7-13-1994

# Efficiency and Stability Issues in the Numerical Computation of Fourier Transforms and Convolutions on the 2-Sphere

D M. Healy Jr
*Dartmouth College*

S S. B. Moore
*Dartmouth College*

D Rockmore
*Dartmouth College*

# EFFICIENCY AND STABILITY ISSUES IN THE NUMERICAL COMPUTATION OF FOURIER TRANSFORMS AND CONVOLUTIONS ON THE 2-SPHERE

D.M. Healy, Jr.
S.S.B. Moore
D. Rockmore

Technical Report PCS-TR94-222

7/94

# Efficiency and Stability Issues in the Numerical Computation of Fourier Transforms and Convolutions on the 2-Sphere *

D.M. Healy Jr., S. S. B. Moore, and D. Rockmore
Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

July 13, 1994

**Abstract**

Earlier work by Driscoll and Healy [20] has produced an efficient algorithm for computing the Fourier transform of band-limited functions on the sphere. In this paper we present a greatly improved inverse transform, and consequent improved convolution algorithm for such functions. We also discuss implementational considerations and give heuristics for allowing reliable floating point implementations of a slightly modified algorithm at little cost in either theoretical or actual performance. This discussion is supplemented with numerical experiments from our implementation in C on a DecStation 5000. These results give strong indications that the algorithm is both reliable and efficient for a large range of useful problem sizes.

## 1 Introduction

The calculation of Fourier expansions and convolutions of functions on the 2-sphere are related problems which have been identified as important computational issues in many areas of applied science [31, 49, 12, 26, 27, 36]. Until the recent work of Driscoll and Healy [20] the solutions proposed to this problem were approximate [3, 43]. In exact arithmetic the Driscoll-Healy algorithm efficiently computes exactly the convolution of functions on $S^2$, assuming that the functions have finite expansions in terms of spherical harmonics. Their paper [20] also provides a priori error estimates and numerical experiments for crucial steps in the algorithm. These results strongly suggest the possibility of an effective floating point implementation of the algorithm.

This paper continues and supplements the work in [20]. The main theoretical result presented here is a more efficient inverse transform for such "band-limited" functions on the sphere, thereby providing a substantial speed-up for the full convolution algorithm. By taking advantage of a matrix formulation of the efficient forward transform algorithm, we are able to derive an improved inverse transform whose complexity is of the same order as the fast forward transform detailed in [20]. Efficient inverse and forward transforms then combine to yield a more efficient convolution algorithm. We also treat in some detail issues related to obtaining a reliable, but still efficient implementation of this fast convolution algorithm. We give indications (both theoretical and experimental) of potential problems

*This work supported in part by ARPA as administered by the AFOSR under contract AFOSR-90-0292. Rockmore also supported in part by an NSF Math Sciences Postdoctoral Fellowship.

with a direct finite precision implementation and then provide heuristics for dealing with these issues. These techniques have been well-tested and at present we believe that we have an efficient and reliable finite precision implementation of a fast spherical convolution algorithm. These claims are supported with numerical evidence.

The organization of the paper is as follows. In Section 2 the necessary Fourier analysis background is given. Section 3 summarizes the relevant results from [20] and presents the fast forward transform as a matrix decomposition. Indications are given regarding the inherent parallelizability of the algorithm. Section 4 describes the fast inverse transform and the consequent fast convolution algorithm. In Section 5 numerical aspects are discussed. Evidence is given for the need to modify the algorithm of Section 4 in order to obtain a reliable implementation. Heuristics are presented towards this end and justified both theoretically and experimentally. Additionally, a modified fast transform algorithm is presented which takes advantage of these heuristics to produce an algorithm which behaves much better numerically while remaining highly efficient. To this end, in Section 6, we present some experiments in using our algorithm for matched filtering of signals on the sphere. A natural application of an efficient spherical convolution algorithm would be towards matched filtering on the sphere as well as towards the efficient computation of the bispectrum for band-limited functions. Discussions of these applications, as well as some efficient variations of the algorithm and a work optimal parallel algorithm, are presented in Section 6. We close with a brief discussion and suggestions for further work in Section 7.

## 2 Fourier analysis on the 2-sphere

As usual, $S^2$ denotes the 2-sphere or unit sphere in $\mathbf{R}^3$. A unit vector in $\mathbf{R}^3$ is described uniquely by an angle $\theta$, $0 \leq \theta \leq \pi$ measured down from the $z$-axis and angle $\phi$, $0 \leq \phi < 2\pi$ measured counterclockwise off the $x$-axis. Thus, if $\omega \in S^2$, then we may write $\omega(\theta, \phi) = (\cos\phi \sin\theta, \sin\phi \sin\theta, \cos\theta)$.

The rotation group of $\mathbf{R}^3$, denoted $SO(3)$ acts transitively on $S^2$. In coordinates, this is the group of those 3 by 3 invertible matrices with real entries for which inverse and transpose coincide. Transitivity means simply that any point on the sphere can be rotated into any other. This gives an identification of points on the 2-sphere with the coset space $SO(3)/SO(2)$, where $SO(2) < SO(3)$ is the subgroup of rotations which fix the north pole, $\eta = (0, 0, 1)$. Under this identification the coset $g \cdot SO(2) \mapsto g\eta$.

Let $L^2(S^2)$ denote the Hilbert space of square integrable functions on the 2-sphere with the usual inner product,

$$\langle f, h \rangle = \int_{S^2} f\overline{h} d\omega.$$

Here $d\omega$ denotes the rotation-invariant ($SO(3)$-invariant) measure on $S^2$. As is well known, the **spherical harmonics** provide an orthonormal basis for $L^2(S^2)$. For any nonnegative integer $l$ and integer $m$ with $|m| \leq l$, the $(l, m)$-spherical harmonic $Y_l^m$ is a harmonic homogeneous polynomial of degree $l$. Thus, the harmonics of degree $l$ span a subspace of $L^2(S^2)$ of dimension $2l+1$. In coordinates, $Y_l^m$ is defined by

$$Y_l^m(\theta, \phi) = (-1)^m \sqrt{\frac{(2l+1)(l-m)!}{4\pi(l+m)!}} P_l^m(\cos\theta) e^{im\phi}, \tag{1}$$

where $P_l^m$ is the $(l, m)$-**associated Legendre function**. The functions $P_l^m$ satisfy a three term recurrence

$$(l-m+1)P_{l+1}^m(x) - (2l+1)xP_l^m + (l+m)P_{l-1}^m(x) = 0. \tag{2}$$

The expansion of any function $f \in L^2(S^2)$ in terms of spherical harmonics is written

$$f = \sum_{l \geq 0} \sum_{|m| \leq l} \hat{f}(l,m) Y_l^m. \tag{3}$$

In (3), $\hat{f}(l,m)$ denotes the $(l,m)$-**Fourier coefficient**, (with respect to the basis of spherical harmonics) which is defined in the usual way as

$$\hat{f}(l,m) = \langle f, Y_l^m \rangle = \int_{S^2} f \overline{Y_l^m} d\omega.$$

In analogy with the case of functions on the circle, we say that $f \in L^2(S^2)$ is **band-limited** with **band-limit** or **bandwidth** $B \geq 0$ if $\hat{f}(l,m) = 0$ for all $l \geq B$.

Defining convolution uses the structure of $S^2$ as a quotient of the group $SO(3)$. Also generalizing the case of the circle, the left convolution of of $h$ by $f$ for two functions $f, h \in L^2(S^2)$, is defined as

$$f \star h(\omega) = \int_{g \in SO(3)} f(g\eta) h(g^{-1}\omega) dg.$$

Here $dg$ denotes the (essentially) unique invariant volume form on $SO(3)$.

# 3 A spherical convolution algorithm

Let $f \in L^2(S^2)$ have bandwidth $B$. Thus, letting $n = B^2$, then $f$ has at most $O(n)$ (potentially) nonzero Fourier coefficients. In [20] Driscoll and Healy show how these Fourier coefficients can be computed exactly (in exact arithmetic) from $4n$ samples (particular values of $f$) in $O(n \log^2 n)$ arithmetic operations. This is then used as the first step in deriving an $O(n^{1.5})$ algorithm for computation of the convolution of two band-limited functions of bandwidth $B$ on $S^2$.

In Section 4 we improve this convolution result, and give an $O(n \log^2 n)$ algorithm. The key idea is a matrix description of Driscoll and Healy's fast forward transform. This allows us to derive a fast inverse transform, which was the original bottleneck to a fast(er) convolution in [20]. We give the matrix formulation in Section 3.3. We begin by collecting the relevant earlier results from [20] in Sections 3.1 and 3.2. We refer the reader to this earlier paper for details.

## 3.1 A convolution theorem for $L^2(S^2)$.

As in the more familiar case of convolution on the circle via the usual (abelian) FFT [9], the spherical convolution algorithm may be decomposed into three basic steps:

- (1) Computation of a forward transform (from the "time" to "frequency" domain);
- (2) Pointwise multiplication of the appropriate transforms;
- (3) Computation of the inverse transform of the result of step (2).

Of course, this approach assumes the existence of some "nice" relationship between the transform and the convolution. This is precisely the first main result of [20].

**Theorem 1** *([20], Theorem 1), Let $f, h \in L^2(S^2)$. Then*

$$\widehat{(f \star h)}(l,m) = 2\pi \sqrt{\frac{4\pi}{2l+1}} \hat{f}(l,m) \hat{h}(l,0).$$

Note that in particular, the convolution of functions of bandwidth $B$, yields a function of bandwidth $B$.

### 3.2 A sampling theorem for band- limited functions on $S^2$ and fast forward transform

The implementation of Step (1) in Section 3.1 requires some sort of "sampling theorem" or quadrature rule to reduce the necessary integrals to sums. Furthermore, it would be most excellent if the sample points allowed the computations to be performed efficiently and stably.

A minor reworking of the sampling theorem of [20] shows that if $f \in L^2(S^2)$ has bandwidth $B$, then the grid given by sample points $\{(\theta_j, \phi_k) | 0 \leq j, k < 2B\}$, for $\theta_j = \frac{\pi(j+1/2)}{2B}$ and $\phi_k = \frac{\pi(k+1/2)}{B}$ gives a quadrature rule for computing the Fourier coefficients of $f$.

**Theorem 2** *([20], Theorem 3) Let $f(\theta, \phi)$ be a band-limited function on $S^2$ such that $\hat{f}(l, m) = 0$ for all $l \geq B$. Then*

$$\hat{f}(l,m) = \frac{\sqrt{2\pi}}{2B} \sum_{j=0}^{2B-1} \sum_{i=0}^{2B-1} a_j^{(B)} f(\theta_j, \phi_k) \overline{Y_l^m}(\theta_j \ \phi_k),$$

*for $l < B$ and $|m| \leq l$. The weights $a_j^{(B)}$ are defined to be the unique solution vector for the system of equations*

$$a_0^{(B)} P_l(\cos\theta_0) + a_1^{(B)} P_l(\cos\theta_1) + \cdots + a_{2B-1}^{(B)} P_l(\cos\theta_{2B-1}) = \sqrt{2}\delta_0(l).$$

*Here, $\theta_j = \frac{\pi(j+1/2)}{2B}$, $\phi_k = \frac{\pi(k+1/2)}{B}$, $P_l = P_l^0$ (a multiple of the Legendre polynomial of degree $l$) and $\delta_0$ is the characteristic function of 0.*

In [20] a closed form solution for the sample weights $\{a_j^{(B)}\}_j$ is derived. In the present notation, this amounts to

$$a_j^{(B)} = \frac{\sqrt{2}}{B} \sin\left(\frac{\pi(j+1/2)}{2B}\right) \sum_{l=0}^{B-1} \frac{1}{2l+1} \sin\left([2l+1]\frac{\pi(j+1/2)}{2B}\right), \qquad j = 0, ..., 2B-1.$$

Figure 1 shows a plot for a range of problem sizes. Note the similarity to the $sin(\theta)$ factor used for integration on the sphere.

From the point of view of linear algebra, the number of sample points is asymptotically optimal in $B$. That is, if $f$ is determined by its Fourier coefficients $\hat{f}(l, m)$ for $0 \leq l < B, |m| \leq l$, then by simple linear algebra, at least as many samples $(B^2 - 2B + 2)$ are need. Instead, the grid $\{(\theta_i, \phi_j)\}$ uses $4B^2$ points, so asymptotically, $O(B^2)$ points as well.

This grid also provides algorithmic advantages, as we see in the following description of an efficient algorithm to compute the Fourier transform on $S^2$.

**Theorem 3** *If $f(\theta, \phi)$ is in the span of $\{Y_l^m | l < B, |m| \leq l\}$, then the Fourier coefficients $\hat{f}(l, m)$ for $l < B, |m| \leq l$ can be computed in $O(n \log^2 n)$ operations from the $n = 4B^2$, sampled values $f(\frac{\pi(k+1/2)}{2B}, \frac{\pi(j+/2)}{B})$, $0 \leq j, k < 2B - 1$, using a preprocessed data structure of size $O(B \log^2 B)$.*

**Proof Sketch:** From the sampling theorem we know that the transform may be computed by means of finite sums,

$$\hat{f}(l,m) = \sum_{j=0}^{2B-1} \sum_{k=0}^{2B-1} a_k^{(B)} f(\theta_k, \phi_j) Y_l^m(\theta_k, \phi_j)$$

where $\theta_k = \frac{(k+1/2)\pi}{2B}$, $\phi_j = \frac{(j+1/2)\pi}{B}$, and the $a_k^{(B)}$ are as defined in the sampling theorem, Theorem 2. Using (1), we rewrite the above to obtain

$$\hat{f}(l,m) = q_l^m \sum_{k=0}^{2B-1} a_k^{(B)} P_l^m(\cos\theta_k) \sum_{j=0}^{2B-1} e^{-im\phi_j} f(\theta_k, \phi_j),$$

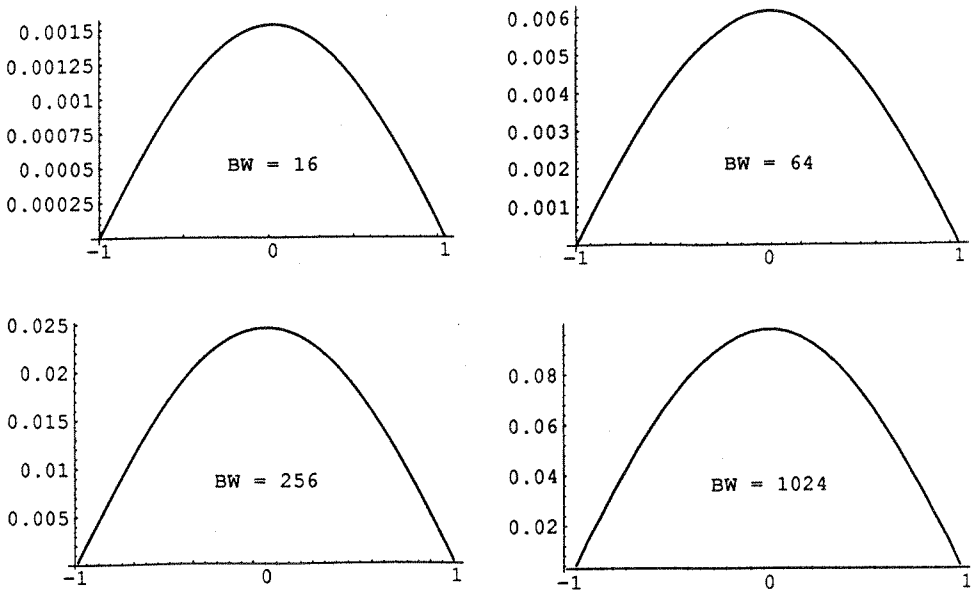



Figure 1: Plot of sample weights for range of problem sizes.

where the $q_l^m$ are the normalization coefficients for the spherical harmonics. The inner sum is computed for each fixed $k$ and for all $m$ in the appropriate range by means of the Fast Fourier Transform [12, 7, 48]; the resulting table $\{\check{f}(\theta_k, m)\}$ is obtained in $O(B^2 \log B)$ operations.

Thus, we are left with computing the Fourier coefficients as the sums,

$$\hat{f}(l,m) = q_l^m \sum_{k=0}^{2B-1} a_k^{(B)} \check{f}(\theta_k, m) P_l^m(\cos\theta_k).$$

For each fixed $m$, the collection of expressions

$$\{\hat{f}(l,m) \;\Big|\; |m| \le l < B\} \tag{4}$$

is obtained by a (discrete) **associated Legendre transform** of the partial results $\{\check{f}(\theta_k, m) \mid 0 \le k < 2B\}$ is . That is to say, it is a set consisting in finite sums of the form $\sum_k h(k) P_l^m(x_k)$ for a sequence of values $h(k)$ and sampled points $x_k$. In [17] it is shown how to use the three term recurrence (2) to evaluate the collection of these transforms (4) for all $|m| < B$, in $O(B \log^2 B)$ time, as opposed to the $O(B^2)$ operations required by direct computation. Given this result and the fact that $m$ ranges over $O(B)$ values, the total time to compute all values of $\hat{f}(l,m)$ is $O(B^2 \log^2 B)$.

