Fast spherical Fourier algorithms

Stefan Kunis and Daniel Potts*

Abstract

Spherical Fourier series play an important role in many applications. A numerically stable fast transform analogous to the Fast Fourier Transform is of great interest. For a standard grid of $\mathcal{O}(N^2)$ points on the sphere, a direct calculation has computational complexity of $\mathcal{O}(N^4)$, but a simple separation of variables reduces the complexity to $\mathcal{O}(N^3)$. Here we improve well-known fast algorithms for the discrete spherical Fourier transform with a computational complexity of $\mathcal{O}(N^2 \log^2 N)$. Furthermore we present, for the first time, a fast algorithm for scattered data on the sphere. For arbitrary $\mathcal{O}(N^2)$ points on the sphere, a direct calculation has a computational complexity of $\mathcal{O}(N^4)$, but we present an approximate algorithm with a computational complexity of $\mathcal{O}(N^2 \log^2 N)$.

2000 AMS Subject Classification: 65T99, 33C55, 42C10, 65T50
Keywords: spherical Fourier transform, spherical harmonics, associated Legendre functions, fast discrete transforms, fast Fourier transform at nonequispaced knots

1 Introduction

Fourier analysis on the sphere $\mathbb{S}^2 \subset \mathbb{R}^3$ has practical relevance in tomography, geophysics, seismology, meteorology and crystallography. The discrete spherical Fourier transform (DSFT) plays an essential role in many applications. In particular, there is a growing need for the fast summation of spherical harmonic expansions (see [6, 26, 21, 32, 14]). Unfortunately, working with spherical harmonics is computationally complex in many respects [5].

The Fourier expansion of a function $f \in L^2(\mathbb{S}^2)$ with bandwidth M is given by

$$f = \sum_{(k,n)\in I^M} a_k^n(f) Y_k^n \ , \tag{1.1}$$

where $I^M := \{(k,n) : k = 0,\ldots,M; n = -k,\ldots,k\}$ and $a_k^n(f)$ are the spherical Fourier coefficients of f with respect to the orthogonal basis of spherical harmonics Y_k^n, where n denotes the order and k the degree.

The aim of this paper is twofold. In the first part we improve a fast algorithm for computing f on a special grid. Since the spherical harmonics $Y_k^n(\vartheta,\varphi)$ $((\vartheta,\varphi) \in S,\ S := [0,\pi] \times [0,2\pi))$ are scaled products of complex exponentials $\mathrm{e}^{\mathrm{i}n\varphi}$ for the longitudinal coordinate and Legendre functions $P_k^{|n|}(\cos\vartheta)$ for the colatitudinal coordinate, the discrete spherical Fourier transform on grids can be split up into ordinary discrete Fourier transforms for exponentials, which can be realized by fast Fourier transform (FFT) techniques, and discrete Legendre function transforms.

*University of Lübeck, Institute of Mathematics, D–23560 Lübeck, (kunis@informatik.uni-luebeck.de, potts@math.uni-luebeck.de).

We give a simple approach to the Legendre function transform based on cascade summation using fast polynomial transforms. Using a fast polynomial transform scheme based on discrete cosine transforms (DCT) leads to the algorithm given in [26], whereas using the fast multipole method (FMM) leads to the algorithm in [32]. Here we work out the commonalities and improve these known algorithms. It is a fact that one can compute the spherical Fourier coefficients $a_k^n(f)$ of a band–limited function by a convenient quadrature rule, for example with Clenshaw–Curtis quadrature [25] or with Gauß quadrature [32]. It is well known that in general a fast algorithm for (1.1) implies the factorization of the transform matrix into a product of sparse matrices. Consequently, once a fast algorithm for (1.1) is known, a fast algorithm for the transposed problem (i.e. computing the spherical Fourier coefficients) can also be obtained by transposing the sparse matrix product (see [6, 15]).

In the second part of this paper we propose a novel fast algorithm for evaluating the band–limited function f in (1.1) at arbitrary nodes. The main idea is to use an algorithm on a special grid and to perform a change of basis, such that f in (1.1) can be represented in the form

$$f\,(\vartheta,\varphi) = \sum_{n=-M}^{M}\sum_{k=-M}^{M} c_k^n \mathrm{e}^{\mathrm{i}k\vartheta}\mathrm{e}^{\mathrm{i}n\varphi}$$

with complex coefficients c_k^n. Then, the computation on arbitrary nodes can be done using the recently developed fast Fourier transform for nonequispaced knots (NFFT) (see [27, 34]). As mentioned above, we immediately have a fast algorithm for the transposed problem from the theoretical point of view, i.e. for computing spherical Fourier coefficients from values of the function f at arbitrary points, provided that a convenient quadrature rule is available (see [12, 20]).

Note that double Fourier series are employed as basis function in spectral methods for the solution of PDEs in spherical coordinates [5, pp. 434].

This paper is organized as follows: In Section 2 we introduce the notation for the discrete cosine transforms. In Section 3 we give two different algorithms for fast polynomial multiplications, one based on the discrete cosine transform and the other on the fast multipole method. The fast transform for Legendre functions is explained in Section 4, where we also describe a stabilization method. Finally, in Section 5 we present a method for computing the discrete spherical Fourier transform at arbitrary nodes. Experimental results with an application to the EGM96 model are discussed in Section 6.

2 Discrete cosine transforms

Let

$$\begin{aligned} \boldsymbol{C}_{N+1} &:= \left(\cos\tfrac{jk\pi}{N}\right)_{j,k=0,\ldots,N}, & \boldsymbol{D}_{N+1} &:= \operatorname{diag}\,(\varepsilon_j^N)_{j=0,\ldots,N}\,, \\ \tilde{\boldsymbol{C}}_N &:= \left(\cos\tfrac{(2j+1)k\pi}{2N}\right)_{j,k=0,\ldots,N-1}, & \tilde{\boldsymbol{D}}_N &:= \operatorname{diag}\,(\varepsilon_j^N)_{j=0,\ldots,N-1} \end{aligned}$$

with $\varepsilon_0^N = \varepsilon_N^N := 1/2$ and $\varepsilon_k^N := 1$ for $k = 1,\ldots,N-1$. The Chebyshev polynomials of the first kind T_k are given by $T_k(x) := \cos(k\,\arccos(x))$. The following transforms are referred to as *discrete cosine transforms* (DCT) *of type* I, II and III, respectively (see [33]):

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

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

where $\boldsymbol{a} := (a_k)_{k=0,\dots,N}$ is the input vector and $\hat{\boldsymbol{a}} := (\hat{a}_j)_{j=0,\dots,N} \in \mathbb{R}^{N+1}$ is the output vector, i.e.

$$\hat{a}_j = \sum_{k=0}^{N} \varepsilon_k^N a_k \cos\frac{jk\pi}{N} = \sum_{k=0}^{N} \varepsilon_k^N a_k\, T_k\left(\cos\frac{j\pi}{N}\right),$$

DCT–II (N): $\mathbb{R}^N \to \mathbb{R}^N$ with

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

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

$$\hat{b}_j = \sum_{k=0}^{N-1} b_k \cos\frac{j(2k+1)\pi}{2N} = \sum_{k=0}^{N-1} b_k\, T_j\left(\cos\frac{(2k+1)\pi}{2N}\right),$$

DCT–III (N): $\mathbb{R}^N \to \mathbb{R}^N$ with

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

i.e.

$$\hat{b}_j := \sum_{k=0}^{N-1} \varepsilon_k^N b_k \cos\frac{k(2j+1)\pi}{2N} = \sum_{k=0}^{N-1} \varepsilon_k^N b_k\, T_k\left(\cos\frac{(2j+1)\pi}{2N}\right).$$

In the following, let $N = 2^t$ $(t \in \mathbb{N})$. There exist various fast algorithms that perform the discrete cosine transforms defined above with $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$ arithmetical operations (see [28]). Fast algorithms for DCTs based on [30] can be found in [2] (see also [31]). Concerning the inverse DCTs, it is easy to check the following relation (see [2])

Lemma 2.1. *We have*

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

3 Fast polynomial multiplication

A polynomial P of degree $N-1$, denoted by $P \in \Pi_{N-1}$, is uniquely determined by values at N different points. Let us assume, however, that we are given $N+1$ values for a polynomial $P \in \Pi_{N-1}$ $(N = 2^t, t \in \mathbb{N})$, and let these values be given at Chebyshev nodes $\cos(j\pi/N)$ $(j = 0,\dots,N)$, which means that we have the $N+1$ values $\left(P\left(\cos\frac{j\pi}{N}\right)\right)_{j=0,\dots,N}$. Further, let $Q \in \Pi_N$ be a fixed polynomial with known values $\left(Q\left(\cos\frac{j\pi}{2N}\right)\right)_{j=0,\dots,2N}$.

In the following we will describe two different ways of computing the values $\left(R\left(\cos\frac{j\pi}{2N}\right)\right)_{j=0,\dots,2N}$, where $R = PQ$, in a fast way. Since the polynomial $R \in \Pi_{2N-1}$ is uniquely determined by the values of the product $P\left(\cos\frac{j\pi}{2N}\right) Q\left(\cos\frac{j\pi}{2N}\right)$ $(j = 0,\dots,2N)$ we have to compute the values $\left(P\left(\cos\frac{(2j+1)\pi}{2N}\right)\right)_{j=0,\dots,N-1}$.

The first algorithm is an exact one and is based on the computation of the Chebyshev coefficients.

