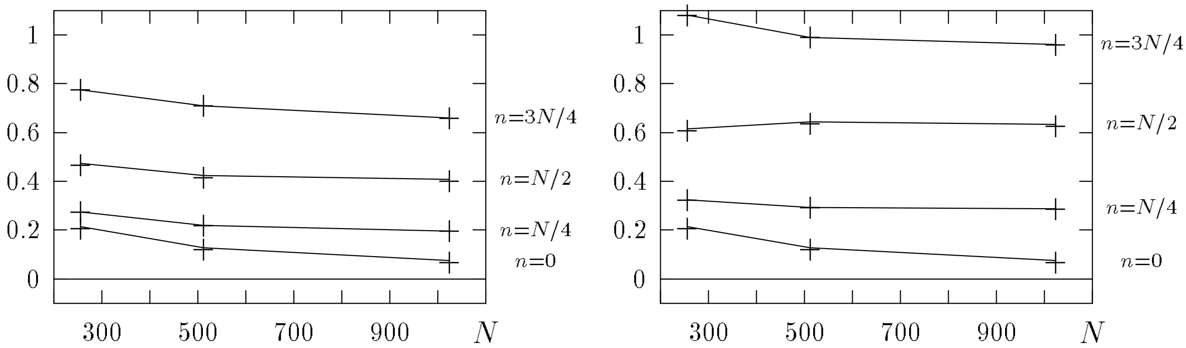

Figure 4: Quotient of the number of arithmetical operations of the stabilized FLFT and the CA for the thresholds 10^4 (right) and 10^6 (left)

Up to now the location of the stabilization steps was precomputed by computer simulation. To determine the location of the stabilization steps in a theoretical way, the following open questions concerning the behaviour of the associated functions $P_k^n(\,\cdot\,, c)$ must be answered. By

$$P_k^n(-x, c) = (-1)^k \, P_k^n(x, c)\,,$$

we conjecture that the associated functions $P_k^n(\,\cdot\,, c)$ fulfil the following property

$$\max\{|P_k^n(x, c)| : x \in [0, 1]\} = P_k^n(1, c)$$

for $n < c$. Moreover we are interested in the behaviour of $P_k^n(1, c)$ as function of c in the case $n < c$.

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., in print.

[3] S. Belmehdi, On the associated orthogonal polynomials, J. Comput. Appl. Math. **32** (1991), 311 – 319.

[4] G.L. Browning, J.J. Hack, and P.N. Swarztrauber, A comparison of three numerical methods for solving differential equations on the sphere, Monthly Weather Rev., **117** (1989), 1058 – 1075.

[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. Comput., in print.

[8] W. Freeden and U. Windheuser, Combined spherical harmonic and wavelet expansion, preprint, Univ. of Kaiserslautern, 1996.

[9] S.S.B. Moore, Efficient stabilization methods for fast polynomial transforms, Thesis, Dartmouth College, 1994.

[10] 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.

[11] Müller, C., *Spherical Harmonics*, Springer, Berlin, 1966.

[12] D. Potts, G. Steidl, and M. Tasche, Kernels of spherical harmonics and spherical frames, in: *Advanced Topics in Multivariate Approximation*, F. Fontanella, K. Jetter and P.–J. Laurent (eds.), World Scientific Publ., in print.

[13] D. Potts, G. Steidl, and M. Tasche, Fast algorithms for discrete polynomial transforms, preprint, TH Darmstadt, 1996.

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

[15] 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.

[16] P.N. Swarztrauber, The vector harmonic transform method for solving partial differential equations in spherical geometry, Monthly Weather Rev. **121** (1993), 3415 – 3437.