■

The efficient calculation of the various associated Legendre transforms is the key to the preceding result. We now turn to a description of this algorithm as a matrix factorization.

### 3.3 Matrix formulation of the fast forward Legendre transforms

Let $\tilde{P}_l^m = q_l^m P_l^m$ denote the normalized associated Legendre function. Then for any complex vector $\mathbf{c} = (c_0, \ldots, c_{2B-1})$, we define the (discrete) normalized associated Legendre transforms (NALT) to be the sequence of summations

$$\left(\hat{\mathbf{c}}(\tilde{P}_0^m), \ldots, \hat{\mathbf{c}}(\tilde{P}_{B-1}^m)\right), \tag{5}$$

where

$$\hat{\mathbf{c}}(\tilde{P}_l^m) = \sum_{k=0}^{2B-1} \tilde{P}_l^m(\cos\theta_k)\mathbf{c}_k$$

and $\theta_k = \frac{\pi(k+1/2)}{2B}$. Note that for $|m| > l$, $\tilde{P}_l^m = 0$.

In [20] the three-term recurrence (2) satisfied by the (un-normalized) associated Legendre functions is used to produce an $O(B\log^2 B)$ algorithm for computing (5). When this is applied to a vector of weighted samples of a band-limited function as described in the proof sketch of Theorem 3, this permits us to calculate the spherical harmonic decomposition of that function efficiently.

In brief, the algorithm proceeds in basically two steps, first projecting the vector $\mathbf{c}$ onto a basis given by sampled cosine functions and then using the recurrence to set up a divide-and-conquer approach to successively produce the individual transforms. This second step may be neatly encoded as a sequence of successive applications of block matrices with circulant blocks, followed by projections to an appropriate data vector. In the case of $m = 0$, the problem is essentially that of computing discrete Legendre polynomial transforms. In this case [20] computes closed form expressions for the relevant matrices in terms of shifted Legendre polynomials [5] [1]. It is also pointed out that for $|m| > 0$, the analogous idea still works, as it depends only on the three-term recurrence.

In this section we work out the explicit details for $|m| > 0$ and write the entire forward transform algorithm as a sequence of matrix-vector multiplications. To pose the problem (and solution) in the framework of linear algebra, we see that the NALT (5) can be written as the matrix-vector product

$$\tilde{\mathcal{P}}_B^m \cdot \mathbf{c} \tag{6}$$

where

$$\tilde{\mathcal{P}}_B^m = \begin{pmatrix} \tilde{P}_m^m(\cos\theta_0) & \tilde{P}_m^m(\cos\theta_1) & \cdots & \tilde{P}_m^m(\cos\theta_{2B-1}) \\ \tilde{P}_{m+1}^m(\cos\theta_0) & \tilde{P}_{m+1}^m(\cos\theta_1) & \cdots & \tilde{P}_{m+1}^m(\cos\theta_{2B-1}) \\ \vdots & \vdots & \cdots & \vdots \\ \tilde{P}_{B-1}^m(\cos\theta_0) & \tilde{P}_{B-1}^m(\cos\theta_1) & \cdots & \tilde{P}_{B-1}^m(\cos\theta_{2B-1}) \end{pmatrix}. \tag{7}$$

Direct computation of the product (6) requires $O(B^2)$ operations. Working from the algorithm in [20] we show that the fast transform algorithm can be written as a factorization of the matrix $\tilde{\mathcal{P}}_B^m$,

$$\tilde{\mathcal{P}}_B^m = \mathcal{O}\mathcal{M}_{b-1}\cdots\mathcal{M}_1\mathcal{M}_0\mathcal{I}, \tag{8}$$

as a chain of $b = \log_2(B) - 1$ sparse matrices, $\mathcal{M}_j$, along with a diagonal matrix $\mathcal{O}$ for normalizing the output and another low complexity operator, $\mathcal{I}$ for arranging the input data. Their successive application requires a total of only $O(B\log^2 B)$ operations. We give here a sufficient description of these operators to demonstrate this complexity bound.

It is the structure of the $\mathcal{M}_j$ which determines that the complexity of the forward transform is $O(B\log^2 B)$. Each $\mathcal{M}_j$ is a $4B \times 4B$ block diagonal matrix of the following form:

[1] In [5] Barracund and Dickinson derive a family of polynomials related to the Legendre poloynomials by adding a parameter to the three-term recurrence defining the Legendre polynomials. This new parameter has the effect of *shifting* the starting point of the recurrence. This generalization makes sense for any family of functions defined by a recurrence and in section 3.4 we use the same techniques applied to the associated Legendre functions. Unfortunately in [5] the newly derived polynomials are called *associated Legendre polynomials* - a name which does not generalize nicely for the associated Legendre functions. Thus, we adopt the terminology *shifted* Legendre polynomials and more generally *shifted* Legendre functions (cf. Section 3.4).

$$\begin{pmatrix} \begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(0) \end{pmatrix} & & & & \\ & \begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(1) \end{pmatrix} & & & \\ & & \begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(2) \end{pmatrix} & & \\ & & & \ddots & \\ & & & & \begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(2^j - 1) \end{pmatrix} \end{pmatrix} \quad (9)$$

As written, each block

$$\begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(k) \end{pmatrix} \quad (10)$$

is $\frac{2B}{2^{j-1}} \times \frac{2B}{2^{j-1}}$ and is itself a block matrix. The upper block $\Pi_j$ is a $\frac{2B}{2^j} \times \frac{2B}{2^{j-1}}$ projection matrix defined as

$$\Pi_j = \begin{pmatrix} \pi_j & \bigcirc \\ \bigcirc & \pi_j \end{pmatrix}$$

where

$$\pi_j = \underset{\substack{\leftarrow \frac{B/2^j}{2} \rightarrow \leftarrow B/2^j \rightarrow \leftarrow \frac{B/2^j}{2} \rightarrow}}{\left( \begin{array}{c|c|c} \bigcirc & Id & \bigcirc \end{array} \right)} \begin{matrix} \uparrow \\ \frac{B}{2^j} \\ \downarrow \end{matrix}.$$

The lower block of (10) is given by the product of $\Pi_j$ with a $\frac{2B}{2^{j-1}} \times \frac{2B}{2^{j-1}}$ matrix $\mathcal{T}_j(k)$. The matrix $\mathcal{T}_j(k)$ is itself a block matrix, composed of four square circulant blocks. Thus we can write this as

$$\mathcal{T}_j(k) = \begin{pmatrix} \mathcal{F}^{-1} & \bigcirc \\ \bigcirc & \mathcal{F}^{-1} \end{pmatrix} \begin{pmatrix} a_j(k) & b_j(k) \\ c_j(k) & d_j(k) \end{pmatrix} \begin{pmatrix} \mathcal{F} & \bigcirc \\ \bigcirc & \mathcal{F} \end{pmatrix} \quad (11)$$

where $a_j(k), b_j(k), c_j(k)$, and $d_j(k)$ are diagonal matrices of size $\frac{2B}{2^j}$ and $\mathcal{F}$ is the $\frac{2B}{2^j}$-dimensional discrete Fourier transform (DFT) operator.

If the application of the DFT is performed using a standard FFT decomposition algorithm for the $\mathcal{F}$ matrices [9], then each of the $2^j$ blocks of the form (10) can applied to an arbitrary input vector in $O(\frac{B}{2^j} \log \frac{B}{2^j})$ operations, thus permitting multiplication by $\mathcal{M}_j$ to be performed in at most $O(B \log \frac{B}{2^j})$

operations. Consequently, the sequence of applications of each of the $\mathcal{M}_j$ requires at most $O(B \log^2 B)$ operations.

It still remains to give closed form expressions for the diagonal matrices $a_j(k), b_j(k)$, $c_j(k), d_j(k)$ defined in (11) which help to make up the matrices $\mathcal{T}_j(k)$. In analogy with the case of the fast discrete Legendre polynomial transform explicated in [20] (esp. Lemma 4), the diagonal matrices $a_j(k)$, $b_j(k)$, $c_j(k)$ and $d_j(k)$ are obtained from the samples of certain trigonometric polynomials, in this instance derived from the associated Legendre functions. For reasons that will become clear in the next section, we call these related trigonometric polynomials **shifted associated Legendre functions**. In our implementations to date, their required sampled values are prestored. The following section shows (1) how these values may be derived efficiently from a recurrence relation and (2) some naive estimates of their growth, useful in identifying potential problem areas in an implementation.

The remaining steps of the algorithm, the input and output matrices, require a total of $O(B \log B)$ computations, so that the complexity of the full algorithm is as advertised. To be more specific, the transform begins with the input operator, which factors as

$$\mathcal{I} = \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P_m^m & 0 \\ 0 & P_{m+1}^m \end{pmatrix} \begin{pmatrix} I \\ I \end{pmatrix} = \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P_m^m \\ P_{m+1}^m \end{pmatrix}$$

where $P_m^m = \mathcal{S}^m(\sigma) = Diag\{\sin^m(\theta_j) | j = 0, ..., 2B-1\}$, $P_{m+1}^m = Diag\{P_{m+1}^m(\theta_j) | j = 0, \ldots, 2B-1\}$ and $\mathcal{C}$ is a cosine transform matrix,

$$\mathcal{C} = \begin{pmatrix} \cos((B-1)\theta_0) & \cos((B-1)\theta_1) & \ldots & \cos((B-1)\theta_{2B-1}) \\ \cos((B-2)\theta_0) & \cos((B-2)\theta_1) & \ldots & \cos((B-2)\theta_{2B-1}) \\ \vdots & \vdots & \ldots & \vdots \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ldots & \vdots \\ \cos(B\theta_0) & \cos(B\theta_1) & \ldots & \cos(\theta_{2B-1}) \end{pmatrix}$$

and $I$ is the $2B \times 2B$ identity matrix. The initial input to $\mathcal{I}$ is the vector of appropriate sample values of the function $f$. Note that the effect of applying $\binom{I}{I}$ is simply to repeat the input.

By using a fast cosine transform algorithm to implement $\mathcal{C}$, (which is again given as a matrix factorization - cf. [50]) this entire step can be done in $O(B \log B)$ operations.

The algorithm concludes with an output operator $\mathcal{O}$. This is a $B \times 4B$ matrix which selects out and normalizes the Legendre coefficients while throwing away extraneous information computed by the preceding steps. Each row has one nonzero normalizing element, so this matrix is applied in $O(B)$ steps. We collect this discussion in the following theorem.

**Theorem 4** *Let the matrices $\mathcal{O}, \mathcal{C}, P_m^m, P_{m+1}^m, M_j$ be defined as in the preceding discussion. Then the discrete normalized associated Legendre transform $\tilde{\mathcal{P}}_B^m \cdot f$ for an arbitrary data vector $f$ of length $2B$ is computed by the succession of products*

$$\mathcal{O}\mathcal{M}_{b-1} \cdots \mathcal{M}_1 \mathcal{M}_0 \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P_m^m & 0 \\ 0 & P_{m+1}^m \end{pmatrix} \cdot f$$

*in $O(B \log^2 B)$ operations.*

The sequence of convolutions and projections coded into the matrices $M_j$ defined in (9) precisely reflect the divide and conquer nature of the fast forward transform algorithm. It is useful, as well as necessary for the discussion in Section Five, to record the results of each of the successive applications of the matrices $M_j$.

Define the weighted inner product

$$\langle g, h\rangle = \sum_{j=0}^{2B-1} a_j f(\cos\theta_j) g(\cos\theta_j) \tag{12}$$

with the $a_j$ and $\theta_j$ defined as in the sampling theorem. For any integer $k$, define $C_k(t) = \cos(kt)$. Then for each $|m| < B$ and $l \geq |m|$ and suitably defined function $f$, we will define a sequence of vectors $Z_l^m$ by

$$Z_l^m(k) = \langle f, C_k P_l^m\rangle \qquad |k| \leq 2B. \tag{13}$$

Note that $Z_l^m(0) = \hat{f}(l,m)$. The effect of the algorithm is to successively compute truncations of the full sequences $Z_l^m(k)$. Let $0 \leq d \leq 2B$. For any of the (a priori) sequences of length $4B$, $Z_l^m$, let $Z_l^{m,d}$ denote the truncation of $Z_l^m$ giving the subsequence

$$Z_l^{m,d} = \Big(Z_l^m(-d), \ldots, Z_l^m(0), \ldots, Z_l^m(d)\Big). \tag{14}$$

Then a little bit of work shows (cf. [20], esp. Section 4) that

$$\mathcal{M}_j \cdots \mathcal{M}_1 \cdot \mathcal{M}_0 \cdot \mathcal{I} \cdot f = \begin{pmatrix} Z_m^{m,d(j)} \\ Z_{m+1}^{m,d(j)} \\ \vdots \\ Z_{m+\frac{rB}{2^{j+1}}}^{m,d(j)} \\ Z_{m+\frac{rB}{2^{j+1}}+1}^{m,d(j)} \\ \vdots \end{pmatrix} \tag{15}$$

where $r$ is meant to run from 0 to $2^{j+1}-1$ and $d(j) = \frac{B}{2^{j+1}}$.

Thus, after the application of $\mathcal{M}_j$, we are in possession of the Fourier coefficients

$$\left\{\hat{f}(m,m), \hat{f}(m+1,m), \ldots, \hat{f}(m+\frac{rB}{2^{j+1}}, m), \hat{f}(m+1+\frac{rB}{2^{j+1}}, m), \ldots, \hat{f}(m+B-\frac{B}{2^{j+1}}, m), \hat{f}(m+1+B-\frac{B}{2^{j+1}}, m)\right\}$$

### 3.4 Shifted associated Legendre functions

The matrices $\mathcal{T}_j(k)$ defined in (11) are computed as products of matrices which arise naturally from the three term recurrence satisfied by the $P_l^m$. Recall that each NALT (5) is computed for a fixed $m > 0$. We think of $m$ as fixed throughout this section.

Since $P_l^m \equiv 0$ for $l < m$, it will prove convenient in this section to define

$$\mathfrak{p}_k^m = P_{m+k}^m. \tag{16}$$

In this new notation the recurrence (2) becomes

$$\mathfrak{p}_{k+1}^m(x) = \frac{2m+2k+1}{k+1} x \mathfrak{p}_k^m(x) - \frac{2m+k}{k+1} \mathfrak{p}_{k-1}^m(x) \tag{17}$$

The matrices $\mathcal{T}_j(k)$ are a composition of $m$ one step convolutions. To determine the closed form expression for $\mathcal{T}_j(k)$ we follow [20] and define a block matrix

$$A(r) = \begin{pmatrix} 0 & I \\ -\frac{2m+r}{r+1} I & \frac{2m+2r+1}{r+1} \cos\sigma \end{pmatrix} \tag{18}$$

where $\sigma$ is the diagonal matrix of sample values $\{\frac{\pi(k+\frac{1}{2})}{2B/2^j} \mid k = \frac{-2B}{2^j}, \ldots, \frac{2B}{2^j} - 1\}$. Then for any indices $r, k \geq 0$, define

$$T(r,k) = \prod_{s=0}^{k} A(r+s) = A(r+k) \cdot A(r+k-1) \cdots A(r) \tag{19}$$

where we note that the factors are to be ordered with indices increasing from right to the left.

In the notation of (19),

$$\mathcal{T}_j(k) = \begin{pmatrix} \mathcal{F}^{-1} & 0 \\ 0 & \mathcal{F}^{-1} \end{pmatrix} T\left((k+1)\frac{2B}{2^{j+1}} + 1,\ \frac{B}{2^j}\right) \begin{pmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{pmatrix}. \tag{20}$$

Note that (20) shows that $\mathcal{T}_j(k)$ is (up to conjugation by DFT matrices) a $2 \times 2$ block matrix with diagonal blocks given by trigonometric polynomials evaluated on the diagonal matrix $\sigma$. In fact, these polynomials are determined by a shifted form of the associated Legendre function recurrence relation (2). Specifically, the definition (19) shows that there are sequences $a_k$, $b_k$ of trigonometric polynomials with

$$\prod_{s=0}^{k} A(r+s) = \begin{pmatrix} a_k(\cos\sigma) & b_k(\cos\sigma) \\ a_{k+1}(\cos\sigma) & b_{k+1}(\cos\sigma) \end{pmatrix}$$

satisfying the recurrence implicit in the identity:

$$\prod_{s=0}^{k} A(r+s) = A(r+k) \prod_{s=0}^{k-1} A(r+s),$$

namely

$$\begin{pmatrix} a_k(\cos\sigma) & b_k(\cos\sigma) \\ a_{k+1}(\cos\sigma) & b_{k+1}(\cos\sigma) \end{pmatrix} = \begin{pmatrix} \bigcirc & Id \\ -\frac{2m+r+k}{r+k+1} I & \frac{2m+2r+2k+1}{r+k+1} \cos\sigma \end{pmatrix} \begin{pmatrix} a_{k-1}(\cos\sigma) & b_{k-1}(\cos\sigma) \\ a_k(\cos\sigma) & b_k(\cos\sigma) \end{pmatrix},$$

which says both sequences are solutions of the linear recurrence

$$y^m_{k+1}(\cos\sigma) = \frac{2m+2r+2k+1}{r+k+1} \cos\sigma\ y^m_k(\cos\sigma) - \frac{2m+r+k}{r+k+1}\ y^m_{k-1}(\cos\sigma) \tag{21}$$

which is exactly the $r$-shifted version of (17). Initial conditions are given by

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = -\frac{2m+r}{r+1} \begin{pmatrix} \underline{0} \\ \underline{1} \end{pmatrix} \qquad \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} \underline{1} \\ \frac{2m+2r+1}{r+1} \cos\sigma \end{pmatrix} \tag{22}$$

where $\underline{0}$ and $\underline{1}$ denote the vectors of length $\frac{2B}{2^j}$ with all entries equal to 0 and 1 respectively.