Note that $P \in \Pi_{N-1}$ can be written with respect to the basis of Chebyshev polynomials, i.e.

$$P = \sum_{k=0}^{N-1} a_k T_k$$

with real coefficients a_k. From the equation

$$\left(P\left(\cos \frac{j\pi}{N}\right)\right)_{j=0,\dots,N} = \boldsymbol{C}_{N+1} \, (a_k)_{k=0,\dots,N}$$

with $a_N = 0$ we use Lemma 2.1 to obtain

$$(a_k)_{k=0,\dots,N} \quad = \quad \frac{2}{N} \boldsymbol{D}_{N+1} \boldsymbol{C}_{N+1} \boldsymbol{D}_{N+1} \left(P\left(\cos \frac{j\pi}{N}\right)\right)_{j=0,\dots,N} \tag{3.1}$$

and finally

$$\left(P\left(\cos \frac{(2j+1)\pi}{2N}\right)\right)_{j=0,\dots,N-1} \quad = \quad \tilde{\boldsymbol{C}}_N^{\mathrm{T}} (a_k)_{k=0,\dots,N-1} \, . \tag{3.2}$$

Now we can summarize these results in Algorithm 3.1.

Algorithm 3.1 Fast polynomial multiplication based on DCTs

Input: $N = 2^t$ $(t \in \mathbb{N})$
$\left(P\left(\cos \frac{j\pi}{N}\right)\right)_{j=0,\dots,N}$
$\left(Q\left(\cos \frac{j\pi}{2N}\right)\right)_{j=0,\dots,2N}$

Compute the Chebyshev coefficients $(a_k)_{k=0,\dots,N-1}$ in (3.1) using a fast DCT-I
Evaluate the values $\left(P\left(\cos \frac{(2j+1)\pi}{2N}\right)\right)_{j=0,\dots,N-1}$ in (3.2) using a fast DCT-III (N)
for $j = 0, \dots, 2N$ **do**
$R\left(\cos \frac{j\pi}{2N}\right) := P\left(\cos \frac{j\pi}{2N}\right) Q\left(\cos \frac{j\pi}{2N}\right)$
end for

Output: $\left(R\left(\cos \frac{j\pi}{2N}\right)\right)_{j=0,\dots,2N}$

Remark 3.1. Note that Algorithm 3.1 requires one DCT-I($N+1$) and one DCT-III(N). Algorithm 2.2 [26] requires one DCT-III($2N$) and one DCT-II($2N$) for a polynomial multiplication based on computing Chebyshev coefficients. This means that Algorithm 3.1 is faster than Algorithm 2.2 in [26]. □

The second algorithm we will compute the values $\left(P\left(\cos \frac{(2j+1)\pi}{2N}\right)\right)_{j=0,\dots,N-1}$ with the help of a Lagrange interpolation formula and a fast approximate realization of the matrix times vector multiplication with a Cauchy matrix. To be more precise, we rewrite P using Lagrange interpolation in the form

$$P(x) = \sum_{j=0}^{N} P\left(\cos \frac{j\pi}{N}\right) \frac{\omega(x)}{\left(x - \cos \frac{j\pi}{N}\right) \omega'\left(\cos \frac{j\pi}{N}\right)}$$

with $\omega(x) := 2^{N-1} \prod_{k=0}^{N} \left(x - \cos \frac{k\pi}{N}\right)$. Let U_{N-1} be the Chebyshev polynomials of the second kind given by

$$U_{N-1}(x) := \frac{\sin(N \arccos(x))}{\sqrt{1-x^2}}, \quad x \in (-1,1) .$$

We see by $\omega(x) = (1-x^2)U_{N-1}(x)$ that

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

and finally

$$P\left(\cos \frac{(2k+1)\pi}{2N}\right) = \frac{(-1)^k}{N} \sin \frac{(2k+1)\pi}{2N} \sum_{j=0}^{N} \frac{\varepsilon_{N,j}(-1)^{j+1} P\left(\cos \frac{j\pi}{N}\right)}{\cos\left(\frac{(2k+1)\pi}{2N}\right) - \cos\left(\frac{j\pi}{N}\right)} \tag{3.3}$$

for $k = 0, \ldots, N-1$. This leads to Algorithm 3.2.

Algorithm 3.2 Fast polynomial multiplication based on FMM

Input: $N = 2^t$ $(t \in \mathbb{N})$
$\left(P\left(\cos \frac{j\pi}{N}\right)\right)_{j=0,\ldots,N}$
$\left(Q\left(\cos \frac{j\pi}{2N}\right)\right)_{j=0,\ldots,2N}$

Compute the values $\left(P\left(\cos \frac{(2k+1)\pi}{2N}\right)\right)_{k=0,\ldots,N-1}$ in (3.3) using a FMM (see Algorithm 3.1 in [8]) with $\mathcal{O}(N \log(1/\varepsilon))$ arithmetical operations
for $j = 0, \ldots, 2N$ **do**
  $R\left(\cos \frac{j\pi}{2N}\right) := P\left(\cos \frac{j\pi}{2N}\right) Q\left(\cos \frac{j\pi}{2N}\right)$
end for

Output: $\left(R\left(\cos \frac{j\pi}{2N}\right)\right)_{j=0,\ldots,2N}$

Remark 3.2. The FMM [13] was originally designed to compute the sums $\sum_{k=1}^{N} \alpha_k/(y_j - x_k)$ for points $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots, y_N\}$ on the complex plane in an approximative way. Algorithm 3.2 is known as the linear–time interpolation algorithm and was introduced in [8]. This algorithm was first applied to fast Legendre transforms in [32]. Concerning numerical experiments Algorithm 3.1 is faster than 3.2 for the transform lengths $t = 2, 3, \ldots, 11$ that were tested. However, Algorithm 3.2 is not restricted to the knots $\cos \frac{l\pi}{N}$ $(l = 0, \ldots, N)$. This fact was exploited in [32] to stabilize Algorithm 4.1 (see Section 4). □

Remark 3.3. Note that equations (3.1), (3.2) and (3.3) imply different representations of the Cauchy matrix $\left(\frac{1}{\cos\left(\frac{(2k+1)\pi}{2N}\right) - \cos\left(\frac{j\pi}{N}\right)}\right)_{k=0,\ldots,N-1;j=0,\ldots,N}$. Further representations of similar Cauchy matrices based on other trigonometric transforms are given in [16]. □

4 Fast transform for Legendre functions

This section collects the basic tools for the efficient and stable computation of the Legendre function transform (see [5, p.399]).

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 $(n \in \mathbb{N}_0; k = n, n+1, \dots)$ as

$$P_k^n(x) := \left(\frac{(k-n)!}{(k+n)!}\right)^{1/2} (1-x^2)^{n/2} \frac{\mathrm{d}^n}{\mathrm{d}x^n} P_k(x) \quad (x \in [-1,1]) . \tag{4.1}$$

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

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

Moreover, the associated Legendre functions fulfill the three-term recurrence relation

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

$$P_{k+1}^n(x) = v_k^n x P_k^n(x) + w_k^n P_{k-1}^n(x) \quad (k = n, n+1, \dots) \tag{4.2}$$

with

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

A simple but powerful idea in [26] is to define the functions P_k^n also for $k < n$. This was done as follows. For even n we start with $P_{-1}^n := 0$ and $P_0^n(x) := \frac{\sqrt{(2n)!}}{2^n n!}$ and for odd n let $P_0^n(x) := P_1^n(x) := \frac{\sqrt{(2n)!}}{2^n n!}(1-x^2)^{1/2}$. We introduce the functions P_k^n by the three-term recurrence relation

$$P_{k+1}^n(x) := (\alpha_k^n x + \beta_k^n) P_k^n(x) + \gamma_k^n P_{k-1}^n(x) \tag{4.3}$$

with

$$\alpha_k^n := \begin{cases} (-1)^{k+1} & \text{for } k < n, \\ v_k^n & \text{otherwise,} \end{cases}$$

$$\beta_k^n := \begin{cases} 1 & \text{for } k < n, \\ 0 & \text{otherwise,} \end{cases}$$

$$\gamma_k^n := \begin{cases} 0 & \text{for } k \le n, \\ w_k^n & \text{otherwise.} \end{cases}$$

It is a fact, that P_k^n is a polynomial of degree k for even n and that $(1-x^2)^{-1/2} P_k^n$ is a polynomial of degree $k-1$ for odd n. Furthermore, these functions coincide with (4.1) for $k \ge n$. The main reason for defining P_k^n also for $k < n$ is that this modification allows a stable computation of the Legendre function transform.

By shifting the index k in (4.3) by $c \in \mathbb{N}_0$ we obtain the *associated Legendre polynomials* $P_k^n(\cdot, c)$ for P_k^n defined by

$$P_{k+1}^n(x,c) := (\alpha_{k+c}^n x + \beta_{k+c}^n) P_k^n(x,c) + \gamma_{k+c}^n P_{k-1}^n(x,c) \qquad (k = 0, 1, \dots) \tag{4.4}$$

with $P_{-1}^n(x,c) := 0$ and $P_0^n(x,c) := 1$.
Induction now yields the following lemma.

Lemma 4.1. *Let $P_c^n(x)$ and $P_k^n(x,c)$ be given by (4.3) and (4.4), respectively. We than have*

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

Lemma 4.1 implies

$$\begin{pmatrix} P_{c+k}^n \\ P_{c+k+1}^n \end{pmatrix} = \boldsymbol{U}_k^n(\cdot, c)^{\mathrm{T}} \begin{pmatrix} P_{c-1}^n \\ P_c^n \end{pmatrix} \tag{4.5}$$

with

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

Let $M \in \mathbb{N}, n \in \mathbb{Z}$ with $M > |n|$ be given. We consider the polynomials

$$g_n(x) := \sum_{k=|n|}^{M} a_k^n P_k^{|n|}(x) \in \Pi_M \tag{4.6}$$

for even n and

$$g_n(x) := \frac{1}{\sqrt{1-x^2}} \sum_{k=|n|}^{M} a_k^n P_k^{|n|}(x) \in \Pi_{M-1} \tag{4.7}$$

for odd n with real coefficients a_k^n.