In [5], Barracund and Dickinson show that for $m = 0$, the recurrence (21) and initial conditions (22) define the sequences uniquely as the **shifted Legendre polynomials** [5] and they investigate these functions in some detail. In particular the asymptotic behavior of these polynomials is studied and their generating function is determined.

A more detailed study of these **shifted associated Legendre functions** for arbitrary $m > 0$ is a straightforward generalization of [5]. Following [5] and using classical results of Robin [47] for associated Legendre functions of the first and second type, we may define $P^m_k(r,x)$ to be a solution of the $r$-shifted recurrence for the associated Legendre functions (21), subject to the initial conditions

$$y^m_{-1}(r,x) = 0 \quad \text{and} \quad y^m_0(r,x) = 1. \tag{23}$$

Consequently, we may write

$$\prod_{s=0}^{k} A(r+s) = \begin{pmatrix} -\frac{2m+r}{r+1} P_{k-1}^m(r+1, \cos\sigma) & P_k^m(r, \cos\sigma) \\ -\frac{2m+r}{r+1} P_k^m(r+1, \cos\sigma) & P_{k+1}^m(r, \cos\sigma) \end{pmatrix}. \tag{24}$$

In order to derive a priori error estimates for the successive application of the matrices $\mathcal{T}_j(k)$ we need to estimate the $P_k^m(r,x)$. When $m=0$ many facts are known about the trigonometric polynomials, $P_k^m(r, \cos\theta)$ [5] including the one we need for our bound; they take their maximum absolute value at $\theta = 0$;

$$\sup_{\theta\in[0,2\pi]} |P_k^0(l, \cos\theta)| = P_k^0(l,1) = \sum_{j=0}^{k} \frac{l}{l+j}. \tag{25}$$

To derive bounds for $m > 0$, we proceed as in [5] and study the generating function for the $P_k^m(r,x)$,

$$g^m(x,t) = \sum_{k=0}^{\infty} P_k^m(r,x) t^k.$$

Using the recurrence (21) as well as the initial conditions (23) it is easy to check that $g^m(x,t)$ satisfies the differential equation

$$\Big[(\Theta + r) - xt(2m + 2\Theta + 2r + 1) + t^2(2m + \Theta + r + 1)\Big] g^m(x,t) = r \tag{26}$$

where

$$\Theta = t\frac{d}{dt}.$$

Solving the differential equation (26) gives

$$g^m(t,x) = \frac{rt^{-r}}{\sqrt{1-2xt+t^2}^{2m+1}} \int_0^t \frac{s^{r-1}(1-2xs+s^2)^m}{\sqrt{1-2xs+s^2}} ds. \tag{27}$$

Using the identity

$$\frac{1}{\sqrt{1-2xt+t^2}} = \sum_{l=0}^{\infty} P_l(x) t^l \tag{28}$$

we may rewrite (27) as

$$\begin{aligned} \textstyle\sum_{k=0}^{\infty} P_k^m(r,x) t^k &= g^m(t,x) \\ &= \tfrac{r}{t^r} \Big(\textstyle\sum_{l=0}^{\infty} P_l(x) t^l\Big)^{2m+1} \int_0^t s^{r-1}(1-2xs+s^2)^m \Big(\textstyle\sum_{l=0}^{\infty} P_l(x) s^l\Big) ds. \end{aligned} \tag{29}$$

By equating coefficients, a naive bound for $P_k^m(r,x)$ can now be obtained. A straightforward expansion of the integrand on the righthand side of (29) and term by by term integration yields

$$\frac{r}{t^r} \int_0^t s^{r-1}(1-2xs+s^2)^m \Big(\sum_{l=0}^{\infty} P_l(x) s^l\Big) ds = r \sum_{l=0}^{\infty} P_l(x) \sum_{j=0}^{m} \binom{m}{j} (1-x^2)^{m-j} \sum_{i=0}^{2j} \binom{2j}{i} (-x)^{2j-i} \frac{t^{l+i}}{r+l+i}. \tag{30}$$

An upper bound on the coefficient of $t^k$ in (29) may be obtained by bounding the absolute values of all the terms in the expansion (29). Since $|x| \le 1$ and $|P_l(x)| \le 1$ for all $l$ we may bound (30) by

$$r \sum_{l=0}^{\infty} \sum_{j=0}^{m} \binom{m}{j} \sum_{i=0}^{2j} \binom{2j}{i} \frac{t^{l+i}}{r+l+i}. \tag{31}$$

Keeping in mind that $\binom{2m}{k}$ is maximized at $k = m$, we see that the coefficient of $t^a$ in (31) is bounded by

$$\frac{ra}{r+a}\binom{2m}{m}2^m.$$

Similar considerations show that the coefficient of $t^b$ in $\left(\sum_{l=0}^{\infty} P_l(x)t^l\right)^{2m+1}$ is bounded by the coefficient of $t^b$ in $(1-t)^{-(2m+1)}$. This is easily seen to be $\binom{2m+b}{b}$. Thus, for all $|x| \leq 1$,

$$|P_k^m(r,x)| \leq \binom{2m}{m}2^m \sum_{a=0}^{k}\binom{2m+a}{a}\frac{ra}{r+a}. \tag{32}$$

The bound (32) is provided as some indication of the rate of growth of the $P_k^m(r,x)$. Numerical experiments seem to indicate that while these bounds are exaggerated, actual rates of growth do appear to be at least exponential in $m$.

### 3.5 Further remarks

1. **Parallelizability.** The divide and conquer nature of this algorithm, as evidenced by the factorization of $\tilde{\mathcal{P}}_B^m$ indicates that the algorithm of Section 3.3 should be easily adapted for parallel implementation.

In particular, we can show that the fast forward spherical transform can be described as a **work optimal** algorithm on a **hypercube** network. The algorithm which we present is in the class of so-called **leveled** hypercube algorithms and as such lends itself to efficient implementation on hypercube-derived networks with no routing delays. We defer this discussion to Section 6.

2. **Other fast polynomial transforms.** The complexity result described in the preceding sections depends only on the three term recurrence satisfied by the associated Legendre functions. In the context of a statistical application, in [21] a general algorithm is detailed for the fast computation of discrete polynomial transforms for any set of polynomials satisfying a three-term recurrence. The example of the Hahn and Chebyshev polynomials is worked out in some detail.

3. **Compact groups.** More generally, the entire discussion of Sections Two and Three can be phrased in the language of representation theory. The action of $SO(3)$ on $S^2$ is a particular instance of an action of a compact group on a homogeneous space. The decomposition of $L^2(S^2)$ into spherical harmonics of a given degree is precisely the decomposition of the regular representation of $SO(3)$ on $S^2$ into irreducible subrepresentations, naturally indexed by degree. For a compact group acting on a homogeneous space, the associated regular representation decomposes into irreducibles, which are naturally indexed by a subset of the dominant integral weights in the corresponding Lie algebra. In this setting a function is said to be band-limited if it can be expressed as a linear combination of functions which appear in irreducible representations whose associated highest weight vector has bounded norm.

Recently, D. Maslen [37] has extended some of the results of [20] to compact groups and their quotients. A sampling theorem is proved for band-limited functions on any compact group an sampling distributions are determined for the particular examples of the quotients $SO(n)/SO(n-1)$ and $SU(n)/SU(n-1)$. The sampling grids permit the application of the techniques described in [21] to obtain fast Fourier transforms and thus fast convolutions for associated band-limited functions.

4. **Finite groups.** Fast Fourier transforms have also been constructed for arbitrary finite groups [18, 13]. The techniques developed naturally generalize the famous Cooley-Tukey FFT [14], which in

the group theoretic formulation give efficient and reliable computation of the discrete Fourier transform for abelian groups. The abelian FFT is well-known to have a wide spectrum of applications (cf. [52] and references therein). The relatively new nonabelian FFT's have also begun to display a range of possible applications in areas such as data analysis, adaptive filtering and data compression [17, 29, 30, 32].

# 4 Fast inversion and convolution

In this section we present results on the complexity of computing the inverse Fourier transform for the sphere and thus, by applying Theorems 1 and 3, results for computing convolution as well. These results improve on those of [20].

For our purposes, the inverse transform is the map which takes as input, a set of complex numbers $c_{l,m}$, interpreted as Fourier coefficients in a spherical harmonic expansion. Thus, these coefficients determine a function $f$ on the sphere via the synthesis formula

$$f = \sum_{l \in \mathbb{N}} \sum_{|m| \leq l} c_{l,m} Y_l^m.$$

For the present discussion, we will consider coefficients which vanish for $l$ above some band-limit $B$, so there is no trouble with the convergence of this formula.

The output of the discrete inverse transform is the set of samples of the synthesised function $f$ at the previously defined equiangular grid. We collect these assumptions in the following definition.

**Definition 1** *The discrete inverse spherical Fourier transform with band-limit $B = 2^b$ maps a collection of complex coefficients $c_{l,m}$, $l < B$ and $|m| \leq l$, to the collection of samples of the inverse spherical Fourier transform*

$$f(\theta_j, \phi_k) = \sum_{l \in \mathbb{N}} \sum_{|m| \leq l} c_{l,m} Y_l^m(\theta_j, \phi_k),$$

*taken at the equiangular grid of the sampling theorem, $\theta_j = \frac{(j+1/2)\pi}{2B}$, $j = 0, ..., 2B-1$; and $\phi_k = \frac{(k+1/2)\pi}{B}$, $k = 0, ..., 2B-1$.*

If we let $n = B^2$, then this map transforms a collection of $O(n)$ Fourier coefficients to $O(n)$ samples of its associated spherical harmonic expansion. Computed directly, this requires $O(n^2)$ calculations. The main result of this section - and new theoretical result of this paper - is that in fact, this may be accomplished in $O(n \log^2 n)$ operations.

**Theorem 5** *Let $B = 2^b$ be a fixed band-limit. The discrete inverse spherical Fourier transform for this band-limit may be computed in $O(n \log^2 n)$ calculations, where $n = B^2$.*

**Proof:** Let $c_{l,m}$, $l < B$ and $|m| \leq l$, be given. We wish to compute the collection of samples of the inverse spherical Fourier transform

$$f(\theta_j, \phi_k) = \sum_{l=0}^{B-1} \sum_{|m| \leq l} c_{l,m} Y_l^m(\theta_j, \phi_k).$$

using (1) we rewrite this in the form

$$\sum_{m=1-B}^{B-1} e^{-im\phi_j} \sum_{l=|m|}^{B-1} c_{l,m}\, q_l^m P_l^m(\cos\theta_k), \tag{33}$$

where $q_l^m$ denotes the appropriate normalization constant. Take $h(\theta_k, m)$ to be the inner sum

$$\sum_{l=|m|}^{B-1} c_{l,m}\, q_l^m P_l^m(\cos\theta_k)$$

Rather than obtaining each of the $O(B^2)$ values for $h$ naively by summing the $O(B)$ terms in a total of $O(B^3)$ time, we will show that there is a more efficient approach. Observe that the vector $\mathbf{h}(m) = \{h(\theta_k, m)\}_{k=0,\ldots,2B-1}$ is obtained as a matrix vector product

$$\mathbf{h}(m) = (\tilde{\mathcal{P}}_B^m)^t \cdot \mathbf{c}_m,$$

with $\mathbf{c}_m = \{c_{l,m}\}_{l=m,\ldots,b-1}$, and $(\tilde{\mathcal{P}}_B^m)^t$ is the Vandermonde-like matrix for the order $m$ normalized associated Legendre functions,

$$(\tilde{\mathcal{P}}_B^m)^t = \begin{pmatrix} \tilde{P}_m^m(\cos\theta_0) & \cdots & \tilde{P}_{B-1}^m(\cos\theta_0) \\ \tilde{P}_m^m(\cos\theta_1) & \cdots & \tilde{P}_{B-1}^m(\cos\theta_1) \\ \vdots & \vdots & \vdots \\ \tilde{P}_m^m(\cos\theta_{2B-1}) & \cdots & \tilde{P}_{B-1}^m(\cos\theta_{2B-1}) \end{pmatrix}.$$

As indicated by the notation, this matrix is the transpose of the matrix described in equation (7)which is applied to the vector of weighted function samples to implement the forward transform (NALT).

In Section 3.3, an algorithm was described effecting the efficient application of the matrix $\tilde{\mathcal{P}}_B^m$ to a vector; when computed this way, the NALT applies the matrix in $O(B\log^2 B)$ operations rather than the apparent $O(B^2)$. A decomposition for the efficient application of $\tilde{\mathcal{P}}_B^{mt}$ follows from that algorithm, as we shall see in more detail momentarily. In fact the complexity is the same.

Accepting this as fact for the moment, the cost of obtaining all of the $2B$ vectors $\mathbf{g}(m)$ is $O(B^2\log^2 B)$. The final result is obtained by applying a regular FFT to each of these vectors at a cost of $O(B\log B)$ per vector, for a total additional cost of $O(B^2\log B)$. Denoting the total number of samples by $n = 4B^2$, we may write the total cost as $O(n\log^2 n)$, as desired.

So the proof all comes down to an efficient algorithm for applying the matrix $(\tilde{\mathcal{P}}_B^m)^t$ Recall that the fast NALT algorithm in section 3.3 gives a factorization into sparse matrices

$$\tilde{\mathcal{P}}_B^m = \mathcal{O}\mathcal{M}_{b-1}\cdots\mathcal{M}_1\mathcal{M}_0\mathcal{I}.$$

The structure of the factors permitted their successive application to a vector to be accomplished with a total cost of $O(B\log^2 B)$ operations rather than the apparent $O(B^2)$.

For a truly fine-grained picture of this complexity, most of the factors in this decomposition are themselves further decomposed. For instance, the circulant blocks of the matrix $\mathcal{M}_j$ are all diagonalized by conjugation with a matrix with $DFT$ blocks. These block DFT matrices are further decomposed into sparse matrix factors by means of the usual FFT factorization, see [53]. In the end, $\mathcal{M}_j$ is itself replaced by a string of very sparse matrix factors.

From this point of view, the complexity is understood in terms of the sparseness of the matrix factors in the decomposition. If we consider an operation to be a multiply followed by an addition, then the operation cost of applying one of these matrix factors to a vector is proportional to the number of entries in that matrix which are different from 0 or 1. The point of the fine-grained factorization is that the factors are all very sparse, so that each may be applied in relatively few operations. So few in fact, that the sum of operations, which is proportional to the total number of nonzero entries in all of the matrices, is $O(B\log^2 B)$.

Suppose we write out the fine-grained factorization of the matrix $\tilde{\mathcal{P}}_B^m = A_N \cdots A_2 A_1$. This automatically gives a sparse factorization of $\tilde{\mathcal{P}}_B^{mt} = A_1^t \cdots A_{N-1}^t A_N^t$; with the same total number of nonzero entries and hence the same number of calculations required to apply it to a vector. Therefore we see that the total complexity of applying $\tilde{\mathcal{P}}_B^m$ is also $O(B \log^2 B)$, as required for the fast inverse transform.

■

Taken together the fast forward and inverse transforms allow for the convolution of two band-limited functions in $L^2(S^2)$ to be computed efficiently - and exactly (in exact arithmetic).

**Theorem 6** *Let $f, g \in L^2(S^2)$ such that by $f$ and $g$ are band-limited with bandwidth $B$. Then the $n = O(B^2)$ sample values of the convolution $f \star g$ (cf. Section 2) at the points $(\theta_j, \phi_k)$ where $\theta_j = \frac{\pi(j+1/2)}{2B}$ and $\phi_k = \frac{\pi(k+1/2)}{B}$ may be computed in $O(n \log^2 n)$ operations, versus the $O(n^2)$ operations required by direct computation.*

**Proof:** Using the algorithm described by Theorem 3, compute the Fourier coefficients of $f$ and $g$. Compute the pointwise products according to Theorem 1 for the convolution $f \star g$ and finally, compute the inverse transform for the Fourier coefficients of the convolution according to Theorem 5.