The concern of the fast Legendre function transform is the fast evaluation of

$$(g_{s,n})_{s=0,\dots,N} := \Big(g_n(\cos\frac{s\pi}{N})\Big)_{s=0,\dots,N} \qquad (N := 2^{\lceil \log_2 M \rceil})\,. \tag{4.8}$$

Note that we generalize this result in Section 6 such that we are able to compute g_n on arbitrary knots. In the following we will restrict our attention to the case for fixed even n. First of all we use (4.3) to obtain

$$g_n = \sum_{k=|n|}^{M-1} a_k^{(0)} P_k^{|n|} = \sum_{k=\lfloor \frac{|n|}{4} \rfloor}^{\lceil \frac{M}{4} \rceil - 1} \left(\sum_{l=0}^{3} a_{4k+l}^{(0)} P_{4k+l}^{|n|} \right)$$

with

$$\begin{aligned} a_k^{(0)}(x) &:= a_k^n \quad (k = 0, \dots, N-3)\,, \\ a_{N-2}^{(0)}(x) &:= a_{N-2}^n + \gamma_{N-1}^{|n|} a_N^n\,, \\ a_{N-1}^{(0)}(x) &:= a_{N-1}^n + \beta_{N-1}^{|n|} a_N^n + \alpha_{N-1}^{|n|} a_N^n x\,, \end{aligned} \tag{4.9}$$

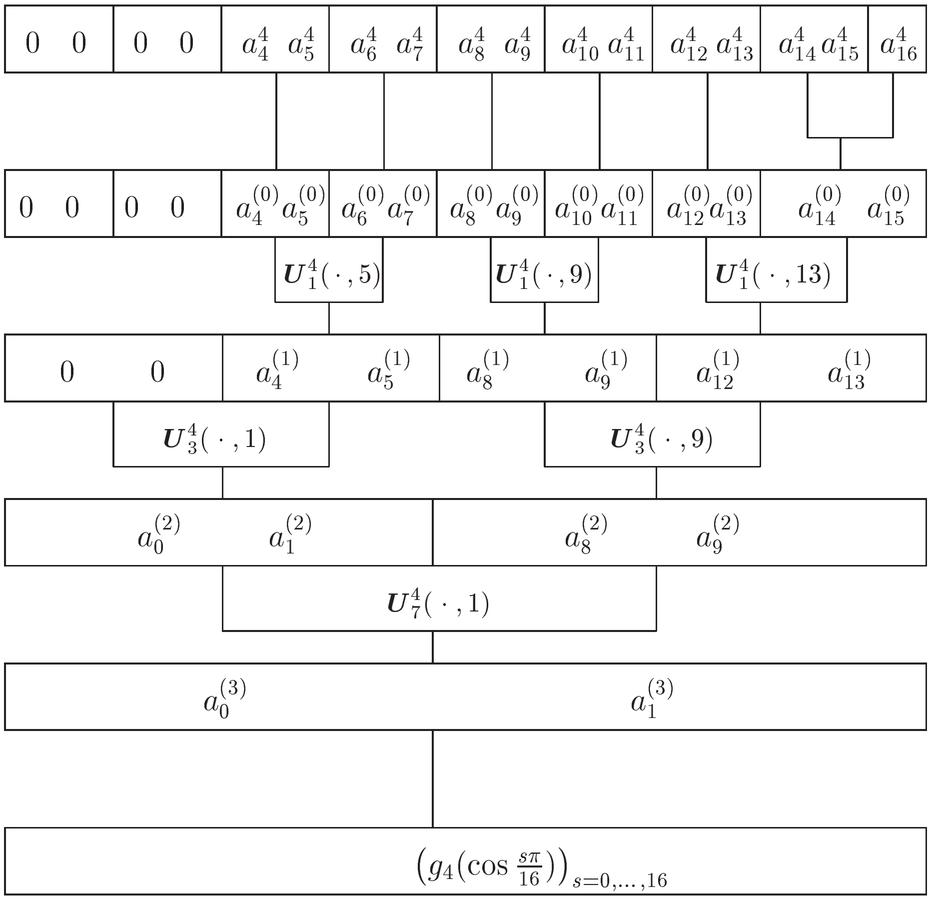

Figure 4.1: Cascade summation for the computation of the values $\left(g_4\left(\cos\frac{s\pi}{16}\right)\right)_{s=0,\dots,16}$.

where $a_k^n := 0$ for $k < |n|$ and for $k > M$. In order to apply Algorithm 3.1 we compute the values $\left(a_k^{(0)}\left(\cos\frac{s\pi}{2}\right)\right)_{s=0,1,2}$ for $k = 2\lfloor\frac{|n|}{2}\rfloor, \dots, 2\left(\lceil\frac{M}{2}\rceil - 1\right)$. We mention without proof that $a_k^{(\tau-1)} \equiv 0$ for $k < 2^\tau\lfloor\frac{|n|}{2^\tau}\rfloor$ and for $k > 2^\tau\left(\lceil\frac{M}{2^\tau}\rceil - 1\right)$ $(\tau = 1, \dots, \lceil\log_2 M\rceil - 1)$. Now we proceed by cascade summation as shown in Figure 4.1. From (4.5) with $k = 1$ and $c = 4l + 1$ it follows that

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

for $l = \lfloor\frac{|n|}{4}\rfloor, \dots, \lceil\frac{M}{4}\rceil - 1$. Thus

$$g_n = \sum_{l=\lfloor\frac{|n|}{4}\rfloor}^{\lceil\frac{M}{4}\rceil-1} a_{4l}^{(1)} P_{4l}^{|n|} + a_{4l+1}^{(1)} P_{4l+1}^{|n|}$$

with

$$\begin{pmatrix} a_{4l}^{(1)} \\ a_{4l+1}^{(1)} \end{pmatrix} := \begin{pmatrix} a_{4l}^{(0)} \\ a_{4l}^{(0)} \end{pmatrix} + \boldsymbol{U}_1^{|n|}(\cdot, 4l+1) \begin{pmatrix} a_{4l+2}^{(0)} \\ a_{4l+3}^{(0)} \end{pmatrix} . \tag{4.10}$$

The degree of the polynomial products in (4.10) is at most 3 so that their computation can be performed by Algorithm 3.1 with $N = 2$, i.e., we compute

$$\left(a_{4l}^{(1)}\left(\cos\frac{s\pi}{4}\right)\right)_{s=0,\dots,4}, \ \left(a_{4l+1}^{(1)}\left(\cos\frac{s\pi}{4}\right)\right)_{s=0,\dots,4} \quad \text{for} \quad l = \lfloor\frac{|n|}{4}\rfloor, \dots, \lceil\frac{M}{4}\rceil - 1.$$

We continue in the obvious manner. In step τ ($1 \le \tau < j$) we use (4.5) with $k = 2^\tau - 1$ and $c = 2^{\tau+1}l + 1$ to compute the values

$$\left(a_{2^{\tau+1}l}^{(\tau)}\left(\cos\frac{s\pi}{2^{\tau+1}}\right)\right)_{s=0,\dots,2^{\tau+1}}, \ \left(a_{2^{\tau+1}l+1}^{(\tau)}\left(\cos\frac{s\pi}{2^{\tau+1}}\right)\right)_{s=0,\dots,2^{\tau+1}}$$

of the polynomials $a_{2^{\tau+1}l}^{(\tau)}, a_{2^{\tau+1}l+1}^{(\tau)} \in \Pi_{2^{\tau+1}-1}$ for $l = \lfloor\frac{|n|}{2^{\tau+1}}\rfloor, \dots, \lceil\frac{M}{2^{\tau+1}}\rceil - 1$. The polynomials are defined by

$$\begin{pmatrix} a_{2^{\tau+1}l}^{(\tau)} \\ \\ a_{2^{\tau+1}l+1}^{(\tau)} \end{pmatrix} := \begin{pmatrix} a_{2^{\tau+1}l}^{(\tau-1)} \\ \\ a_{2^{\tau+1}l+1}^{(\tau-1)} \end{pmatrix} + \boldsymbol{U}_{2^\tau-1}^{|n|}(\,\cdot\,, 2^{\tau+1}l+1) \begin{pmatrix} a_{2^{\tau+1}l+2^\tau}^{(\tau-1)} \\ \\ a_{2^{\tau+1}l+2^\tau+1}^{(\tau-1)} \end{pmatrix}, \tag{4.11}$$

and we apply Algorithm 3.1 (with $N = 2^\tau$) to compute the polynomial products. Note that the values

$$\left(a_{2^{\tau+1}l}^{(\tau-1)}\left(\cos\frac{(2s+1)\pi}{2^\tau}\right)\right)_{s=0,\dots,2^\tau-1}, \ \left(a_{2^{\tau+1}l+1}^{(\tau-1)}\left(\cos\frac{(2s+1)\pi}{2^\tau}\right)\right)_{s=0,\dots,2^\tau-1} \tag{4.12}$$

are not known – they have to be computed from the values

$$\left(a_{2^{\tau+1}l}^{(\tau-1)}\left(\cos\frac{s\pi}{2^\tau}\right)\right)_{s=0,\dots,2^\tau}, \ \left(a_{2^{\tau+1}l+1}^{(\tau-1)}\left(\cos\frac{s\pi}{2^\tau}\right)\right)_{s=0,\dots,2^\tau} .$$

But since $a_{2^{\tau+1}l}^{(\tau-1)}, a_{2^{\tau+1}l+1}^{(\tau-1)} \in \Pi_{2^\tau-1}$ this can be done by DCT-I$(2^\tau+1)$s and DCT-III(2^τ)s as mentioned in (3.1) and (3.2). After step $j-1$ our cascade summation arrives at

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

Now we compute

$$g_n\left(\cos\frac{s\pi}{N}\right) = a_0^{(j-1)}\left(\cos\frac{s\pi}{N}\right) P_0^n\left(\cos\frac{s\pi}{N}\right) + a_1^{(j-1)}\left(\cos\frac{s\pi}{N}\right) P_1^n\left(\cos\frac{s\pi}{N}\right) \tag{4.13}$$

for $s = 0, \dots, N$. Note that $a_0^{(j-1)}, a_1^{(j-1)} \in \Pi_{N-1}$, but we know the values

$$\left(a_0^{(j-1)}\left(\cos\frac{s\pi}{N}\right)\right)_{s=0,\dots,N}, \ \left(a_1^{(j-1)}\left(\cos\frac{s\pi}{N}\right)\right)_{s=0,\dots,N} ,$$

and hence the polynomial product (4.13) is exact.