■

# 5 Numerical Considerations

Sections Three and Four show that for band-limited functions in $L^2(S^2)$, in exact arithmetic, the forward and inverse spherical Fourier transforms as well as convolutions, can be computed exactly and efficiently. However, theoretical performance is but one yardstick by which an algorithm can be measured. As we are strongly motivated by applying the methods here to specific computational tasks, we are also interested in numerical reliability. In this section we examine the stability of our algorithm, primarily from the point of view of numerical experimentation and eventually derive a modified but more reliable version of the fast forward transform of Section 3. We begin by identifying potentially numerically unreliable aspects of our algorithm and proceed to give heuristics for stabilizing the computations. These methods are justified with data from our current implementation in C, which runs on a DecStation 5000. The methods we use do not noticeably affect the theoretical or actual running time of the algorithm. The heuristics are then quantified in the results of Section 5.2 which presents the modified algorithm. Section 5.3 shows that the modifications do not seriously affect the complexity estimates of Section 3.

We note here that all of the numerical experiments were carried out using the $L^2$-normalized associated Legendre functions $\tilde{P}_l^m$ and the $L^2$-normalized recurrence for generation of shifted Legendre functions. In this section on numerical errors, for notational convenience we use the notation $P_l^m$ to indicate the $L^2$-normalized functions.

## 5.1 Sources of Numerical Errors

The goal of this section is to illustrate what appear to be the main sources of numerical errors in a direct implementation of the fast forward transform described in Section 3.3, as well as to present, in some detail, techniques for (greatly) lessening these deleterious effects.

In [20], an extensive error analysis, both a priori and through numerical experimentation, indicates that direct application of the algorithm for fast computation of Legendre polynomial transforms will be reliable. This is effectively the problem of computing $\hat{f}(l, 0)$ only.

In the case of nonzero $m$, it seems that direct implementation will not be reliable. This statement is based (as will be shown) on numerical experimentation, in which experience shows that the majority of any error incurred seems to come from the "first step", the application of the sequence of operators

$$\mathcal{M}_0 \cdot \mathcal{I} f.$$

If we rewrite the input operator $\mathcal{I}$ as

$$\begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P_m^m \\ P_{m+1}^m \end{pmatrix} \begin{pmatrix} I \\ I \end{pmatrix} \tag{34}$$

where $P_m^m$ and $P_{m+1}^m$ represent the $2B \times 2B$ diagonal matrices with diagonal entries $P_m^m(\theta_j)$, and $P_{m+1}^m(\theta_j)$ respectively. Then using equations (9) and (10) the matrix product $\mathcal{M}_0 \cdot \mathcal{I}$ becomes

$$\Pi_0 \begin{pmatrix} \mathcal{F}^{-1} & 0 \\ 0 & \mathcal{F}^{-1} \end{pmatrix} \begin{pmatrix} -\frac{2m+1}{2} P_{\frac{B}{2}-2}^m(2, \cos\sigma) & P_{\frac{B}{2}-1}^m(1, \cos\sigma) \\ -\frac{2m+1}{2} P_{\frac{B}{2}-1}^m(2, \cos\sigma) & P_{\frac{B}{2}}^m(1, \cos\sigma) \end{pmatrix} \begin{pmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{pmatrix} \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P_m^m \\ P_{m+1}^m \end{pmatrix} \tag{35}$$

Application of the last two factors $\Pi_0$ and $\left(\begin{smallmatrix} \mathcal{F}^{-1} & 0 \\ 0 & \mathcal{F}^{-1} \end{smallmatrix}\right)$ can introduce little new error. The former is a projection operator, so it is exact, and the latter is block diagonal with inverses of the DFT matrix on the diagonal which, if computed by any of a variety of FFT algorithms, displays great stability (cf. [46, 11] and references therein). Consequently, if error is to arise in this initial step it will almost surely come from the application of the first four operators,

$$\begin{pmatrix} -\frac{2m+1}{2} P_{\frac{B}{2}-2}^m(2, \cos\sigma) & P_{\frac{B}{2}-1}^m(1, \cos\sigma) \\ -\frac{2m+1}{2} P_{\frac{B}{2}-1}^m(2, \cos\sigma) & P_{\frac{B}{2}}^m(1, \cos\sigma) \end{pmatrix} \cdot \begin{pmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{pmatrix} \cdot \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \cdot \begin{pmatrix} P_m^m \\ P_{m+1}^m \end{pmatrix}. \tag{36}$$

It is worth mentioning the intuition behind the coming discussion. The interior product $\left(\begin{smallmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{smallmatrix}\right)$ is essentially the identity (cf. Theorem 7). Thus the effect of the successive applications of the matrices in (36) is to first apply the matrix $\left(\begin{smallmatrix} P_m^m \\ P_{m+1}^m \end{smallmatrix}\right)$ to the data and follow this by applying the matrix

$$\begin{pmatrix} -\frac{2m+1}{2} P_{\frac{B}{2}-2}^m(2, \cos\sigma) & P_{\frac{B}{2}-1}^m(1, \cos\sigma) \\ -\frac{2m+1}{2} P_{\frac{B}{2}-1}^m(2, \cos\sigma) & P_{\frac{B}{2}}^m(1, \cos\sigma) \end{pmatrix}.$$

The result of such a succession of multiplications is the potential introduction of a most egregious error. We illustrate this with some examples.

Consider the case where the bandwidth $B = 128$ and order $m = 16$. The functions $P_{16}^{16}(x)$ and $P_{17}^{16}(x)$, shown in Figures 2(a) and (b), have the majority of their numerical support localized near zero. On the other hand, the shifted Legendre functions $P_{62}^{16}(2, x)$, $P_{63}^{16}(1, x)$, $P_{63}^{16}(2, x)$, and $P_{64}^{16}(1, x)$, shown in Figures 2(c), (d), (e), and (f), have their numerical support localized near 1 and -1, with very large magnitudes.

Consequently, if $\mathbf{s} = (s_0, \ldots, s_n)$ denotes some sequence sampled between $-1$ and 1 at the points $\cos\sigma$ then the effect of applying the matrices in Eq. (36) to $\mathbf{s}$ is to first drive the values sampled close to 1 and $-1$ to nearly zero by multiplication by $P_{16}^{16}(x)$ and $P_{17}^{16}(x)$, and then subsequently inflate the same elements back to their original range by multiplication by shifted associated Legendre functions, which have very large magnitudes near 1 and -1. Similarly, the shifted associated Legendre functions have small values near $x = 0$ where $P_{16}^{16}(x)$ and $P_{17}^{16}(x)$ have their numerical support. Because of the large dynamic range of the numbers involved in these operations, rounding errors are almost sure to be incurred when fixed-precision arithmetic is used.

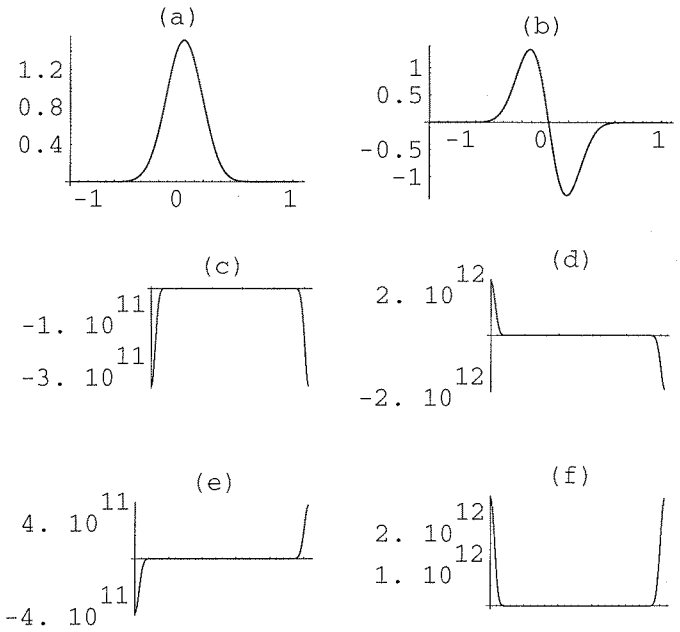



Figure 2: (a) $P_{16}^{16}(x)$, $x \in [-1,1]$; (b) $P_{17}^{16}(x)$; (c) $P_{62}^{16}(2,x)$; (d) $P_{63}^{16}(1,x)$; (e) $P_{63}^{16}(2,x)$; (f) $P_{64}^{16}(1,x)$

To demonstrate this, we first computed the shifted Legendre tridiagonal matrix

$$\begin{pmatrix} -\frac{2m+1}{2} P^m_{\frac{B}{2}-2}(2,\cos\sigma) & P^m_{\frac{B}{2}-1}(1,\cos\sigma) \\ -\frac{2m+1}{2} P^m_{\frac{B}{2}-1}(2,\cos\sigma) & P^m_{\frac{B}{2}}(1,\cos\sigma) \end{pmatrix}$$

for $m = 16$ and $B = 128$ as well as the functions $P_{16}^{16}(x)$ and $P_{17}^{16}(x)$ to double precision (16-digit) accuracy using *Mathematica*©, and then ported the matrices to a C language implementation on a DEC 5000 workstation. Then the sequence of operators

$$\begin{pmatrix} -\frac{2m+1}{2} P^m_{\frac{B}{2}-2}(2,\cos\sigma) & P^m_{\frac{B}{2}-1}(1,\cos\sigma) \\ -\frac{2m+1}{2} P^m_{\frac{B}{2}-1}(2,\cos\sigma) & P^m_{\frac{B}{2}}(1,\cos\sigma) \end{pmatrix} \begin{pmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{pmatrix} \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \tag{37}$$

were applied to $P_{16}^{16}(\cos\sigma)$ and $P_{17}^{16}(\cos\sigma)$ using double precision arithmetic ($P_{16}^{16}(\cos\sigma)$ and $P_{17}^{16}(\cos\sigma)$ were weighted by the appropriate quadrature weights – see Theorem 2). The result was compared to the result of a similar computation which was carried out in Mathematica using 128-digit precision. Figure 3(a) shows the error incurred by the multiplication of $P_{16}^{16}(x)$ to the weight function. Virtually no error is incurred by this step. Figures 3(b) and (c) graph the relative error incurred by the application of the shifted Legendre functions $P_{62}^{16}(2,x)$ and $P_{63}^{16}(1,x)$, respectively, and Figure 3(d) graphs the total relative error. All wraparound effects (cf. Section 5.2) have been removed, and from the graphs it is clear that most of the error in this first step is introduced by the application of the matrices of shifted Legendre functions. A similar error is incurred by the application of $P_{63}^{16}(2,x)$ and $P_{64}^{16}(1,x)$.

When this error is in turn multiplied by the subsequent operator $\left(\begin{smallmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{smallmatrix}\right)$ and mapped back to the frequency domain, it spreads over the entire $Z_{80}^{16}$, $Z_{81}^{16}$ sequences (cf. Eq. (13)). Since these sequences

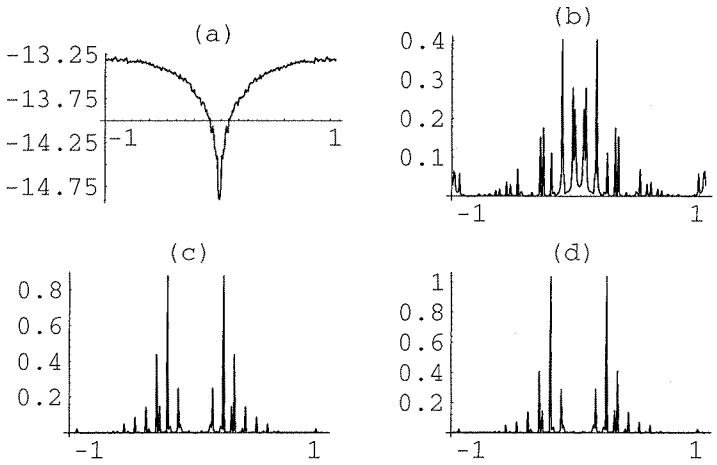



Figure 3: (a) Relative errors resulting from application of $P_{16}^{16}(x)$ to the weighting function. Scale is $\log_{10}$, indicating that virtually no error is incurred by this step; (b) Relative error incurred by the application of the shifted Legendre function $P_{62}^{16}(2,x)$; (c) Relative error incurred by the application of the shifted Legendre function $P_{63}^{16}(1,x)$; (d) total relative error of applying $P_{62}^{16}(2,x)$ and $P_{63}^{16}(1,x)$ blocks to weighted sequences $P_{16}^{16}(x)$ and $P_{17}^{16}(x)$. In all graphs, the effects have aliasing have been removed.

are used to compute almost half of the remaining Legendre coefficients, we expect that these Legendre coefficients will be inaccurate. To obtain preliminary confirmation for this, we applied the algorithm to samples of a synthesized function, all of whose Fourier coefficients are 1 up to bandwidth 128. This is a reasonable initial test case as it is a (suitably normalized) maximally oscillatory band-limited function on the sphere. The results of this test case may be found in Figure 4(a) which shows the expected high degree of relative error, increasing as the degree of the computed coefficient. In Figure 4(b), the $\log_{10}$ scale is used, showing more clearly the regular increase in the error as the degree of the Legendre coefficients increase.

As $m$ increases, the errors increase rapidly. Figure 5 shows the relative errors for similar computations as displayed in Figure 3(d) but for $m = 24$ and $m = 32$. The errors are so large here as to render the results virtually meaningless.

One possible way in which to potentially lessen the effect of applying these numerically opposed operators is to combine them into a single operator which may be applied at once. Again, working from the assumption that the product of $\mathcal{F}\ \mathcal{C}$ is effectively the identity, some insight into why the reliability is improved comes from considering that

$$\begin{pmatrix} P^m_{m+\frac{B}{2}}(x) \\ P^m_{m+\frac{B}{2}+1}(x) \end{pmatrix}_{2B\times 1} = \begin{pmatrix} -\frac{1}{2}P^m_{\frac{B}{2}-2}(2,x) & P^m_{\frac{B}{2}-1}(1,x) \\ -\frac{1}{2}P^m_{\frac{B}{2}-1}(2,x) & P^m_{\frac{B}{2}}(1,x) \end{pmatrix} \begin{pmatrix} P^m_m(x) \\ P^m_{m+1}(x) \end{pmatrix}_{2B\times 1}. \tag{38}$$

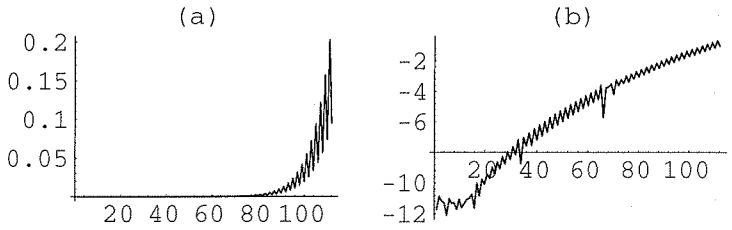



Figure 4: (a) Relative errors in computed Legendre coefficients when input is a synthesized function with all Legendre coefficients equal to 1; (b) Same as (a), but plotted on a $\log_{10}$ scale

where $x$ is any vector of samples between $-1$ and 1. Note that (38) follows directly from the linear algebraic formulation of the computation of $P^m_{m+\frac{B}{2}}(x)$ and $P^m_{m+\frac{B}{2}+1}(x)$ using the three-term recurrence rule.

The direct application of the lefthand side of (38) would appear to be much less dangerous than the successive application of the factors of the righthand side. Using $P^{16}_{80}(x)$, $Z^{16}_{80}$ can be computed directly. Figure 6 plots $P^{16}_{80}(x)$. Compare this to the functions shown in Figure 2 which are used to compute $Z^{16}_{80}$ by equations (35).

The point is that by directly applying $P^{16}_{80}(x)$, we can avoid the problem of applying operators with large shifts in dynamic range, thereby gaining some improvement in the numerical properties of the algorithm. Thus, we now examine carefully how the initial steps combine, leading us to a modified algorithm with improved numerical properties.

## 5.2 Treating the Instability - a Modified Algorithm

In the previous section we have indicated a potential numerical problem with using a straightforward application of the matrix factorization to compute the forward transform. The main point of Section 5 was to motivate the idea that serious instabilities in the direct computation of the forward transform by the factorization (35) can be avoided by combining the input operator $\mathcal{I}$ with the initial block diagonal convolution matrix. As we have tried to suggest, the effect of this combination should result

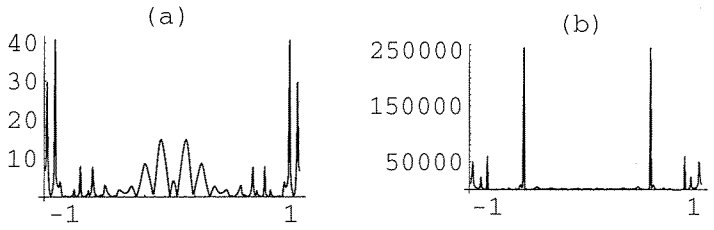



Figure 5: (a) Relative errors resulting from application of $P_{62}^{24}(2,x)$ and $P_{63}^{24}(1,x)$ blocks; (b) Relative errors resulting from application of $P_{62}^{32}(2,x)$ and $P_{63}^{32}(1,x)$ blocks.

in, essentially, the direct application of the matrix

$$\begin{pmatrix} P^m_{m+\frac{B}{2}} \\ P^m_{m+\frac{B}{2}+1} \end{pmatrix}$$

to the initial input $f$; further, that this should provide a more numerically reliable way to compute the initial step of the algorithm. More precisely we prove the following result.

**Theorem 7** *Let $f$ be a sequence of length $2B$. Let $\mathcal{F}$ and $\mathcal{C}$ be the $2B \times 2B$ DFT and DCT matrices as defined previously. Then*

$$\begin{pmatrix} -\frac{2m+1}{2} P^m_{\frac{B}{2}-2}(2,\cos\sigma) & P^m_{\frac{B}{2}-1}(1,\cos\sigma) \\ -\frac{2m+1}{2} P^m_{\frac{B}{2}-1}(2,\cos\sigma) & P^m_{\frac{B}{2}}(1,\cos\sigma) \end{pmatrix} \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} P^m_m \\ P^m_{m+1} \end{pmatrix} \cdot f =$$

$$\begin{pmatrix} P^m_{m+B/2}(\cos\sigma) \\ P^m_{m+B/2+1}(\cos\sigma) \end{pmatrix} \cdot Lf_e + \begin{pmatrix} A \\ A' \end{pmatrix}$$

*where the vector $Lf_e$ is a* **decimated low-pass even extension** *(cf. (39) and (45) below) of the data $f$, $\cos(\sigma)$ is the diagonal matrix with entries $\{\cos(\pi(k+1/2)/B) | k = -B, ...B-1\}$, and the vectors $A$ and $A'$ are "error" vectors and have the property that upon application of the remaining matrices in the factorization (8), they have no effect on the ultimate computation of the NALT and thus, may be ignored in the subsequent computation.*

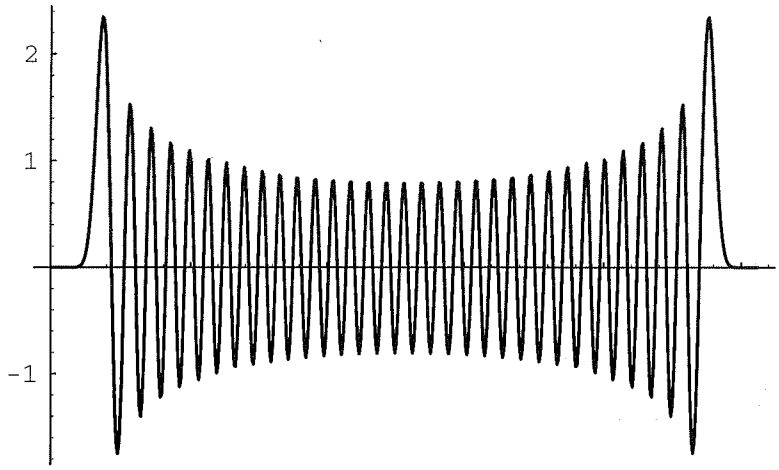



Figure 6: The function $P_{80}^{16}(x)$ shown here does not display the large shifts in dynamic range which are characteristic of the functions $P_{16}^{16}(x)$, $P_{16}^{17}(x)$, $P_{62}^{16}(2,x)$, and $P_{63}^{16}(1,x)$ which can be used to compute it. Compare to Figure 1.

It is worth remarking here that the effect of this theorem is to set up a **different** algorithm for the forward transform - although as we will see, one of very nearly the same complexity. In this modified algorithm, operators are combined differently and additional output is introduced. Fortunately, as we shall see, this additional output does not affect the final output of the NALT algorithm.

To prove Theorem 7, we must introduce the notions of **even extension** as well as the **decimated low-pass** and **high-pass** operators on a sequence.

Given a sequence $\mathbf{s} = (s(0), \ldots, s(n-1))$ define the **even extension of s**, to be a sequence of length $2n$, denoted $\mathbf{s}_e = (s_e(-n), \ldots, s_e(n-1))$ and defined by

$$s_e(j) = \left\{ \begin{array}{ll} 0 & j = -n \\ s_{-j} & -n < j < 0 \\ 2s_0 & j = 0 \\ s_j & 0 < j < n. \end{array} \right\} \tag{39}$$

**Lemma 1** *Let* $\mathbf{s}_e$ *be the even extension of some real-valued sequence* $\mathbf{s}$*, where* $\mathbf{s} = (s(0), \ldots, s(B-1))$*, as per (39); then* $\mathcal{F}\mathbf{s}_e = \mathcal{F}^{-1}\mathbf{s}_e$ *when the rows of* $\mathcal{F}$ *are ordered to compute the sequence*

$$(\hat{s}(-B), \ldots, \hat{s}(0), \ldots, \hat{s}(B-1)),$$

*where* $\hat{s}(k)$ *is the Fourier transform of* $\mathbf{s}_e$ *computed at frequency* $k/2B$*.*

**Proof**: Consider the transform coefficient $\hat{s}(k)$ computed as the dot product of the appropriate row of $\mathcal{F}$ and $\mathbf{s}_e$, i.e.,

$$\hat{s}(k) = \sum_{j=-B+1}^{B-1} s_e(j) \exp(-i2\pi kj/2B). \tag{40}$$

Any pair of elements $s_e(-j)$ and $s_e(j)$ are multiplied by the complex conjugate pair $\exp(i2\pi kj/2B)$ and $\exp(-i2\pi kj/2B)$ and summed, but since $s_e(-j) = s_e(j)$,

$$s_e(-j)\exp(i2\pi kj/2B) + s_e(j)\exp(-i2\pi kj/2B) = 2s_e(j)\cos(2\pi kj/2B). \tag{41}$$

$\mathcal{F}$ is an orthogonal matrix; therefore, $\mathcal{F}^{-1}$ is the Hermitian of $\mathcal{F}$. After applying the Hermitian mapping, an element $\omega_k^j$ in $\mathcal{F}$ is mapped to $\omega_j^{-k}$ in $\mathcal{F}^{-1}$. If $\mathcal{F}^{-1}$ is applied to $\mathbf{s}_e$, then for the same row and same index $j$ as in (40),

$$\hat{s}(k) = \sum_{j=-B+1}^{B-1} s_e(j) \exp(i2\pi kj/2B). \tag{42}$$

Then for some index $j$,

$$s_e(-j)\exp(-i2\pi kj/2B) + s_e(j)\exp(i2\pi kj/2B) = 2s_e(j)\cos(2\pi kj/2B), \tag{43}$$

which is the same result as in (41). Since the frequency $k$ and index $j$ were arbitrarily chosen, equality holds for any row, and therefore $\mathcal{F}\mathbf{s}_e = \mathcal{F}^{-1}\mathbf{s}_e$.

■

In matrix notation, the mapping from $\mathbf{s}$ to $\mathbf{s}_e$ is accomplished by applying the $2n \times n$ operator

$$E = \begin{pmatrix} & & & & & & 0 \\ & & & & & & 1 \\ & & & & & 1 & \\ & & & & \iddots & & \\ & & & 1 & & & \\ & & 2 & & & & \\ & & & 1 & & & \\ & & & & \iddots & & \\ & & & & & 1 & \\ & & & & & & 1 \end{pmatrix} \tag{44}$$

to the input sequence $\mathbf{s}$ represented as a column vector.

Finally, for any evenly extended sequence $\mathbf{s}_e$ define the **decimated low-pass** sequence as follows. Via the DFT we may write a Fourier series representation of $\mathbf{s}_e$ as

$$\begin{aligned} s_e(j) &= \frac{1}{2n} \sum_{k=-n}^{n-1} \hat{\mathbf{s}}_e(k) \exp(-2\pi ikj/2n) \\ &= \frac{1}{2n} \sum_{|k|\le n/2-1} \hat{\mathbf{s}}_e(k) \exp(-2\pi ikj/2n) + \frac{1}{2n} \sum_{n/2<|k|<n} \hat{\mathbf{s}}_e(k) \exp(-2\pi ikj/2n). \end{aligned}$$

Keeping only the first of these sums, comprised of terms corresponding to lower frequencies, gives a sequence which is "oversampled". Therefore, we define the decimated low-pass version of $s_e$ to be the half-length sequence obtained by keeping every other sample of the low frequency terms:

$$Ls_e(j) = \frac{1}{2n} \sum_{|k|\le n/2-1} \hat{\mathbf{s}}_e(k) \exp(-2\pi ik(2j)/2n) \tag{45}$$

With this notation in place we see that the cosine transform can be rewritten as an even extension, followed by a DFT, followed by a low-pass projection.

**Lemma 2** *The following equality holds,*

$$\mathcal{C} = \Pi_B \cdot \mathcal{F}_c \cdot E \tag{46}$$

*where $E$ is as in (44), $\mathcal{F}_c$ is the "near-DFT" matrix defined in Eq. (47) below, and $\Pi_B$ is a low-pass projection operator (as defined below).*

**Proof:** First define

$$\mathcal{F}_c = \begin{pmatrix} \omega_{2B-1}^{-2B} & \omega_{2B-1}^{-2B+1} & \omega_{2B-1}^{-2B+2} & \cdots & \omega_{2B-1}^{2B-1} \\ \omega_{2B-2}^{-2B} & \omega_{2B-2}^{-2B+1} & \omega_{2B-2}^{-2B+2} & \cdots & \omega_{2B-2}^{2B-1} \\ & & \vdots & & \\ \omega_{1}^{-2B} & \omega_{1}^{-2B+1} & \omega_{1}^{-2B+2} & \cdots & \omega_{1}^{2B-1} \\ \omega_{0}^{-2B} & \omega_{0}^{-2B+1} & \omega_{0}^{-2B+2} & \cdots & \omega_{0}^{2B-1} \\ \omega_{-1}^{-2B} & \omega_{-1}^{-2B+1} & \omega_{-1}^{-2B+2} & \cdots & \omega_{-1}^{2B-1} \\ & & \vdots & & \\ \omega_{-2B+2}^{-2B} & \omega_{-2B+2}^{-2B+1} & \omega_{-2B+2}^{-2B+2} & \cdots & \omega_{-2B+2}^{2B-1} \\ \omega_{-2B+1}^{-2B} & \omega_{-2B+1}^{-2B+1} & \omega_{-2B+1}^{-2B+2} & \cdots & \omega_{-2B+1}^{2B-1} \\ \omega_{-2B}^{-2B} & \omega_{-2B}^{-2B+1} & \omega_{-2B}^{-2B+2} & \cdots & \omega_{-2B}^{2B-1} \end{pmatrix} \tag{47}$$

where $\omega_j = e^{\frac{i2\pi j}{4B}}$. Then the projection operator $\Pi_B$ is defined as reading off the central portion $\{\sigma_{-B}, ..., \sigma_{B-1}\}$ from a sequence $\{\sigma_{-2B}, ..., \sigma_{2B-1}\}$. In particular, it has the following effect on the even extension of an arbitrary sequence $\mathbf{s}$ of length $2B$,

$$\begin{pmatrix} \hat{s}_{B-1} \\ \hat{s}_{B-2} \\ \vdots \\ \hat{s}_1 \\ \hat{s}_0 \\ \hat{s}_{-1} \\ \vdots \\ \hat{s}_{-B+2} \\ \hat{s}_{-B+1} \\ \hat{s}_{-B} \end{pmatrix} = \frac{1}{2}\Pi_{lp}\mathcal{F}_c \begin{pmatrix} 0 \\ s_{2B-1} \\ \vdots \\ s_2 \\ s_1 \\ 2s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{2B-2} \\ s_{2B-1} \end{pmatrix} \tag{48}$$

where $\hat{s}$ denotes cosine transform (for the present). Note that $\hat{s}_j = \hat{s}_{-j}$ since $\cos(-\theta) = \cos(\theta)$. ■

With these facts and definitions in hand we can now proceed to the proof of Theorem 7.

**Proof** (of Theorem 7): We begin the analysis with the discrete sequences $f$ and $P_m^m$, each of length $2B$. It is useful for this proof for a sequence $\mathbf{s}$ of length $n$, to denote by

$$\mathrm{diag}(\mathbf{s})$$

the $n \times n$ diagonal matrix with diagonal entries $s(j)$.

Using Lemmas 1 and 2, the first few steps of the algorithm compute a low-passed Fourier transform of the even extension of the pointwise product $(\mathrm{diag}(P_m^m)f)$,

$$\mathcal{C}P_m^m f \equiv \Pi_B \mathcal{F}_c (\mathrm{diag}(P_m^m)f)_e \tag{49}$$

where $\Pi_B$ is as defined above.

The righthand side of (49) is projection of the DFT of a pointwise product of sequences. This may in turn be written as the cyclic convolution of the individual DFT's of each of the sequences. Thus if $\mathcal{P} = \mathcal{F}(P_m^m)_e$, the Fourier transform of $(P_m^m)_e$, and let $\mathcal{D} = \mathcal{F}f_e$, the Fourier transform of $f_e$, then

$$\mathcal{F}_c(P_m^m)_e f_e = \mathcal{P} \star \mathcal{D}. \tag{50}$$

Here the righthand side indicates the cyclic convolution of sequences of length $4B$.

The discussion simplifies somewhat if we now assume that $m$ is even (odd $m$ is treated in the proof of Theorem 8 in Section 5.3). In this case, since

$$P_m^m(\cos\theta) \propto \sin^m(\cos\theta) = (1 - \cos^2(\cos\theta))^{\frac{m}{2}}$$

we see that the DFT of $(P_m^m)_e$ has nonzero frequencies only between $B$ and $-B$. In general, for a sequence $\mathbf{s} = (s(-n), \ldots, s(0), \ldots, s(n-1))$, let $\mathbf{s}_{[k]}$ denote the projected sequence

$$\mathbf{s}_{[k]}(j) = \begin{cases} 0 & \text{if } |j| \geq k; \\ s(j) & \text{otherwise.} \end{cases}$$

Thus, we may write for $m$ even,

$$\mathcal{P} = \mathcal{P}_{[B]} = \mathcal{P}_{[m+1]}$$

A useful consequence of this is that within the subrange $-B + m < l < B - m$ the convolution may be computed from the reduced projection of the data, $\mathcal{D}_{[B]}$

$$(\mathcal{P} \star \mathcal{D})(l) = (\mathcal{P}_{[B]} \star \mathcal{D}_{[B]})(l), \qquad \text{for } -B + m < l < B - m \tag{51}$$

since

$$(\mathcal{P} \star \mathcal{D})(l) = \sum_{j=-m}^{m} \mathcal{P}(j)\mathcal{D}(l-j). \tag{52}$$

where the range of summation is reduced to the support of $\mathcal{P}$.

For indices $l$ not in the range $[-B + m, B - m]$ the equality (51) does not hold. For these indices, computing with the truncated data $\mathcal{D}_{[B]}$ instead of the actual data introduces an error. Because this error is confined to the outer parts of the sequence it will turn out that it may be effectively ignored without affecting the end goal of computing the NALT, a fact that we will show later.

More formally, we have obtained

$$\mathcal{C}P_m^m f = \Pi_B \mathcal{F}_c[(P_m^m)_e f_e] = \Pi_B(\mathcal{P} \star \mathcal{D}) = \Pi_B((\mathcal{P}_{[B]} \star \mathcal{D}_{[B]}) + \mathcal{A}), \tag{53}$$

where $\mathcal{A}$ is the error vector. However, as noted in (51), $(\mathcal{P} \star \mathcal{D})(l) = (\mathcal{P}_B \star \mathcal{D}_B)(l)$ when $-B + m \leq l \leq B - m$, so the support of $\mathcal{A}$ must be localized to $[-B-1, -B+m] \cup [B-m, B-1]$.

In the next step of the NALT algorithm, the sequence we have obtained so far, which is precisely $Z_m^m$, is convolved with sequences which represent the Fourier transforms of shifted Legendre functions. We perform this convolution using the DFT; this means that our sequence is next hit by a DFT operation, $\mathcal{F}$. So applying Lemmas 1 and 2 we can now write the first few steps of the factorization as

$$\mathcal{F}\mathcal{C}(P_m^m f) = \mathcal{F}^{-1}\Pi_B(\mathcal{P}_{[B]} \star \mathcal{D}_{[B]}) + \mathcal{F}^{-1}\mathcal{A} = (\mathrm{diag}(LP_m^m) \cdot Lf)_e + A, \tag{54}$$

where $L(P_{m_e}^m)$ and $Lf_e$ are the decimated low-passed versions of $P_{m_e}^m$ and $f_e$, and $A$ is the high-frequency error whose Fourier transform is $\mathcal{A}$. Note that since $m < B$, then $\mathcal{P} = \mathcal{P}_{[B]}$, and therefore $(LP_m^m)_e$ coincides with a subsampled version of $(P_m^m)_e$. One of the implications of (54) is that the

sequences $Lf$ and $LP_m^m$, which can be used to compute $Z_m^m$ (up to the error $A$) are now of length $B$, compared to $f$ and $P_m^m$ which are of length $2B$. Since $Z_m^m$ was originally defined in (13) as

$$Z_m^m(k) = \langle f, C_k P_m^m \rangle, \ |k| \leq B, \tag{55}$$

with the inner product defined as a sum over $2B$ terms, then (54) says that $Z_m^m$ can be computed using sequences of *half the length* of those used in (13), up to an error that we will show is irrelevant to the final result. In Section 5.3, we will exploit this reduction in size to improve the efficiency of a technique for achieving numerical reliability.