Stabilization issues

Unfortunately, an implementation of the algorithm presented in the preceding section demonstrates numerical instability for $n > 16$ ($n < -16$ as well). The reason for this is that some of the associated Legendre polynomials $P_k^n(x,c)$ involved in the algorithm become very large for $|x| \approx 1$ while the values of the polynomials $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$ and $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau+1}$ may be arbitrary small for these values of x.

In [18, Theorem 3.1] it is proved that

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

Using an integral representation

$$P_k^n(x) = \frac{2^n n!}{(2n)!}\left(\frac{(k+n)!}{(k-n)!}\right)^{1/2} \frac{\left(1-x^2\right)^{n/2}}{\pi} \int_0^\pi \left(x + \left(x^2-1\right)^{\frac{1}{2}}\cos\theta\right)^{k-n} \sin^{2n}\theta\,\mathrm{d}\theta$$

as given in [23, page 185] we can use Lemma 4.1 to obtain

$$\left(\frac{(k+c+n)!}{(k+c-n)!}\right)^{\frac{1}{2}} = P_k^n(1,c)\left(\frac{(c+n)!}{(c-n)!}\right)^{\frac{1}{2}} + w_c^n P_{k-1}^n(1,c+1)\left(\frac{(c-1+n)!}{(c-1-n)!}\right)^{\frac{1}{2}} \tag{4.14}$$

for $c > n$. With $P_0^n(1,c) = 1$ and $Z_{(n,k,c)} := \left(\frac{(k+c+n)!(c-n)!}{(k+c-n)!(c+n)!}\right)^{\frac{1}{2}}$ induction yields

$$\begin{aligned} P_k^n(1,c) &= Z_{(n,k,c)} \sum_{i=0}^{k}\left(\frac{(c-n-1+i)!\,(c+n)!}{(c-n-1)!\,(c+n+i)!}\right) \\ &= \frac{(n+c)\,Z^2_{(n,k,c)} + n - c}{2nZ_{(n,k,c)}}, \end{aligned} \tag{4.15}$$

which becomes larger for $c \to n,\ k \to \infty$ (for fixed n). The multiplication of these large and small values results in very large and small function values for the polynomials $a^{(\tau)}_{2^{\tau+1}l}$ (see Figure 4.2).

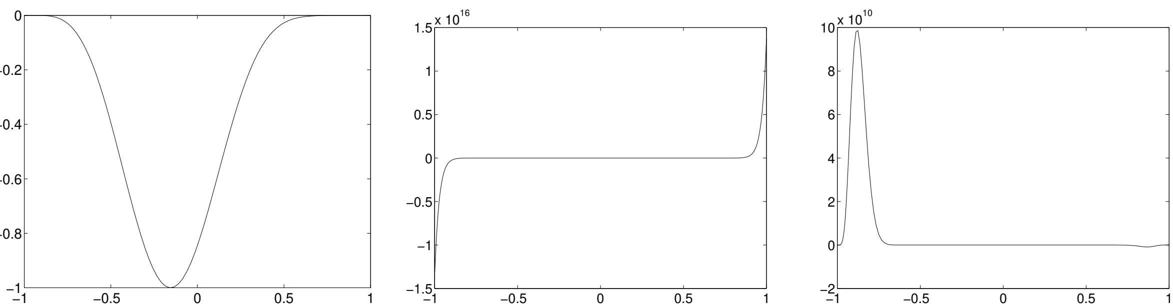

Figure 4.2: Polynomial $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$, associated Legendre polynomial $P^n_{2^\tau-1}\left(\,\cdot\,,2^{\tau+1}l+1\right)$ and the product $P^n_{2^\tau-1}\left(\,\cdot\,,2^{\tau+1}l+1\right)a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$ for values $\tau = 5,\ n = 64$ and $l = 1$.

The numerical problems result from the computation of the intermediate values of the polynomials $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$. Global interpolation methods like FFT–based or DCT–based algorithms (see [6, 26, 14] resp. Algorithm 3.1) or FMM–based algorithms (see [32] resp. Algorithm 3.2) uniformly cause an error in $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$, i.e., a small relative error in $\max_{|x|\le 1} |a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}(x)|$ is added uniformly to $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$. Thus, computing the product $P^n_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$ leads to a huge relative error.

We illustrate the error propagation by a simple example. Figure 4.3 (left) shows the relative error in the perturbed polynomial $\tilde{a}^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$ which is amplified in the product by orders of magnitude, see Figure 4.3 (right). The increase of the relative error is caused by the huge values of $P^n_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right)$ for $|x| \approx 1$.

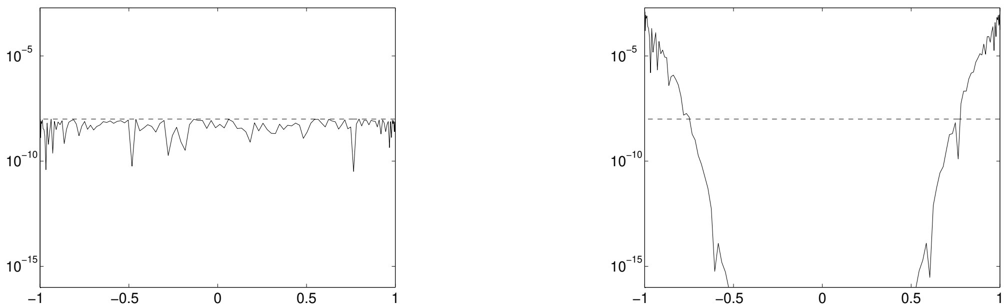

Figure 4.3: Error propagation for computation of intermediate values based on a global interpolation method, the maximum allowable error is 10^{-8} (dashed lines), as above $\tau = 5$, $n = 64$ and $l = 1$; the relative error $\left\| a^{(\tau-1)}_{2^{\tau+1}l+2^\tau} \right\|^{-1}_\infty \left| \tilde{a}^{(\tau-1)}_{2^{\tau+1}l+2^\tau} - a^{(\tau-1)}_{2^{\tau+1}l+2^\tau} \right|$ (left) and the relative error $\left\| P^n_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau} \right\|^{-1}_\infty \left| P^n_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right) \tilde{a}^{(\tau-1)}_{2^{\tau+1}l+2^\tau} - P^n_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau} \right|$ of the product (right).

Consequently, the simplest idea consists of replacing the ordinary cascade summation step by “special” stabilization steps whenever the values $|P^n_k(1, c)|$ involved in the algorithm cross some threshold. This straightforward idea was first formulated in [26]. To avoid the multiplications with large values, the multiplications with $\boldsymbol{U}^n_{2^\tau-1}(\,\cdot\,, 2^{\tau+1}l+1)$ were originally replaced by multiplications with $\boldsymbol{U}^n_{2^\tau(2l+1)-1}(\,\cdot\,, 1)$, which fulfills

$$\boldsymbol{U}^n_{2^\tau(2l+1)-1}(\,\cdot\,, 1) \;=\; \boldsymbol{U}^n_{2^{\tau+1}l-1}(\,\cdot\,, 1)\, \boldsymbol{U}^n_{2^\tau-1}(\,\cdot\,, 2^{\tau+1}l+1).$$

The entries of $\boldsymbol{U}^n_{2^\tau(2l+1)-1}(\,\cdot\,, 1)$ are significantly smaller than those of $\boldsymbol{U}^n_{2^\tau-1}(\,\cdot\,, 2^{\tau+1}l+1)$ for $|x| \approx 1$. Of course, the high–order zeros of $\boldsymbol{U}^n_{2^{\tau+1}l-1}(\,\cdot\,, 1)$ for $|x| = 1$ are the reason for this fact. Furthermore, Lemma 4.1 implies

$$\begin{pmatrix} P^n_{2^{\tau+1}l+2^\tau} \\ P^n_{2^{\tau+1}l+2^\tau+1} \end{pmatrix} = \boldsymbol{U}^n_{2^\tau(2l+1)}(\,\cdot\,, 1)^{\mathrm{T}} \begin{pmatrix} P^n_0 \\ P^n_1 \end{pmatrix}.$$

Therefore, the stabilization step is equivalent to

$$g_n^{\text{stab (new)}} = g_n^{\text{stab (old)}} + a^{(\tau-1)}_{2^{\tau+1}l+2^\tau} P^{|n|}_{2^{\tau+1}l+2^\tau} + a^{(\tau-1)}_{2^{\tau+1}l+2^\tau+1} P^{|n|}_{2^{\tau+1}l+2^\tau+1}.$$

In other words, instead of using (4.11), we evaluate the polynomials which would cause numerical problems in $\mathcal{O}(M\log M)$ floating point operations. As usual we also have to compute the values (4.12) in $\mathcal{O}(2^\tau\tau)$ operations and set

$$\begin{pmatrix} a^{(\tau)}_{2^{\tau+1}l} \\ a^{(\tau)}_{2^{\tau+1}l+1}\end{pmatrix} := \begin{pmatrix} a^{(\tau-1)}_{2^{\tau+1}l} \\ a^{(\tau-1)}_{2^{\tau+1}l+1}\end{pmatrix}.$$

For the purpose of performing a complexity analysis it would be necessary to know an upper bound for

$$s := \#\{(c,k,n) :\ P_k^n(1,c) \text{ is used in Algorithm 4.1 and } P_k^n(1,c) \text{ exceeds the threshold}\}.$$

If $s=\mathcal{O}(\log M)$ our stabilization technique would still lead to an $\mathcal{O}\left(M\log^2 M\right)$ algorithm.