If the preceding analysis is performed with the function $P_{m+1}^m$ in place of $P_m^m$, a similar result holds:

$$\mathcal{FC}(P_{m+1}^m f) = \mathcal{F}^{-1}\Pi_B(\mathcal{P}_{1_{[B]}} \star \mathcal{D}_{[B]}) + \mathcal{F}^{-1}\mathcal{A}_1 = (LP_{m+1}^m \cdot Lf)_e + A_1, \tag{56}$$

where $\mathcal{P}_1 = \mathcal{F}(P_{m+1}^m)_e$, the Fourier transform of $(P_{m+1}^m)_e$, and $\mathcal{A}_1$ is the Fourier transform of $A_1$. Thus, collecting the above discussion we may write

$$\begin{pmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{F} \end{pmatrix} \begin{pmatrix} \mathcal{C} & 0 \\ 0 & \mathcal{C} \end{pmatrix} \begin{pmatrix} \mathrm{diag}(P_m^m) \\ \mathrm{diag}(P_{m+1}^m) \end{pmatrix} f = \left( \begin{pmatrix} \mathrm{diag}(LP_{m_e}^m) & 0 \\ 0 & \mathrm{diag}(LP_{m+1_e}^m) \end{pmatrix} \begin{pmatrix} Lf_e \\ Lf_e \end{pmatrix} \right) + \begin{pmatrix} A \\ A_1 \end{pmatrix}. \tag{57}$$

The next step of the matrix factorization applies to this the tridiagonal matrix of shifted Legendre functions. Application of the tridiagonal matrix of shifted Legendre functions now produces

$$\begin{pmatrix} -\frac{2m+1}{2} P_{\frac{B}{2}-2}^m(2, \cos\sigma) & P_{\frac{B}{2}-1}^m(1, \cos\sigma) \\ -\frac{2m+1}{2} P_{\frac{B}{2}-1}^m(2, \cos\sigma) & P_{\frac{B}{2}}^m(1, \cos\sigma) \end{pmatrix} \left[ \begin{pmatrix} LP_{m_e}^m & 0 \\ 0 & LP_{m+1_e}^m \end{pmatrix}_{4B\times 4B} \begin{pmatrix} Lf_e \\ Lf_e \end{pmatrix}_{4B\times 1} \right] + \begin{pmatrix} A \\ A_1 \end{pmatrix} \tag{58}$$

$$= \begin{pmatrix} P_{m+B/2}^m(\cos\sigma) \\ P_{m+B/2+1}^m(\cos\sigma) \end{pmatrix} \cdot Lf_e + \begin{pmatrix} A' \\ A_1' \end{pmatrix}. \tag{59}$$

This shows that the first few steps of the NALT algorithm may be combined into one application of diagonal Legendre function matrices applied to a low-passed version of the data, up to an error vector with the vectors $A'$ and $A_1'$ obtained from the application of the shifted Legendre function matrix to the error vectors $A$ and $A_1$.

To finish the proof, we must deliver on previous promises to demonstrate the harmlessness of these error terms. We'll show that they consist of high frequency terms, which is what we will need to insure they do not impact adversely the output of the algorithm. The Fourier transforms $\mathcal{A}$ and $\mathcal{A}_1$ of $A$ and $A_1$ are supported in $[-B, -B+m] \cup [B-m, B]$; the support of the Fourier transforms $\mathcal{F}(A')$ and $\mathcal{F}(A_1')$ of the new error vectors $A'$ and $A_1'$ can be analyzed in terms of the bandwidths of the shifted Legendre functions. Indeed, the new errors are obtained by pointwise multiplication of the vectors $A$ and $A_1$ by the sample sequences of the shifted Legendre functions. This corresponds to convolution in the Fourier domain, which spreads the frequency support of $\mathcal{A}$ and $\mathcal{A}_1$ by an amount bounded by the bandwidth of the shifted Legendre functions.

The bandlimit of the Fourier transforms of the sequences representing the shifted Legendre functions depends on the order $m$. For example, if $m < B/2 - 1$, then the bandlimit of the sequences will be $B/2$; if $B/2 < m < 3B/4 - 1$, then there is no need to compute $Z_{m+B/2}^m$, since $m + B/2 > B$. Instead, the sequence $Z_{m+B/4}^m$ will be computed, where the corresponding shifted Legendre sequence transforms are bandlimited to $B/4$. We will assume that $m < B/2 - 1$, but for other values of $m$, the appropriate modifications should be clear.

Consequently, the support of the Fourier transforms $\mathcal{F}(A')$ and $\mathcal{F}(A'_1)$ of the new error vectors $A'$ and $A'_1$ are supported in $[-B, -B + B/2 + m] \cup [B - B/2 - m, B]$, since application of the shifted Legendre functions increases the size of the support of the errors by $B/2$.

We now show that the error vector above, which we introduce by replacing the first steps of the algorithm with a direct multiplication of the even extension of the sampled data by the Legendre functions, does not affect the results of the NALT when the balance of the steps in that algorithm are run as initially described.

The next step of the NALT algorithm is the application of a Fourier transform operator to the sequences in (59), followed by a low-pass operator $\Pi_{B/2}$. This throws out the highest frequency components of the error but still supports it in the localized range of $[-B/2, -B/2+m] \cup [B/2-m, B/2]$. At this stage we have computed the sequences $Z^m_{m+B/2}$ and $Z^m_{m+B/2+1}$, which are correct in the range $[-B/2 + m, B/2 - m]$ and $[-B/2 + m + 1, B/2 - m - 1]$, respectively.

The algorithm continues by again computing the Fourier transforms of $(Z^m_m)_{B/2}, (Z^m_m)_{B/2}, Z^m_{m+B/2}$ and $Z^m_{m+B/2+1}$, applying tridiagonal matrices of shifted Legendre functions bandlimited to $B/4$, Fourier transforming the results, and applying ideal low-pass filters with bandlimit $B/4$ to compute $Z^m_{m+B/4}, Z^m_{m+B/4+1}, Z^m_{m+3B/4}$, and $Z^m_{m+3B/4+1}$. The error is now supported in the range $[-B/4, -B/4+m] \cup [B/4 - m, B/4]$. This sequence of operations characterized by geometrically decreasing bandwidths $B/2, B/4, B/8, \ldots$ continues until the current bandwidth $B/2^k \leq m$. At this stage, the error extends over the entire range of the most recently computed $Z^m_{m+r}$ sequences, but *this does not occur until* $m + r \geq B$, which is outside the range of interest. Up to this point, the $0^{th}$ frequency component of each $Z^m_{m+r}$ sequence is *not* corrupted by error, and these components are the Legendre coefficients we desire. A pictorial representation of wraparound effects is shown in Figure 7. ■

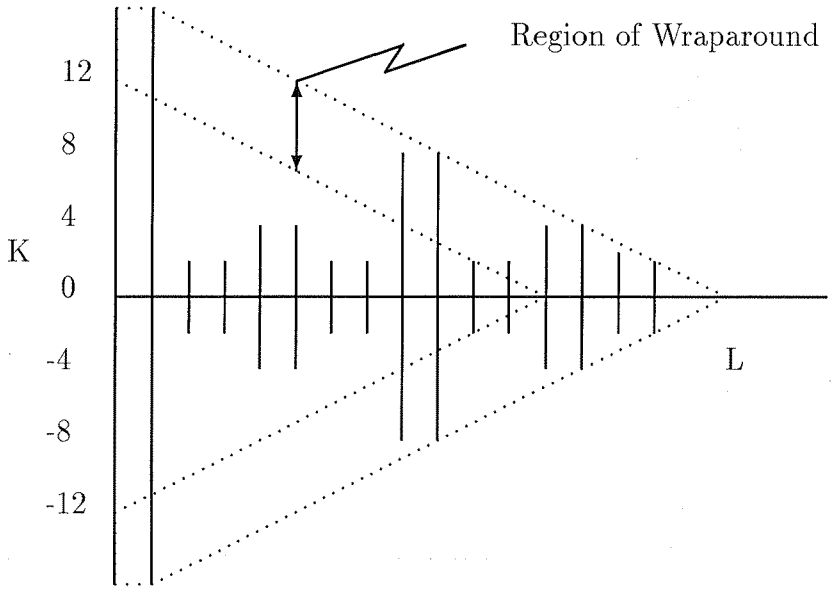



Figure 7: Wraparound effect for bandwidth = 16, m = 4. The range of the wraparound error for each $Z^4_{4+l}$ is represented by the dashed box. Note that for this problem, the first 12 Legendre coefficients are computed outside the support of the wraparound effect and are therefore unaffected by this error.

## 5.3 Reliability and Efficiency of Stability Bypass Operations

As we pointed out in Section 5.1, a straightforward implementation of the Driscoll-Healy algorithm demonstrates numerical problems as the order $m$ of the problem increases. In this section, we discuss a technique for improving the numerical reliability of the algorithm and demonstrate its performance with experiments. We then culminate this discussion with a proof that this gain in stability can be achieved without a significant loss in asymptotic complexity or run-time efficiency.

Earlier we pointed out that Eq. (55) can be used as an alternative to (35) because it bypasses a potential source of numerical errors and computes the sequences $Z^m_{m+B/2}, Z^m_{m+B/2+1}$ exactly in exact arithmetic, up to wraparound error. In fact, a more efficient method is revealed in the proof of Theorem 7, specifically in Equations 54 and 59. Instead of using $2B$ samples each of $f$ and $P^m_{m+B/2}$ to compute $Z^m_{m+B/2}$, only $B$ samples of $Lf$, the smoothed, decimated version of $f$, and $P^m_{m+B/2}$ are necessary to accurately compute the Legendre coefficients. More formally, $Z^m_{m+B/2}$ can be obtained as

$$Z^m_{m+B/2}(k) = \langle Lf, C_k P^m_{m+B/2} \rangle = \langle (Lf) P^m_{m+B/2}, C_k \rangle \tag{60}$$

Of course, the usefulness of (60) is contingent upon its numerical reliability, and our experiments have shown that this fixed precision computation is accurate. Continuing with our example from Section 5.1, where $B = 128$, $m = 16$, if we compute $Z^{16}_{80}$ and $Z^{16}_{81}$ directly using Eq. (60) and a fast cosine transform, the accuracy improves considerably (see Figure 8(a) and (b)). We will refer to this technique of directly computing $Z^m_{m+B/2}$ using Eq. (60) and a fast cosine transform as a *stability bypass* operation.

As the order $m$ continues to increase, however, more of the matrices $\mathcal{T}_j$ produce numerically inaccurate results. From Figure 8, it is apparent that the $\mathcal{T}_1$ matrix mapping $Z^{16}_{16}$, $Z^{16}_{17}$ to $Z^{16}_{48}$, $Z^{16}_{49}$ is introducing some numerical errors that are propagated to $Z$ sequences which use $Z^{16}_{48}$, $Z^{16}_{49}$ as input. While this error at $m = 16$ may be acceptable, as the degree $m$ increases, the errors become quite large. For example, as shown in Figures 9(a) and (b), at a bandwidth of $B = 128$ and order $m = 24$, numerical errors are $\approx O(10^{-1})$, and when $m = 32$, the errors are $\approx O(10^1)$.

Using similar experimentation as in Section 5.1, or by examining Figure 3, one concludes that a majority of this error is incurred in the step used to compute the sequences $Z^m_{m+B/4}$ and $Z^m_{m+B/4+1}$. It is natural then to ask if the stability bypass idea can be used efficiently to compute other sequences besides $Z^m_{m+B/2}$. Assuming that this is so (we show this in Theorem 8), the next question concerns the placement of these stability bypass operations in such a way as to minimize their number but to maximize their effectiveness in reducing numerical errors. If we examine the growth of the shifted Legendre functions as well as the recurrence relation, we can make some predictions regarding placement of bypass operations. From the estimates of the bounds of the shifted associated Legendre functions $P^m_{m+r}(v, x)$ (cf. Section 3.4), we know that the size increases as a function of $r$. Thus one expects that the matrices $\mathcal{T}_j(k)$ with small values of $j$ will produce numerical inaccuracies. This suggests a scheduling of bypass operations at $r = B/2, B/4, 3B/4, \ldots$ which is potentially $O(B)$ operations. On the other hand, the recurrence relation for the Legendre functions suggests that the matrices $\mathcal{T}_j(k)$ improve numerically as $v$ gets large, or that when $v > B/2$, the need for bypass operations decreases. A possible solution is the scheduling of $O(\log\ B)$ bypass operations at $r = 2, 4, 8, 16, \ldots, B/2$. We will see in Theorem 8 that this sequence of stability bypass operations can be performed in only $O(B\ \log\ B)$ time, but of course this result is useless unless the desired numerical accuracy is obtained.

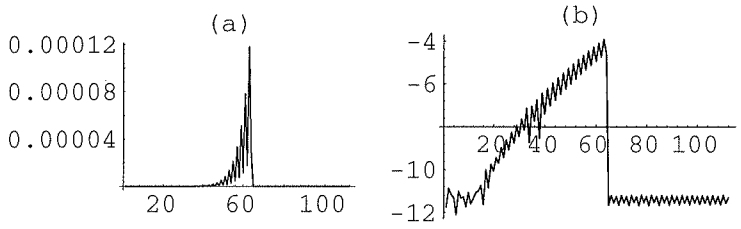



Figure 8: (a) Relative errors in Legendre series coefficients for a synthesized function with all Legendre coefficients equal to 1. A stability bypass operation was used to compute $Z_{80}^{16}$ and $Z_{81}^{16}$; (b) Same as (a), but graphed using a $\log_{10}$ scale, showing that the bypass operation improves the accuracy at this stage by approximately 8 orders of magnitude.

Figures 10 and 11 graph some of the numerical results using the stability bypass technique for $B = 256$ and $B = 512$. The number and placement of bypass operations is indicated in the legends. For bandwidths $B \leq 128$ and all $m < B$, the logarithmic scheduling scheme appears to be sufficient. For bandwidths $256 \leq B \leq 1024$, additional bypass operations are necessary, but a polylogarithmic scheduling scheme appears to be sufficient. Specifically, a $\log^2$ schedule (see Figure 12) was sufficient to maintain a high level of accuracy for these bandwidths.

At $B = 512$ and $B = 1024$, a few additional bypass operations are necessary as $m$ nears $B$, but the $O(\log^2)$ complexity of the scheduling appears to be maintained. Although the complexity of the algorithm increases with the polylogarithmic scheduling, the constant is small, and it appears that this technique does not seriously degrade the running time with respect to a naive $O(B^2)$ implementation. In fact, we found that in some cases the runtime efficiency was improved when more bypass operations than necessary to maintain numerical reliability were used at strategic locations. By determining experimentally where the breakpoint occurs, an implementation using a constant spacing of bypass operations for a given value of $B$ (with possibly some additional operations necessary for stability) can be *faster* in execution time than an implementation using only the bypass operations necessary for stability. In Figures 13 and 14, the (stable) fast implementation with constant spacing of bypass operations is compared to the naive algorithm for various $B$ and order $m$. These graphs indicate that the stabilized fast implementation becomes an attractive alternative at reasonable problem sizes.

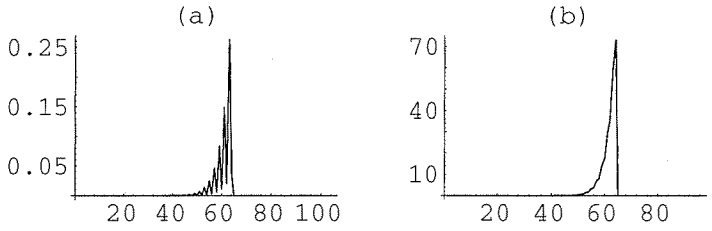


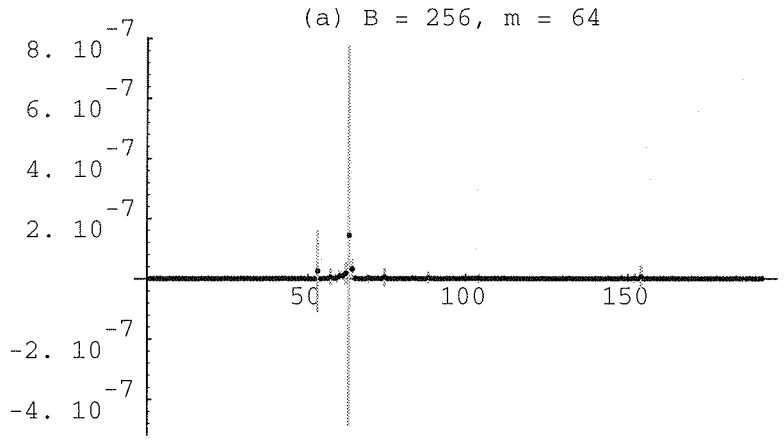

Figure 9: Relative errors in Legendre series coefficients for a synthesized function with all Legendre coefficients equal to 1 using a bypass operation. (a) $m = 24$; (b) $m = 32$.

Further testing showed that the fast inverse Legendre transform has nearly identical numerical properties and efficiency, as expected. The stability bypass technique can also be used to improve the numerical reliability. One way to derive the appropriate calculations is to rewrite the matrix formulation of the forward fast Legendre transform (cf. Section 3.3) to include the stability bypass operations, and then take the transpose of the formulation to get a stable inverse.

We close this section by pointing out that a careful, but relatively straightforward, analysis shows that if the stability bypass operations are scheduled at logarithmic spacing, this does not noticeably affect the complexity of the fast forward transform. If a polylogarithmic schedule is used, the complexity is still asymptotically better than a naive method. Thus we have not had to trade efficiency to gain reliability.

**Theorem 8** *(***Bypass Theorem***): Let $f$ be a sequence of length $2B$, thought of as being derived from samples of a bandlimited function of bandwidth $B$ on the sphere. Then the matrix-vector product $\tilde{P}_B^m \cdot f$ (cf. (6)) can be computed using the modified algorithm with stability bypass operations scheduled at logarithmic spacing in $O(B \log^2 B)$ operations. The additional time incurred by applying the stability bypass operations is at most $O(B \log B) + O(m \log^2 B)$. Furthermore, if polylogarithmic $(\log^k)$ scheduling of bypass operations is used, then the additional time incurred is $O(B \log^k B) + O(m \log^{k+1} B)$.*

**Proof**: Before proceeding, some notes are in order. Throughout this discussion, we will frequently refer to both a function $P(x)$ and a sequence $P$ corresponding to a set of discrete samples of $P(x)$.



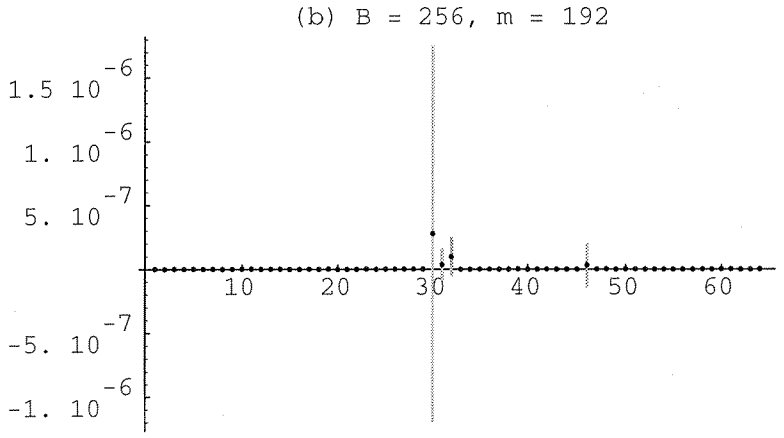




Figure 10: Relative errors of fast transform of synthesized functions with random normal Legendre coefficients (mean = 0, deviation = 1) computed over 30 trials. Black dots indicate mean relative error, and vertical gray bars indicate the standard deviation. Bypass operations were initially used to compute $Z^m_{m+r}$ for $r = 4, 8, 16, 32, 64, 128$, with additional operations added as necessary to keep errors below $\approx 10^{-6}$. (a) $B = 256$ and degree $m = 64$. Additional bypass operations used at $r = 96, 160$. (b) $B = 256$ and degree $m = 192$. Additonal bypass operation used at $r = 48$.

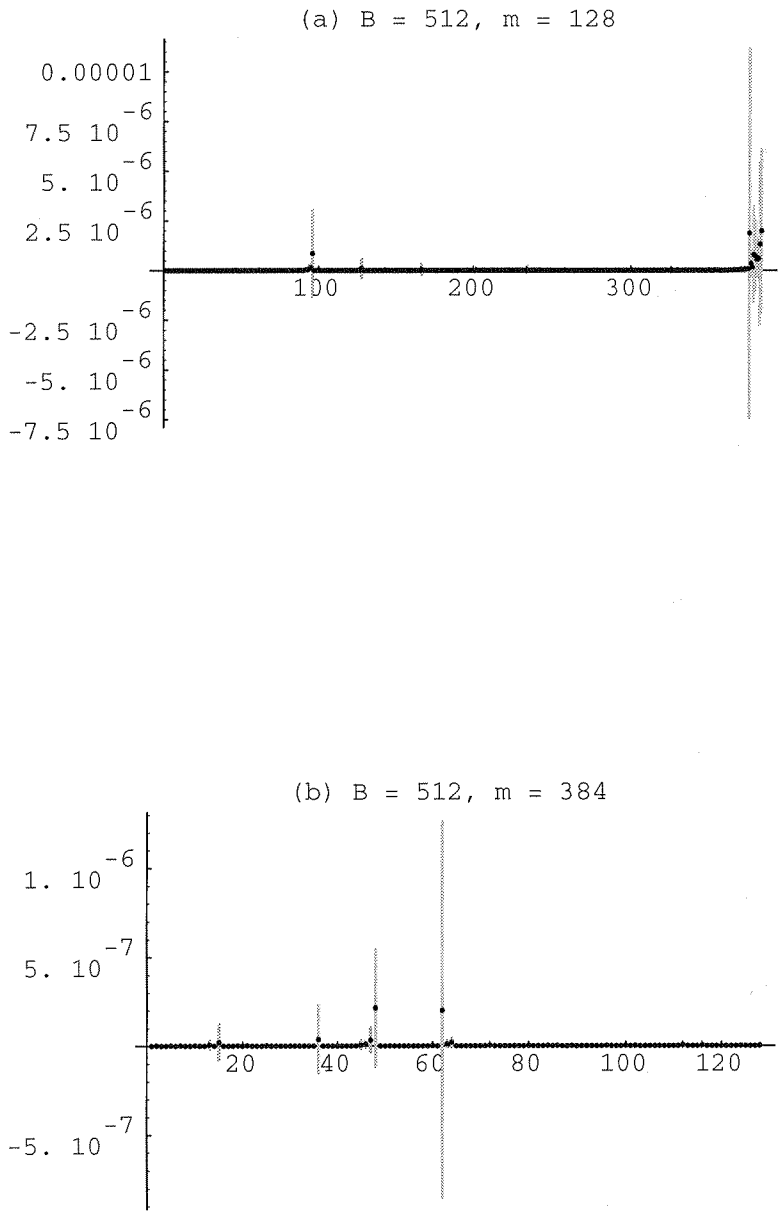



Figure 11: Relative errors using bypass operations trials similar to $B = 256$ case. Bypass operations initially placed at $r = 4, 8, 16, 32, 64, 128, 256$. (a) bandwidth $B = 512$ and degree $m = 128$. Additional bypass operators at $r = 48, 96, 160, 192, 224, 288, 320$. (b) bandwidth $B = 512$ and degree $m = 384$. Additional bypass operations at $r = 24, 48, 80, 96, 112$.

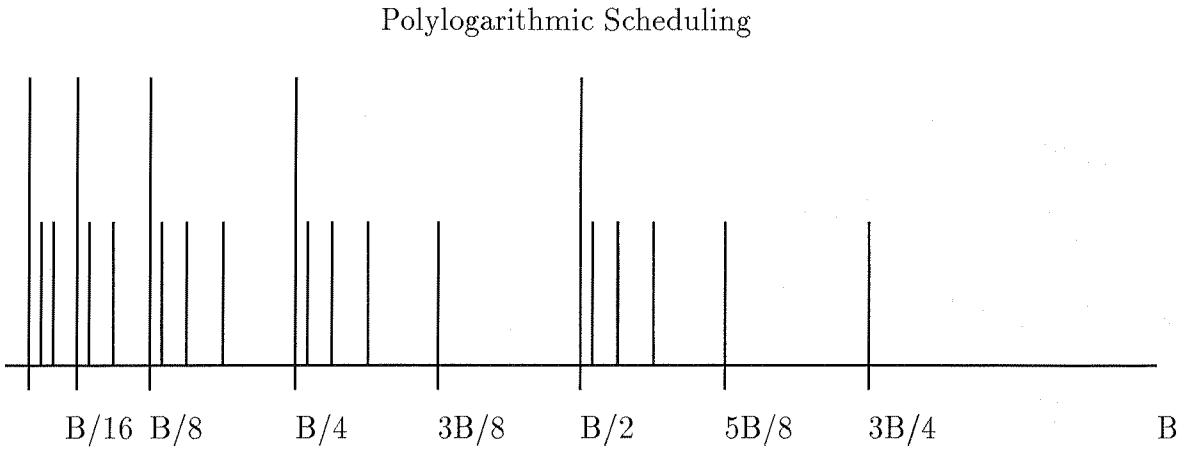



Figure 12: A polylogarithmic ($\log^2$)scheduling of stability bypass operations.

Unless otherwise stated, a sequence $P$ is generated by sampling $P(x)$ at the $2B$ points $\cos\theta_j$, where $\theta_j = \frac{\pi(2j+1)}{4B}$, $j = 0, 1, \ldots 2B-1$. Also, we denote the sinusoidal functions and their corresponding sequences by $\sin(\theta)$, $\cos(\theta)$ and $\sin\theta$, $\cos\theta$, respectively.

This proof will be divided into two parts, one for the case when $m$ is even, and the other when $m$ is odd. This division results from the fact that the discrete cosine transform of $P^m_{m+r}$ has $m+r$ non-zero coefficients when $m$ is even, but for $m$ odd this is not the case. To show this, note that when $m$ is even, the function

$$P^m_m(\cos(\theta)) \propto \sin^m(\theta) = (1-\cos^2(\theta))^{\frac{m}{2}}, \tag{61}$$

and therefore all coefficients of order greater than $m$ in the cosine series representation of the function $P^m_m(\cos(\theta))$ are zero. Similarly, the discrete cosine transform of the sequence $P^m_m$ has all coefficients greater than $m$ equal to zero. Using the three-term recurrence relation, it is easy to show that the discrete cosine transform of $P^m_{m+r}$ has the property that all coefficients of order greater than $(m+r)$ are zero.

If $m$ is odd, then

$$P^m_m(\cos(\theta)) \propto \sin(\theta)\sin^{m-1}(\theta) = \sin(\theta)(1-\cos^2(\theta))^{\frac{m-1}{2}}. \tag{62}$$

As we have noted, the discrete cosine transform of $P^m_m$ is equivalent to $\mathcal{F}(P^m_m)_e$, the discrete Fourier transform of the even extension of $P^m_m$. Then from Eq. (62), $\mathcal{F}(P^m_m)_e$ is equal to the convolution of $\mathcal{F}(\sin\theta)_e$, the Fourier transform of the even extension of $\sin\theta$, and $\mathcal{F}(\sin^{m-1}\theta)_e$, the Fourier transform of the even extension of $\sin^{m-1}\theta$. However, the cosine series representation of the function $\sin(\theta)$ is infinite; accordingly, the Fourier transform of the sequence $(\sin\theta)_e$ is not bandlimited. Convolving $\mathcal{F}(\sin\theta)_e$ and $\mathcal{F}(\sin^{m-1}\theta)_e$ results in a sequence $\mathcal{F}(P^m_m)_e$ which is not bandlimited. Clearly the sequences $P^m_{m+r}$ generated from $P^m_m$ by the three-term recurrence relation are also not bandlimited.

**Even $m$ case:**

It is useful to prove the logarithmic spacing ($k=1$) result first before proving the polylogarithmic ($\log^k$) case for $k>1$. If a logarithmic spacing of stability bypass operations is used, then the sequences $Z^m_{m+r}$ to be computed using bypass operations are $Z^m_{m+B/2}, Z^m_{m+B/4}, Z^m_{m+B/8}\cdots$, as well as the sequences

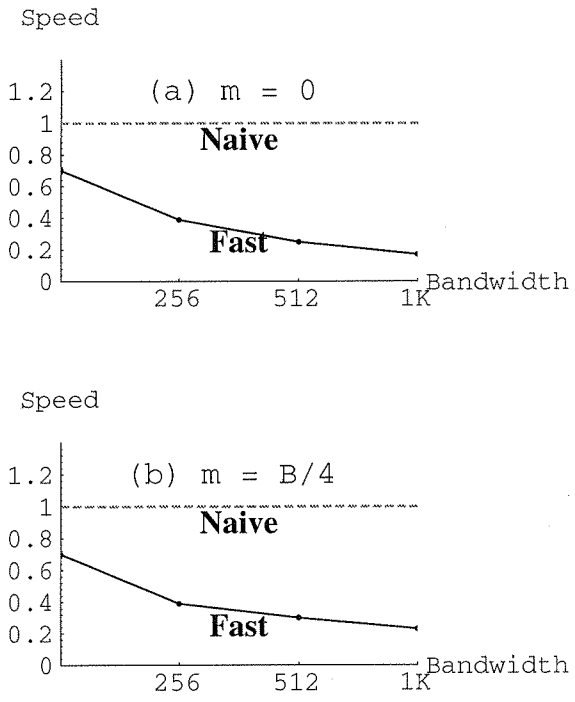



Figure 13: Ratio of the running time of the stabilized fast algorithm to the naive algorithm for various bandwidths $B$. (a) degree $m = 0$; (b) degree $m = B/4$. Algorithms implemented in C and run on a DEC Alpha workstation.

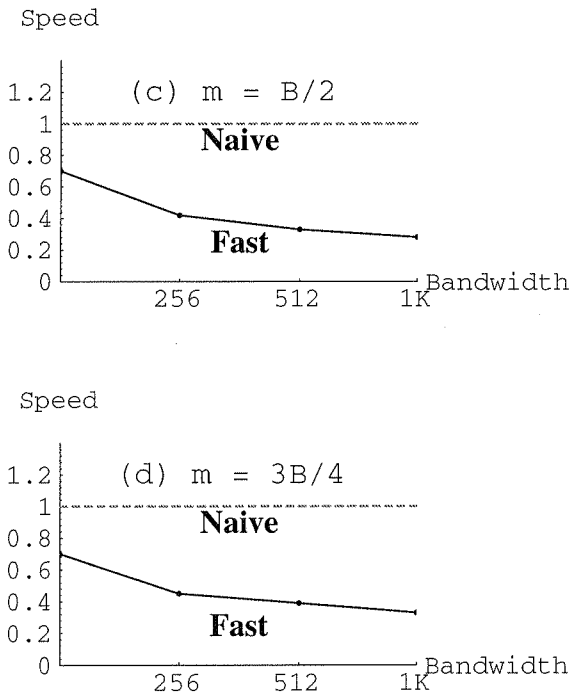



Figure 14: Ratio of the running time of the stabilized fast algorithm to the naive algorithm for various bandwidths $B$. (a) degree $m = B/2$; (b) degree $m = 3B/4$.

$Z^m_{m+B/2+1}, Z^m_{m+B/4+1}, Z^m_{m+B/8+1} \cdots$ where we recall that

$$Z^m_{m+r}(k) = \langle f, P^m_{m+r} C_k \rangle = \langle f P^m_{m+r}, C_k \rangle \tag{63}$$

for $k$ in some specified range $[-j, j]$. The computation of the second set of sequences $Z^m_{m+B/2+1}, Z^m_{m+B/4+1} \cdots$ is of the same complexity as the set $Z^m_{m+B/2}, Z^m_{m+B/4} \cdots$, so we will disregard it in the discussion because computing these sequence only increases the overall complexity by a constant.

In Section 5.2, we showed that Eq. (63) can be computed as the DFT of the even extension of $f \cdot P^m_{m+r}$. As noted above, the even extensions of the sequences $P^m_{m+r}$ are bandlimited to $[-(m+r), m+r]$. As before, let $\mathcal{D} = \mathcal{F} f_e$, the Fourier transform of the even extension of $f$, and let $\mathcal{P}^{m+r} = \mathcal{F}(P^m_{m+r})_e$, the Fourier transform of the even extension of the sequence $P^m_{m+r}$. From the well-known dual convolution result for functions in $l^2(\mathbb{Z}_N)$ [42],

$$\mathcal{F}(f \cdot P^m_{m+r})_e = \mathcal{D} \overset{4B}{\star} \mathcal{P}^{m+r} = Z^m_{m+r}, \tag{64}$$

where $\overset{N}{\star}$ denotes the cyclic convolution operator on sequences of length $N$.