We summarize the results including the stabilization, in Algorithm 4.1.

Algorithm 4.1 Fast Legendre function transform

Input: $M\in\mathbb{N}_0$, $n\in\mathbb{Z}$ ($|n|\le M$), $(a_k^n)_{k=|n|,\ldots,M}$

Precompute: $j:=\lceil\log_2 M\rceil$, $N:=2^j$ and $\boldsymbol{U}^{|n|}_{2^\tau-1}\left(\cos\frac{s\pi}{2^{\tau+1}}, 2^{\tau+1}l+1\right)$
for $\tau=1,\ldots,j-1$, $l=\lfloor\frac{|n|}{2^{\tau+1}}\rfloor,\ldots,\lceil\frac{M}{2^{\tau+1}}\rceil-1$ and $s=0,\ldots,2^{\tau+1}$
$\left(P^{|n|}_{2^\tau(2l+1)}\left(\cos\frac{s\pi}{2^j}\right)\right)_{s=0,\ldots,N}$ and $\left(P^{|n|}_{2^\tau(2l+1)}\left(\cos\frac{s\pi}{2^j}\right)\right)_{s=0,\ldots,N}$
for the stabilization steps

Compute $\left(a_k^{(0)}\right)_{k=0,\ldots,N-1}$ using (4.9)
for $\tau=1,\ldots,j-1$ **do**
  for $l=\lfloor\frac{|n|}{2^{\tau+1}}\rfloor,\ldots,\lceil\frac{M}{2^{\tau+1}}\rceil-1$ **do**
    Compute intermediate values of $a^{(\tau-1)}_{2^{\tau+1}l}$, $a^{(\tau-1)}_{2^{\tau+1}l+1}$ using DCTs, see (4.12)
    if multiplication with $\boldsymbol{U}^{|n|}_{2^\tau-1}\left(\cdot, 2^{\tau+1}l+1\right)$ is stable **then**
      Compute the values of $a^{(\tau)}_{2^{\tau+1}l}$, $a^{(\tau)}_{2^{\tau+1}l+1}$ in (4.11) with Algorithm 3.1
    else
      Compute $g_n^{\text{stab (new)}} = g_n^{\text{stab (old)}} + a^{\tau-1}_{2^\tau(2l+1)}P^{|n|}_{2^\tau(2l+1)} + a^{\tau-1}_{2^\tau(2l+1)+1}P^{|n|}_{2^\tau(2l+1)+1}$ using a method similar to Algorithm 3.1 (DCT–Is and zero–padding)
      Update $a^{(\tau)}_{2^{\tau+1}l} = a^{(\tau-1)}_{2^{\tau+1}l}$ and $a^{(\tau)}_{2^{\tau+1}l+1} = a^{(\tau-1)}_{2^{\tau+1}l+1}$
    end if
  end for
end for
Compute $(g_{s,n})_{s=0,\ldots,N} = \left(g_n\left(\cos\frac{s\pi}{N}\right)\right)_{s=0,\ldots,N}$ using (4.13)
Compute $(g_{s,n})_{s=0,\ldots,N} = (g_{s,n})_{s=0,\ldots,N} + \left(g^{\text{stab}}_{s,n}\right)_{s=0,\ldots,N}$

Output: $(g_{s,n})_{s=0,\ldots,N}$

Remark 4.2. If we replace the fast polynomial multiplication in Algorithm 4.1 by Algorithm 3.2 we end up with an approximate algorithm similar to the one given in [32]. Note that in Algorithm 3.2 the polynomial multiplication is not restricted to the points $\cos\frac{s\pi}{2^{\tau+1}}$. Hence, one can improve the numerical stability by choosing different sampling points (see [32]). $\square$

Remark 4.3. If we realize the polynomial multiplication in Section 3 by the FFT, we obtain the transposted version of the Driscoll-Healy algorithm [6]. In [7] the autors used also the DCT. See [25] how this algorithms are related. □

5 Discrete spherical Fourier transform

In this section, we propose an algorithm to evaluate band–limited functions on the sphere, given by their spherical Fourier coefficients, at arbitrary nodes. More precisely, given the spherical Fourier coefficients $(a_k^n)_{(k,n)\in I^M} \in \mathbb{C}^{(M+1)^2}$ $(M \in \mathbb{N})$ and nodes $(\vartheta_d, \varphi_d)_{d=0,\ldots,D-1} \in S^D$ $(D \in \mathbb{N})$, we are interested in the fast computation of

$$\begin{aligned}
(f(\vartheta_d,\varphi_d))_{d=0,\ldots,D-1} &= \left(\sum_{(k,n)\in I^M} a_k^n(f) Y_k^n(\vartheta_d,\varphi_d)\right)_{d=0,\ldots,D-1} \\
&= \left(\sum_{k=0}^{M}\sum_{n=-k}^{k} a_k^n Y_k^n(\vartheta_d,\varphi_d)\right)_{d=0,\ldots,D-1} \\
&= \left(\sum_{n=-M}^{M} h_n(\cos\vartheta_d)\,\mathrm{e}^{\mathrm{i}n\varphi_d}\right)_{d=0,\ldots,D-1},
\end{aligned} \tag{5.1}$$

with

$$h_n(\cos\vartheta) := \begin{cases} g_n(\cos\vartheta) & \text{for even } n, \\ (\sin\vartheta)\, g_n(\cos\vartheta) & \text{otherwise,} \end{cases} \tag{5.2}$$

where we consider the g_n given in equations (4.6) and (4.7) with complex coefficients.

5.1 DSFT on special grids

We start with the problem of evaluating f for given spherical Fourier coefficients $(a_k^n)_{(k,n)\in I^M}$ on the grid $(\vartheta_s,\varphi_t)_{s=0,\ldots,N,\, t=0,\ldots,2N-1} \in S^{2N(N+1)}$ with $\vartheta_s := \frac{s\pi}{N}$, $\varphi_t := \frac{t\pi}{2N}$ and $N := 2^{\lceil \log_2 M\rceil}$. By separation of the variables and evaluation of

$$(h_{s,n})_{s=0,\ldots,N} := (h_n(\vartheta_s))_{s=0,\ldots,N} \tag{5.3}$$

for $n = -M,\ldots,M$, followed by Fourier transformations

$$(f(\vartheta_s,\varphi_t))_{t=0,\ldots,2N-1} := \left(\sum_{n=-M}^{M} h_{s,n}\mathrm{e}^{\mathrm{i}n\varphi_t}\right)_{t=0,\ldots,2N-1} \tag{5.4}$$

for $s = 0,\ldots,N$, we obtain an $\mathcal{O}\left(M^2N\right)$ algorithm. By using the fact that $P_k^{|n|}$ is an even (odd) function when $n-k$ is even (odd) and that $\cos\vartheta_s = -\cos\vartheta_{N-s}$, we save half the arithmetical operations. We use two modified Clenshaw algorithms to compute

$$\left(h_{s,n}^{\text{even}}\right)_{s=0,\dots,\frac{N}{2}} := \left(\sum_{\substack{k=|n|,\\ k-n \text{ even}}}^{M} a_k^n P_k^{|n|}(\cos\vartheta_s)\right)_{s=0,\dots,\frac{N}{2}}, \tag{5.5}$$

$$\left(h_{s,n}^{\text{odd}}\right)_{s=0,\dots,\frac{N}{2}} := \left(\sum_{\substack{k=|n|,\\ k-n \text{ odd}}}^{M} a_k^n P_k^{|n|}(\cos\vartheta_s)\right)_{s=0,\dots,\frac{N}{2}} \tag{5.6}$$

and finally set

$$h_{s,n} = \begin{cases} h_{s,n}^{\text{even}} + h_{s,n}^{\text{odd}} & \text{for } s = 0,\dots,\frac{N}{2}, \\ h_{N-s,n}^{\text{even}} - h_{N-s,n}^{\text{odd}} & \text{otherwise,} \end{cases} \tag{5.7}$$

instead of performing the direct computation in (5.3).

Remark 5.1. This technique is known as the Parity-Exploiting Matrix Multiplication (PMMT) and is used in the T639 code of the European Center for Medium–Range Weather Forecasting, see [24]. □

However, we can obtain an asymptotically faster transform for (5.3) by using the fast associated Legendre transforms to compute $(g_{s,n})_{s=0,\dots,N}$ (see definition (4.8)) for $n = -M,\dots,M$ and applying (5.2). The longitudinal steps (5.4) are just conventional fast Fourier transforms (FFTs) for $s = 0,\dots,N$. Because (5.3) can be computed more quickly for small M using the modified Clenshaw algorithm, we implemented this DSFT as follows:

- Algorithm 3.1 makes use of direct cosine transforms up to a length of 64 (see [29]) and of an implementation following [1] otherwise.
- During precomputation, test for every $n = -M,\dots,M$ whether $(h_{s,n})_{s=0,\dots,N}$ can be evaluated more quickly using Algorithm 4.1 or using the modified Clenshaw algorithm.

We obtain an exact DSFT algorithm (Algorithm 5.1) for the grids $(\vartheta_s,\varphi_t)_{s=0,\dots,N,\, t=0,\dots,2N-1}$. Oversampling can be carried out by using zero–padding techniques for the latitudinal as well as for the longitudinal direction.

5.2 Slow spherical Fourier transform on arbitrary nodes

We obtained a direct $\mathcal{O}\left(M^2N\right)$ algorithm instead of $\mathcal{O}\left(M^2N^2\right)$ in the last section by a separation of variables over the grid. For arbitrary nodes, we only have an $\mathcal{O}\left(M^2D\right)$ algorithm, where D is the number of nodes (see Algorithm 5.2 and see [5, pp. 402]).

5.3 Fast spherical Fourier transform on arbitrary nodes

Given the problem (5.1) we first use Algorithm 4.1 to change the basis, a step which is independent of the nodes. After this, we use the fast Fourier transform for nonequispaced data (NFFT) to compute the values of the function at arbitrary nodes. Note that the fast Fourier transform requires sampling on an equally spaced grid, which represents a significant limitation for many applications. The aim of the NFFT is to overcome this drawback. The NFFT can be realized

Algorithm 5.1 DSFT on equispaced grids in S

Input: $M \in \mathbb{N},\ (a_k^n)_{(k,n)\in I^M} \in \mathbb{C}^{(M+1)^2}$
Precompute: $j := \lceil \log_2 M \rceil,\ N := 2^j$,
$\theta_s := \frac{s\pi}{N}\ (s = 0, \ldots, N),\ \varphi_t := \frac{t\pi}{2N}\ (t = 0, \ldots, 2N-1)$,
see additional precomputation of Algorithm 4.1

Set $h_{s,n} := 0$ for $s = 0, \ldots, N$ and $n = -N, \ldots, N$
for $n = -M, \ldots, M$ **do**
  if direct computation using (5.7) is faster **then**
    Compute $(h_{s,n})_{s=0,\ldots,N}$ using the modified Clenshaw algorithm
  else
    Compute $(g_{s,n})_{s=0,\ldots,N}$ using Algorithm 4.1
    if n odd **then**
      Compute $(h_{s,n})_{s=0,\ldots,N} = \left(\left(\sin \frac{s\pi}{N}\right) g_{s,n}\right)_{s=0,\ldots,N}$
    else
      Set $(h_{s,n})_{s=0,\ldots,N} = (g_{s,n})_{s=0,\ldots,N}$
    end if
  end if
end for
for $s = 0, \ldots, N$ **do**
  Compute $(f(\vartheta_s, \varphi_t))_{t=0,\ldots,2N-1}$ using a FFT$((h_{s,n})_{n=-N,\ldots,N-1})$ of length $2N$
  Compute $(f(\vartheta_s, \varphi_t))_{t=0,\ldots,2N-1} = (f(\vartheta_s, \varphi_t))_{t=0,\ldots,2N-1} + \left(h_{s,N} \mathrm{e}^{\mathrm{i}N\varphi_t}\right)_{t=0,\ldots,2N-1}$
end for

Output: $(f(\vartheta_s, \varphi_t))_{s=0,\ldots,N,\ t=0,\ldots,2N-1}$

Algorithm 5.2 direct computation of (5.1)

Input: $M \in \mathbb{N},\ (a_k^n)_{(k,n)\in I^M} \in \mathbb{C}^{(M+1)^2},\ D \in \mathbb{N},\ (\vartheta_d, \varphi_d)_{d=0,\ldots,D-1} \in S^D$

for $d = 0, \ldots, D-1$ **do**
  for $n = -M \ldots M$ **do**
    Compute $h_{d,n} = \sum\limits_{k=|n|}^{M} a_k^n P_k^{|n|}(\cos \vartheta_d)$ using the Clenshaw algorithm
  end for
  Compute $f(\vartheta_d, \varphi_d) = \sum\limits_{n=-M}^{M} h_{d,n} \mathrm{e}^{\mathrm{i}n\varphi_d}$
end for

Output: $(f(\vartheta_d, \varphi_d))_{d=0,\ldots,D-1}$

in an efficient way by approximating trigonometric polynomilas by the sum of translates of a 1–periodic function φ with good localization in time and frequency. Beylkin et al. [3, 4] prefer B–splines and Rokhlin et al. [9] Gaussians. By the results in [11, 22, 10] we prefer to apply the NFFT with Kaiser–Bessel functions. Details concerning NFFT algorithms can be found for example in [27] and a software package can be found in [17].

Details concerning the NFFT algorithms can be found in [27] for example.

For arbitrary $(\vartheta,\varphi) \in S$ we have

$$f(\vartheta,\varphi) = \sum_{n=-M}^{M} h_n(\cos\vartheta)\,\mathrm{e}^{\mathrm{i}n\varphi}$$

where the h_n are as given by (5.2). By using Algorithm 4.1 we compute the values

$$(g_{s,n})_{s=0,\ldots,N} \qquad (N = 2^{\lceil \log_2 M \rceil})$$

(the polynomial part of h_n) for $n=-M,\ldots,M$ as given by (4.8). Furthermore, we obtain the Chebyshev coefficients $(\tilde{a}_k^n)_{k=0,\ldots,N} \in \mathbb{C}^{N+1}$ in

$$g_n(\cos\vartheta) = \sum_{k=0}^{M} \tilde{a}_k^n T_k(\cos\vartheta)$$

for even n and

$$g_n(\cos\vartheta) = \sum_{k=0}^{M-1} \tilde{a}_k^n T_k(\cos\vartheta)$$

for odd n by using equation (3.1). Note that g_n are polynomials of degree M or $M-1$ for even or odd n, respectively. Rewriting g_n by using

$$T_k(\cos\vartheta) = \cos(k\vartheta) = \frac{1}{2}\left(\mathrm{e}^{\mathrm{i}k\vartheta} + \mathrm{e}^{-\mathrm{i}k\vartheta}\right)$$

leads to a truncated Fourier series

$$g_n(\cos\vartheta) = \sum_{k=-(M-1)}^{M-1} b_k^n \mathrm{e}^{\mathrm{i}k\vartheta}$$

with

$$b_k^n := \begin{cases} \tilde{a}_{|k|}^n & \text{for } k=0, \\ \frac{\tilde{a}_{|k|}^n}{2} & \text{otherwise.} \end{cases} \tag{5.8}$$

For odd n, we have

$$\begin{aligned} \sin(\vartheta) \sum_{k=-(M-1)}^{M-1} b_k^n \mathrm{e}^{\mathrm{i}k\vartheta} &= \frac{1}{2\mathrm{i}}\left(\mathrm{e}^{\mathrm{i}\vartheta} - \mathrm{e}^{-\mathrm{i}\vartheta}\right) \sum_{k=-(M-1)}^{M-1} b_k^n \mathrm{e}^{\mathrm{i}k\vartheta} \\ &= \sum_{k=-M}^{M} \tilde{b}_k^n \mathrm{e}^{\mathrm{i}k\vartheta} \end{aligned}$$

with

$$\tilde{b}_k^n := \begin{cases} \frac{-b_{k+1}^n}{2\mathrm{i}} & \text{for } k=-M,-M+1, \\ \frac{b_{k-1}^n}{2\mathrm{i}} & \text{for } k=M-1,M, \\ \frac{b_{k-1}^n - b_{k+1}^n}{2\mathrm{i}} & \text{otherwise.} \end{cases} \tag{5.9}$$

We obtain

$$h_n(\cos\vartheta) = \sum_{k=-M}^{M} c_k^n \mathrm{e}^{\mathrm{i}k\vartheta}$$

with

$$c_k^n := \begin{cases} b_k^n & \text{for even } n, \\ \tilde{b}_k^n & \text{for odd } n\,. \end{cases} \tag{5.10}$$

Thus, we have

$$f(\vartheta,\varphi) \;=\; \sum_{n=-M}^{M}\sum_{k=-M}^{M} c_k^n \mathrm{e}^{\mathrm{i}k\vartheta}\mathrm{e}^{\mathrm{i}n\varphi}. \tag{5.11}$$

The given spherical Fourier coefficients in (5.1) are transformed to the (ordinary) Fourier coefficients in (5.11) by an exact $\mathcal{O}\left(M^2\log^2 M\right)$ algorithm (independent of the nodes (ϑ_d,φ_d)). The geometrical interpretation is that the sphere is mapped to the "outer half" of the torus while the inner half is continued smoothly (see Figure 5.1). First we divide f into an even and

Figure 5.1: Topography of the earth on the sphere (left) and as 'outer half' of a torus (right).

an odd part (relative to the poles) and construct an even and odd continuation on the torus, respectively. Figure 5.2 illustrates this fact. In a final step we evaluate f for arbitrary nodes using the bivariate Fourier transform for nonequispaced data (NFFT, see [27]). The arithmetical complexity of Algorithm 5.3 is given by $\mathcal{O}\left(M^2\log^2 M + m^2 D\right)$, where m is a cut–off parameter, see Algorithm 12.1 in [27, page 251] for details.

6 Numerical results

We implemented the presented algorithms in C and tested them in two systems
(I) Intel-Celeron 64MB RAM, SuSe-Linux and
(II) Sun-SPARC 768MB RAM, SunOS 5.6.
We chose the threshold for stabilization in Algorithm 4.1 as 100000. The first tests evaluate time complexity and accuracy of the new algorithms in comparison to the direct computation while the second test demonstrates an application to the EGM96 data. This is a global model of the earth's gravitational potential that includes orthogonal coefficients up to a degree and

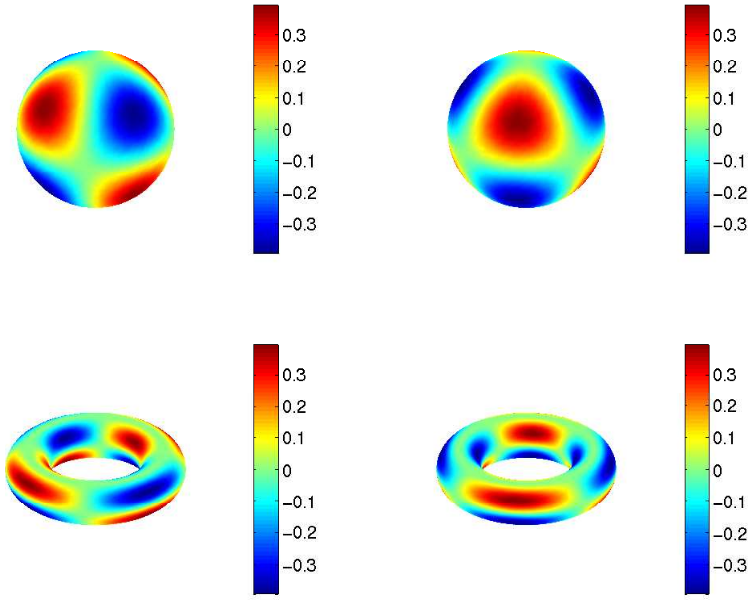

Figure 5.2: Plots of the normalized spherical harmonic $Y_3^{-2} := \frac{1}{4}\sqrt{\frac{105}{2\pi}}\sin^2\theta\cos\theta\mathrm{e}^{-2\mathrm{i}\varphi}$, top: real part (left) and imaginary part (right), bottom: mapping to the torus, the "inner half" is an even continuation with respect to the north/south pole.

Algorithm 5.3 DSFT for arbitrary nodes

Input: $M \in \mathbb{N}$, $(a_k^n)_{(k,n)\in I^M} \in \mathbb{C}^{(M+1)^2}$, $m, D \in \mathbb{N}$, $\alpha \in \mathbb{R}$, $(\vartheta_d, \varphi_d)_{d=0,\ldots,D-1} \in S^D$
Precompute: $N := 2^{\lceil \log_2 M \rceil}$,
see additional precomputation of Algorithm 4.1

for $n = -M, \ldots, M$ **do**
  Compute $(g_{s,n})_{s=0,\ldots,N}$ using Algorithm 4.1
  Compute $(\tilde{a}_k^n)_{k=0,\ldots,N}$ using equation (3.1)
  Compute $(c_k^n)_{k=-M,\ldots,M}$ using equation (5.8),(5.9) and (5.10)
end for
Compute $(f(\vartheta_d, \varphi_d))_{d=0,\ldots,D-1}$ from the Fourier coefficients $(c_k^n)_{k,n=-M,\ldots,M}$ using a bivariate NFFT at the nodes $(\vartheta_d, \varphi_d)_{d=0,\ldots,D-1}$ (see [27])

Output: $(f(\vartheta_d, \varphi_d))_{d=0,\ldots,D-1}$

---

order of 360 (see [19]).
First we chose random data $a_k^n \in [0,1]$ and random nodes $(\vartheta_d, \varphi_d) \in S$. Table 6.1 compares the elapsed CPU time for Algorithm 5.2 (direct) and Algorithm 5.3 (new) for different bandwidths and different numbers of nodes (oversampling factor $\alpha = 2$, cut–off parameter $m = 4$ with a Gaussian kernel, see Algorithm 12.1 in [27, page 251] for details). Furthermore, we determined

M	64		128		256	
D	direct	new	direct	new	direct	new
1	0.01	0.37	0.01	1.80	0.01	8.61
10	0.01	0.38	0.05	1.82	0.20	8.61
100	0.13	0.42	0.49	1.84	1.82	8.67
1000	1.27	0.43	4.82	1.88	18.24	8.71
10000	12.61	0.53	49.04	1.96	176.41	8.81
20000	25.44	0.65	98.20	2.09	352.75	8.95

Table 6.1: CPU time in seconds on system (I).

the relationship between the cut–off parameter m and the relative error

$$\varepsilon := \frac{\left\| \left(f_{\text{approx}}\left(\vartheta_d, \varphi_d\right) - f_{\text{exact}}\left(\vartheta_d, \varphi_d\right)\right)_{d=1,\ldots,D} \right\|_\infty}{\left\| \left(f_{\text{exact}}\left(\vartheta_d, \varphi_d\right)\right)_{d=1,\ldots,D} \right\|_\infty}$$

where $f_{\text{approx}}\left(\vartheta_d, \varphi_d\right)$ denotes the value computed by Algorithm 5.3 and $f_{\text{exact}}\left(\vartheta_d, \varphi_d\right)$ denotes the one computed by Algorithm 5.2. We have an exponential decay as shown in Table 6.2 and proved in [27]. The third test concerns the time complexity with respect to a given bandwidth

m	1	2	3	4	5	6	7	8
ε	5.0e-02	7.7e-03	3.0e-04	1.9e-05	7.1e-06	5.8e-07	5.1e-08	2.3e-08

Table 6.2: Relative error ε on system (I) ($M = 128$, $\alpha = 2$ and $D = 100$).

and an approximate number of nodes. We chose the number of nodes as $D = M^2$; note that for $M \geq 200$, the **direct** computation was run using a fixed number of nodes ($D = 100$) , and the required CPU–time was estimated as $t = \frac{M^2 t_{\text{used}}}{100}$. Figure 6.1 shows a log-plot of elapsed CPU time. Note that for a bandwidth of $M = 500$ our fast algorithm requires 92 seconds but the direct computation takes almost 7 hours.

Further tests concern an application of Algorithm 5.3 to the earth's geopotential model (EGM96). Figure 6.2 and Figure 6.3 show a reconstruction on the whole sphere while Figure 6.4 shows the so–called "zoom-in" property. Given a global model we are able to zoom-in to an arbitrary region for arbitrary combinations of spherical frequencies.

7 Conclusion

While a direct computation of (1.1) for arbitrary nodes has a computational complexity of $\mathcal{O}\left(DM^2\right)$ we were able to show that $\mathcal{O}\left(M^2 \log^2 M + \log^2(1/\varepsilon)D\right)$ arithmetical operations are sufficient. Given a band–limited function on the sphere and remembering its "equiareal resolution" provided by the addition theorem for spherical harmonics (see e.g. [5, p.400]) it does not seem suitable to use the standard grid. Especially in time-stepping algorithms where one has to deal with the so–called pole problem an "almost equiareal" grid together with a convenient quadrature rule and the suggested Algorithm 5.3 would enable fast versions of these algorithms.

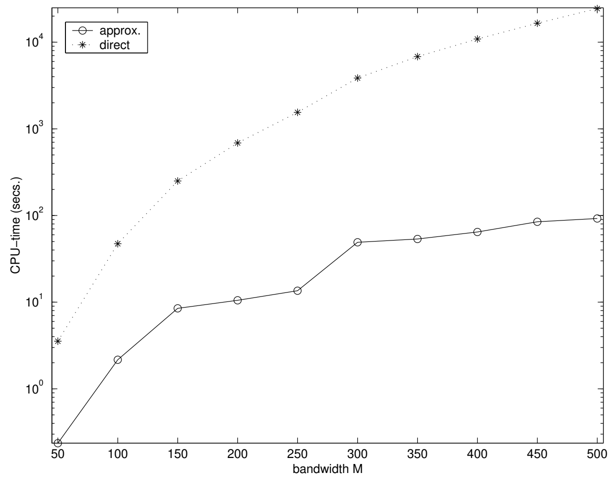

Figure 6.1: CPU–time on system (II) ($\alpha = 2$, $m = 4$).

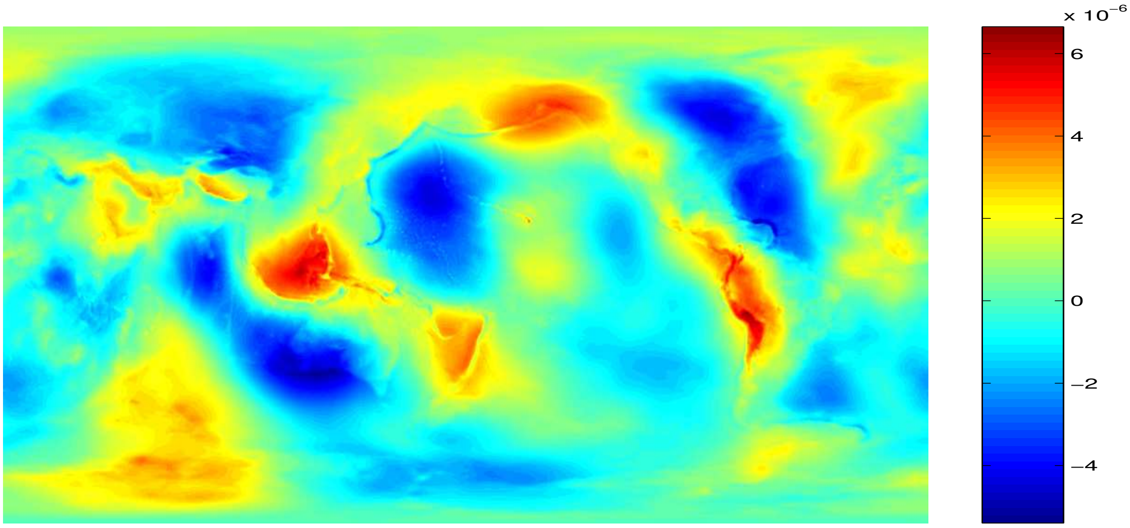

Figure 6.2: Reconstruction of EGM96 data for $k \geq 4$, normalization to orthonormal spherical harmonics.

Note that the choice of nodes in the algorithm in [32] is arbitrary only within an individual coordinate.

Furthermore we were able to show which algorithms are the essential parts of the spherical Fourier transform (also for the generalized setting of arbitrary nodes) and that they are independent of each other. Almost certainly, any fast spherical Fourier algorithm will use a decomposition like the one in Figure 4.1, fast polynomial transforms and (standard) FFTs. Therefore, it seems worth to investigate the associated Legendre polynomials $P_k^{|n|}(x, c)$ and their numerical behavior more deeply.

In [21] the author uses a local approximative method for computing the fast spherical harmonic transform. We will investigate local interpolation methods in combination with our simple

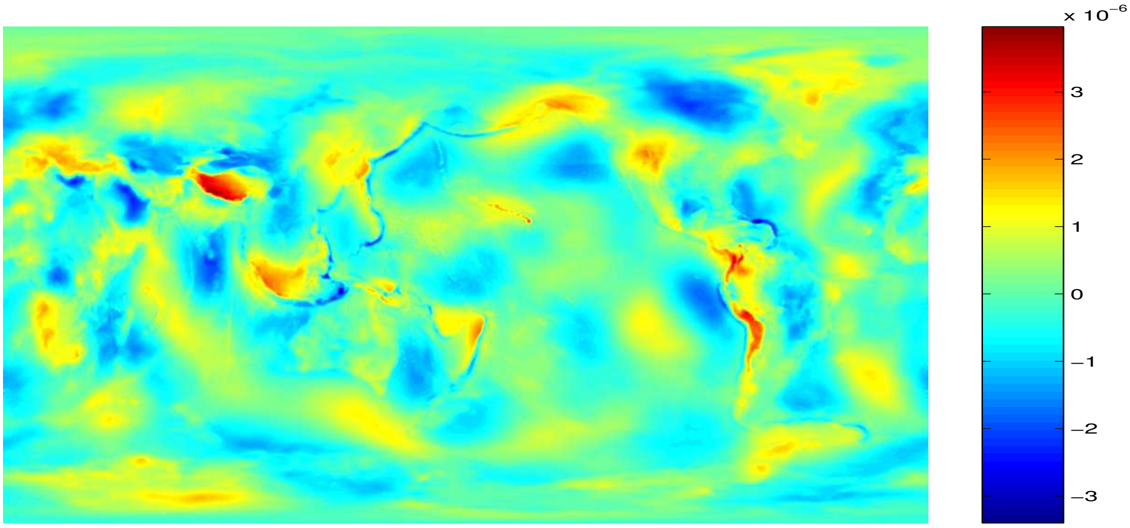

Figure 6.3: Reconstruction of EGM96 data for $k \geq 8$, normalization to orthonormal spherical harmonics.

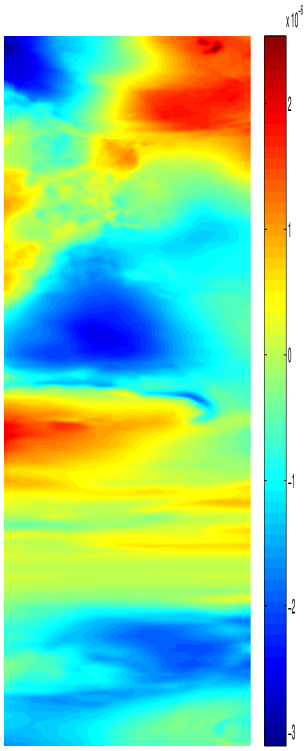

Figure 6.4: Zoom in (eastern part of South–America, Figure 6.2).

cascade summation in a forthcoming paper. Most likely, local interpolation methods will add an error weighted by the actual value $a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}(x)$, illustrated by Figure 7.1 (left). Thus, computing the product $P^n_{2^\tau-1}\left(\,\cdot\,, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$ leads only to a small relative error, see Figure 7.1 (right).

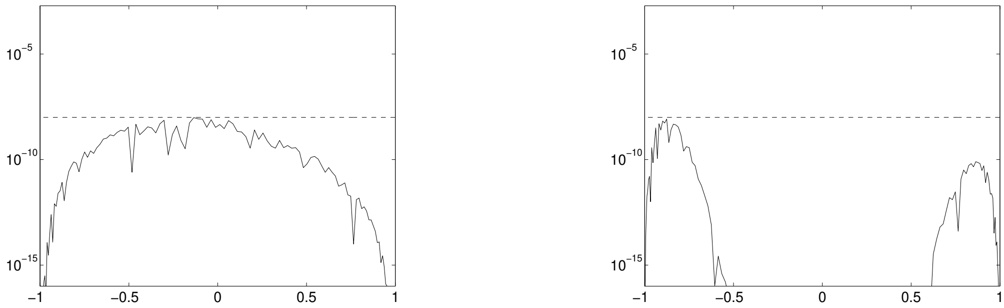

Figure 7.1: Expected error propagation for computation of intermediate values based on local methods, the maximum allowable error is 10^{-8} (dashed lines), as above $\tau = 5$, $n = 64$ and $l = 1$, the perturbed polynomial is denoted by $\tilde{\tilde{a}}^{(\tau-1)}_{2^{\tau+1}l+2^\tau}$; the relative error $\left\|a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}\right\|_\infty^{-1} \left|\tilde{\tilde{a}}^{(\tau-1)}_{2^{\tau+1}l+2^\tau} - a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}\right|$ (left) and the relative error $\left\|P^n_{2^\tau-1}\left(\,\cdot\,, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}\right\|_\infty^{-1} \left|P^n_{2^\tau-1}\left(\,\cdot\,, 2^{\tau+1}l+1\right) \tilde{\tilde{a}}^{(\tau-1)}_{2^{\tau+1}l+2^\tau} - P^n_{2^\tau-1}\left(\,\cdot\,, 2^{\tau+1}l+1\right) a^{(\tau-1)}_{2^{\tau+1}l+2^\tau}\right|$ of the product (right).

Acknowledgment. We would also like to thank the referees for their valuable suggestions.

References

[1] G. Baszenski. Programmpaket zur Berechnung diskreter trigonometrischer Transformationen. 1995. http://www.iuk.fh-dortmund.de/~baszenski/.

[2] G. Baszenski and M. Tasche. Fast polynomial multiplication and convolution related to the discrete cosine transform. *Linear Algebra Appl.*, 252:1 – 25, 1997.

[3] G. Beylkin. On the fast Fourier transform of functions with singularities. *Appl. Comput. Harmon. Anal.*, 2:363 – 381, 1995.

[4] G. Beylkin and R. Cramer. A multiresolution approach to regularization of singular operators and fast summation. SIAM *J. Sci. Comput.*, 24:81 – 117, 2002.

[5] J. P. Boyd. *Chebyshev and Fourier Spectral Methods.* Dover Press, New York, second edition, 2000.

[6] J. Driscoll and D. Healy. Computing Fourier transforms and convolutions on the 2–sphere. *Adv. Appl. Math.*, 15:202 – 250, 1994.

[7] J. Driscoll, D. Healy, and D. Rockmore. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. SIAM *J. Comput.*, 26:1066 – 1099, 1996.

[8] A. Dutt, M. Gu, and V. Rokhlin. Fast algorithms for polynomial interpolation, integration and differentiation. SIAM *J. Numer. Anal.*, 33:1689 – 1711, 1996.

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

[10] J. A. Fessler and B. P. Sutton. Nonuniform fast Fourier transforms using min-max interpolation. IEEE *Trans. Signal Process.*, 51:560 – 574, 2003.

[11] K. Fourmont. Schnelle Fourier–Transformation bei nichtäquidistanten Gittern und tomographische Anwendungen. Dissertation, Universität Münster, 1999.

[12] W. Freeden, T. Gervens, and M. Schreiner. *Constructive Approximation on the Sphere.* Oxford University Press, Oxford, 1998.

[13] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325 – 348, 1987.

[14] D. Healy, P. Kostelec, S. S. B. Moore, and D. Rockmore. FFTs for the 2-sphere – Improvements and variations. *J. Fourier Anal. Appl.*, 9, 2003.

[15] D. Healy, S. S. B. Moore, and D. Rockmore. Efficiency and stability issues in the numerical computation of Fourier transforms and convolutions on the 2-Sphere. Technical report, Dartmouth College, Hanover, 1994.

[16] G. Heinig and K. Rost. Representations of Cauchy matrices with Chebyshev nodes using trigonometric transformations. In D. Bini, E. Tyrtyshnikov, and P. Yalamov, editors, *Structured Matrices: Recent Developments in Theory and Computation*, volume 4 of *Advances in Computation: Theory and Practice*, pages 135 – 148, Nova Science Inc., 2001.

[17] S. Kunis and D. Potts. NFFT, Softwarepackage, C subroutine library. http://www.math.uni-luebeck.de/potts/nfft, 2002.

[18] R. Lasser. Orthogonal polynomials hypergroups ii – the symmetric case. *Tran. Amer. Math. Soc.*, 341:749 – 770, 1994.

[19] F. G. Lemoine and E. C. Pavlis et.al. The development of the joint NASA GSFC and NIMA geopotential model EGM96. Technical report, NASA Goddard Space Flight Center, Greenbelt, Maryland, 20771 USA, 1998.

[20] H. Mhaskar, F. Narcowich, and J. Ward. Quadrature formulas on spheres using scattered data. *Math. Comput.*, to appear.

[21] M. J. Mohlenkamp. A fast transform for spherical harmonics. *J. Fourier Anal. Appl.*, 5:159 – 184, 1999.

[22] A. Nieslony and G. Steidl. Approximate factorizations of Fourier matrices with nonequispaced knots. *Linear Algebra Appl.*, to appear.

[23] F. W. J. Olver. *Asymptotics and Special Functions.* Academic Press, 1974.

[24] A. Persson. User guide to ecmwf forecast products. Technical report, European Center for Medium–Range Weather Forecast, 2000.

[25] D. Potts, G. Steidl, and M. Tasche. Fast algorithms for discrete polynomial transforms. *Math. Comput.*, 67:1577 – 1590, 1998.

[26] D. Potts, G. Steidl, and M. Tasche. Fast and stable algorithms for discrete spherical Fourier transforms. *Linear Algebra Appl.*, 275:433 – 450, 1998.

[27] D. Potts, G. Steidl, and M. Tasche. Fast Fourier transforms for nonequispaced data: A tutorial. In J. J. Benedetto and P. J. S. G. Ferreira, editors, *Modern Sampling Theory: Mathematics and Applications*, pages 247 – 270, Boston, 2001. Birkhäuser.

[28] K. Rao and P. Yip. *Discrete Cosine Transforms.* Academic Press, Boston, 1990.

[29] http://www.ece.cmu.edu/~spiral/.

[30] G. Steidl. Fast radix–p discrete cosine transform. *Appl. Algebra Eng. Commun. Comput.*, 3:39 – 46, 1992.

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

[32] R. Suda and M. Takami. A fast spherical harmonics transform algorithm. *Math. Comput.*, 71:703 – 715, 2001.

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

[34] A. F. Ware. Fast approximate Fourier transforms for irregularly spaced data. SIAM *Rev.*, 40:838 – 856, 1998.