In our proof of Theorem 7 in Section 5.2, we showed that in the NALT algorithm, the wraparound errors are localized to the high-frequency elements of the sequences $Z^m_{m+r}$ in such a way that they do not affect the accuracy of the computed Legendre coefficients. For the sequences $Z^m_{m+B/2}, Z^m_{m+B/4}, Z^m_{m+B/8} \cdots$ to be computed using bypass operations, the NALT algorithm requires that only a certain range of elements in each sequence be uncorrupted by wraparound errors. Specifically, for $Z^m_{m+B/2}$, only the sequence elements in the range $[-B/2+m, B/2-m-1]$ need to be uncorrupted. For $Z^m_{m+B/4}$, we need uncorrupted elements in the range $[-B/4, B/4]$, for $Z^m_{m+B/8}$ in the range $[-B/8, B/8]$, for $Z^m_{m+B/16}$ in the range $[-B/16, B/16]$, etc. In general, let the required range of uncorrupted sequence elements be $[-l, l]$. Because $\mathcal{P}^{m+r}$ is bandlimited to $[-(m+r), m+r]$, for $j \in [-l, l]$ we have that

$$\mathcal{D} \overset{4B}{\star} \mathcal{P}^{m+r}(j) = (\Pi_{[m+r+l]} \mathcal{D}) \overset{2(m+r+l)}{\star} (\Pi_{[m+r+l]} \mathcal{P}^{m+r})(j), \quad j \in [-l, l] \tag{65}$$

where $\Pi_{[k]}$ is the projection operator defined in the proof of Lemma 2. Essentially, Eq. (65) says that for $j \in [-l, l]$, the values $\mathcal{D} \overset{4B}{\star} \mathcal{P}^{m+r}(j)$ can be computed as the cyclic convolution of the appropriate sequences of length $2(m+r+l)$ instead of the cyclic convolution of sequences of length $4B$. Therefore, the $Z^m_{m+r}$ sequences specified by the logarithmic spacing can be computed by

$$\begin{aligned}
Z^m_{m+B/2} &= (\Pi_B \mathcal{D}) \overset{2B}{\star} (\Pi_B \mathcal{P}^{m+B/2}) \\
Z^m_{m+B/4} &= (\Pi_{B/2+m} \mathcal{D}) \overset{2(\frac{B}{2}+m)}{\star} (\Pi_{B/2+m} \mathcal{P}^{m+B/4}) \\
Z^m_{m+B/8} &= (\Pi_{B/4+m} \mathcal{D}) \overset{2(\frac{B}{4}+m)}{\star} (\Pi_{B/4+m} \mathcal{P}^{m+B/8}) \\
&\vdots
\end{aligned}$$

Using fast Fourier transforms, the cyclic convolution of two sequences of length $N$ can be computed in $O(N \log N)$ [6, 42]. $\mathcal{D}$ is obtained in $O(N \log N)$, and the values of the various $\mathcal{P}^{m+r}$ are constants and can be precomputed and stored. Then for the complexity of the bypass operations at logarithmic spacing we have

$$O(2B \log 2B + (B+2m) \log (B+2m) + (B/2+2m) \log (B/2+2m) + (B/4+2m) \log (B/4+2m) \ldots \tag{66}$$

which is $O(B \log B) + O(m \log^2 B)$, as stated in the theorem. Since this bound is less than the complexity of the NALT algorithm, the NALT complexity remains unchanged.

For the polylogarithmic ($\log^k$) case, we will first prove the stated bound for $m = 0$, and then the extension to $m > 0$ will be clear. Here we will adopt the notation $Z_r^0 = Z_r$. When $k = 1$, we have already shown that the cost of computing the sequences $Z_{B/2}, Z_{B/4}, \ldots$ using bypass operations is $O(B \log B)$. When $k = 2$, in addition to computing the sequences for $k = 1$, we also need to compute $Z_{B/2+B/4}, Z_{B/2+B/8} \ldots, Z_{B/4+B/8}, Z_{B/4+B/16} \ldots$, i.e., for every sequence $Z_r$ computed when $k = 1$, the $k = 2$ case additionally computes $Z_{r+r/2}, Z_{r+r/4} \ldots$ using bypass operations. To determine all of the sequences for the arbitrary $k$ case, one can construct a tree graph of height $k$ where each node in the graph is associated with a unique sequence $Z_r$, and where the children of the node associated with $Z_r$ are themselves associated with the sequences $Z_{r+r/2}, Z_{r+r/4} \ldots$. By assigning each node a *label*, or value, in a certain way, we can derive the stated complexity bound from the structure of the tree.

Specifically, let $T = (V, E)$ be a tree graph, where $V$ denotes the set of nodes and $E$ the set of edges. Except for the root node, each node in the tree will be associated with a unique sequence $Z_r$. Each node $V_r$ has a label $label[r]$ which will be assigned to one of the values in the set $labels = \{B/2, B/4, \ldots 4, 2, 0\}$. During the construction of the tree, the label value assigned to any node $V_r$, with the exception of the root node, will be the number of sequence elements in $Z_r$ which need to be uncorrupted by wraparound effects. To construct $T$, we begin with a root node $V_0$ with $label[0] = 0$. Level 1 of the tree is created by adding to the root the $(\log B) - 1$ child nodes $V_{\frac{B}{2}}, V_{\frac{B}{4}}, \ldots V_4, V_2$, and assigning their labels the values $B/2, B/4, \ldots 4, 2$, respectively. To create level 2 of the tree, for each node $V_r$ in level 1, add $(\log label[r]) - 1$ child nodes $V_{r+r/2}, V_{r+r/4}, \ldots V_{r+4}, V_{r+2}$, and assign their labels the values $label[r]/2, label[r]/4, \ldots 4, 2$, respectively. For example, we add to the node $V_{\frac{B}{4}}$ the child nodes $V_{\frac{B}{4}+\frac{B}{8}}, V_{\frac{B}{4}+\frac{B}{16}}, \ldots V_{\frac{B}{4}+4}, V_{\frac{B}{4}+2}$, and assign the values $B/8, B/16, \ldots 2$, respectively, to their labels. We continue to add levels to the tree in this manner until there are $k$ levels.

Each node $V_r$ is associated with the sequence $Z_r$, and we note that the index $r$ of any node $V_r$ is unique and can be computed by traversing the path from the root to the node $V_r$ and adding the label values of each node on the path, including $label[r]$. We also note that since $Z_r(k) = \langle fP_r, C_k \rangle$, the index $r$ corresponds to the bandlimit of the sequence $P_r$, which, from our discussion of the logarithmic ($k = 1$) case, is one of the two parameters that determine the complexity of computing $Z_r$ using a bypass operator. The other parameter is the number of sequence elements in $Z_r$ which need to be uncorrupted by wraparound effects; by construction, this value is $label[r]$. Thus, one can determine the cost of computing any sequence $Z_r$ by traversing the path from the root to the node $V_r$, summing label values along the way including the value $label[r]$, and then adding $label[r]$ again, for a total of $r + label[r]$. By the same arguments used in the k=1 spacing case, the cost to compute $Z_r$ is $O((r + label[r]) \log (r + label[r]))$. Computing the cost for every node in $T$ gives the complexity of computing a ($\log^k$) spacing of bypass operations.

In order to determine the cost for computing the entire tree $T$, we begin by considering the cost of computing only the leaves of the tree. As we traverse the path from the root to some arbitrary leaf $V_r$, the label value of each node on the path is at most half the value of the label of its parent (with the exception of the nodes on level 1 whose parent is the root node $V_0$). Using this observation, the cost of computing the leaves of the tree is on the order of

$$\sum_{i_1=1}^{lg\ B-1} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} \sum_{i_k=1}^{i_{k-1}-1} (2^{i_1} + 2^{i_2} + \ldots + 2^{i_{k-1}} + 2^{i_k} + 2^{i_k}) \log (2^{i_1} + 2^{i_2} + \ldots + 2^{i_{k-1}} + 2^{i_k} + 2^{i_k}) \quad (67)$$

operations. To bound this summation, note that $(2^{i_1} + 2^{i_2} + \ldots + 2^{i_{k-1}} + 2^{i_k} + 2^{i_k}) \leq 2^{i_1+1}$ since

$i_1 > i_2 > \ldots > i_k$. Then Eq. (67) is upper bounded by

$$\sum_{i_1=1}^{lg\ B-1} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} \sum_{i_k=1}^{i_{k-1}-1} 2^{i_1+1}\ \log\ 2^{i_1+1}. \tag{68}$$

When $i_1 = \log\ B - 1$, the $(k-1)$ innermost summations are bounded by

$$\sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} \sum_{i_k=1}^{i_{k-1}-1} B\ \log\ B = O(B\ \log^k\ B). \tag{69}$$

When $i_1 = \log\ B - 2$,

$$\sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} \sum_{i_k=1}^{i_{k-1}-1} \frac{B}{2}\ \log\ \frac{B}{2} = O(B/2\ \log^k\ B/2), \tag{70}$$

where the hidden constant is smaller than for the case where $i_1 = \log\ B - 1$. Continuing in this manner gives

$$O(B\ \log^k\ B + \frac{B}{2}\ \log^k\ \frac{B}{2} + \frac{B}{4}\ \log^k\ \frac{B}{4} + \ldots) = O(B\ \log^k\ B), \tag{71}$$

for the complexity of computing the leaves.

Now we can easily determine the cost of computing the entire tree. Consider the cost of computing the parents of the leaves:

$$\sum_{i_1=1}^{lg\ B-1} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} (2^{i_1} + 2^{i_2} + \ldots + 2^{i_{k-1}} + 2^{i_{k-1}})\ \log\ (2^{i_1} + 2^{i_2} + \ldots + 2^{i_{k-1}} + 2^{i_{k-1}}). \tag{72}$$

By a similar argument as for the leaves, this cost is bounded by $O(B\ \log^{k-1}\ B)$, and the cost of computing the grandparents of the leaves is $O(B\ \log^{k-2}\ B)$. As we continue to move up the tree from the leaves, we obtain

$$O(B\ \log^k\ B + B\ \log^{k-1}\ B + B\ \log^{k-2}\ B \ldots) = O(B\ \log^k\ B) \tag{73}$$

as the cost of computing the entire tree.

When $m \neq 0$, the tree is constructed as before, but after construction the label of the root node $V_0$ is assigned the value $m$. This increases the sum of the values of the labels on the path from the root to a node $V_r$ by $m$, which corresponds exactly to the increase in the bandwidth of $P^m_{m+r}$ over $P^0_r$, and gives the complexity $O(r + m + label[r])\ \log\ (r + m + label[r])$ for computing $Z^m_{m+r}$. This modification to Eq. (67), the cost of computing the leaves, is bounded by

$$\sum_{i_1=1}^{lg\ B-1} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_{k-1}=1}^{i_{k-2}-1} \sum_{i_k=1}^{i_{k-1}-1} (2^{i_1+1} + m)\ \log\ (2^{i_1+1} + m). \tag{74}$$

Using the same arguments as for the $m = 0$ case, the cost of the leaves is bounded by $O(B\ \log^k\ B) + O(m\ \log^{k+1}\ B)$, and the cost of the entire tree is also $O(B\ \log^k\ B) + O(m\ \log^{k+1}\ B)$, as stated in the theorem.

**Odd $m$ case:**

We have already stated that for $m$ odd, the cosine transform of $P_{m+r}^m$ is not bandlimited. Nevertheless, the Driscoll-Healy sampling theorem [20] allows us to compute the sequences $Z_{m+r}^m = \langle f P_{m+r}^m, C_k \rangle$ using only $2B$ samples each of $f$ and $P_{m+r}^m$, but this method requires $O(B \log^{k+1} B)$ time for $\log^k$ spacing, which is essentially a factor of $(\log B)$ greater than the stated result. In order to improve the efficiency, let us first reconsider Eq. (62) for odd $m$:

$$P_m^m(\cos\theta) \propto \sin(\theta)\sin^{m-1}(\theta) = \sin(\theta)(1-\cos^2(\theta))^{\frac{m-1}{2}}. \tag{75}$$

As we noted earlier, because the sequence $\sin\theta$ is not bandlimited, then neither is $P_m^m$. Now, let us denote by $\mathcal{R}^m(r,\cdot))$ the operator which applies the three-term recurrence rule $r$ times to a sequence. Hence, we can generate $P_{m+r}^m$ by $P_{m+r}^m = \mathcal{R}^m(r, P_m^m)$. Since the three-term recurrence is a linear recurrence, we have that $\mathcal{R}^m(r, h\cdot g) = h \cdot \mathcal{R}^m(r,g)$ for two sequences $h$ and $g$; therefore, we can compute the product $f \cdot P_{m+r}^m$ by $\mathcal{R}^m(r, f\cdot P_m^m) = f \cdot \mathcal{R}^m(r, P_m^m)$. Extending this idea, we have

$$f \cdot P_{m+r}^m = (f \cdot \sin\theta)\mathcal{R}^m(r, P_m^m/\sin\theta). \tag{76}$$

The cosine transform of the sequence

$$P_m^m/\sin\theta = (1-cos^2\theta)^{\frac{m-1}{2}} = \sin^{m-1}\theta \tag{77}$$

has all coefficients of order $m$ and greater equal to zero; therefore, the cosine transform of the sequence $G_{m+r}^m = \mathcal{R}^m(r, P_m^m/\sin\theta)$ has all coefficients of order $m+r$ and greater equal to zero. Intuitively, the sequence $G_{m+r}^m$ is generated by the same recurrence used to compute $P_{m+r}^m$, but instead of initializing the recurrence with $P_m^m$, the sequence $(P_m^m/\sin\theta)$ is used instead.

Using these new bandlimited sequences $G_{m+r}^m$, we have the identity

$$Z_{m+r}^m(k) = \langle f, P_{m+r}^m C_k \rangle = \langle f\cdot\sin\theta, G_{m+r}^m C_k \rangle. \tag{78}$$

Since $G_{m+r}^m$ is bandlimited, we can now make the same arguments as in the case where $m$ is even for improving the efficiency of a $(\log^k)$ spacing of bypass operations, where we replace $\mathcal{D}$ with $\mathcal{D}' = \mathcal{F}(f\cdot\sin\theta)_e$ and $\mathcal{P}^{m+r}$ with $\mathcal{G}^{m+r} = \mathcal{F}(G_{m+r}^m)_e$. This results in the stated complexity bound. ■

# 6 Further directions

## 6.1 Efficient Variations of the Driscoll-Healy Algorithm

We have shown through experimentation that our implementation of the Driscoll-Healy algorithm is faster than a naive implementation of the Legendre transform for reasonable problem sizes. The naive implementation projects the weighted $2B$ data values onto the function $P_l^m$ to compute the coefficient $\hat{f}(l,m)$:

$$\hat{f}(l,m) = \sum_{k=0}^{2B-1} f(\cos\frac{(2k+1)\pi}{4B}) P_l^m(\cos\frac{(2k+1)\pi}{4B}) = \langle f, P_l^m \rangle. \tag{79}$$

For a fixed value of $m$, the discrete summation method of Eq. (79) uses $2B(B-m)$ arithmetic operations to compute all of the coefficients $\hat{f}(l,m)$ for $l = m, m+1, \ldots, B-1$. If, however, we consider the dual of the computation for Eq. (79) in the transform domain, a large constant reduction in the arithmetic complexity is realized, making the dual naive algorithm, or "semi-naive" algorithm, more attractive in terms of runtime efficiency for some of the smaller problem sizes. The concepts

underlying the semi-naive algorithm can also be used to improve the runtime efficiency of the Driscoll-Healy algorithm, and in some cases a hybrid algorithm using both the semi-naive and variants of the Driscoll-Healy algorithm gives the best performance. In the following discussion, we present the semi-naive algorithm as well as some modified fast algorithms and provide some guidelines to assist in the implementation process.

Before proceeding, we cannot overstress the fact that which of these algorithms will perform best for a given computing environment depends heavily on implementation constraints such as machine architecture, memory and storage availability, expected problem sizes, programming resources, etc. All of these constraints, as well as constant factors, are not represented in the asymptotic complexity measurement of these algorithms. One measure which is independent of the environment is the total number of multiplications required for a given algorithm. We have chosen this measure to compare the various algorithms, and give exact formulas.

From the analysis of Section 5.2, we have that

$$Z_l^m(k) = \langle f, P_l^m C_k \rangle = \sum_{j=-B}^{B-1} \hat{f}_e(j) \hat{P}_{l_e}^m(j-k) \tag{80}$$

where $\hat{f}_e$ and $\hat{P}_{l_e}^m$ are the Fourier transforms of the even extensions of $f$ and $P_l^m$. Since $\hat{f}(l,m) = Z_l^m(0)$, setting $k = 0$ in Eq. (80) gives an alternate method for computing $\hat{f}(l,m)$. Assuming that $\hat{f}_e$ and $\hat{P}_{l_e}^m$ have already been computed, then it appears that Eqs. (79) and (80) both require $2B$ multiplications and $2B-1$ additions. However, since $\hat{P}_{l_e}^m$ is bandlimited to $[-l,l]$ for even values of $m^2$, and since only even (odd) coefficients are non-zero when $l$ is even (odd), a reduction in the number of arithmetic operations is effected. Also, since $\hat{f}_e(j) = \hat{f}_e(-j)$ and $\hat{P}_{l_e}^m(j) = \hat{P}_{l_e}^m(-j)$, symmetry can be exploited to reduce further the number of arithmetic operations by a factor of 2. Thus, by using Eq. (80) instead of Eq. (79), and assuming that the Fourier transforms of the functions $P_{l_e}^m$ are precomputed and stored, a semi-naive algorithm is realized which is of the same order of complexity as the naive algorithm, but which has a much smaller constant. Specifically, for the $m = 0$ case, the semi-naive algorithm computes the coefficients $\hat{f}(0,0), \hat{f}(1,0), \ldots, \hat{f}(B-1,0)$ using only $(\frac{B^2}{4} - \frac{B}{2} + B \log B)$ multiplications and $(3B \log B + 2B - B + 1)$ additions, where we have used the fast cosine transform of [50] to compute $\hat{f}_e$. This represents a reduction by a factor of approximately 8 compared to the time-domain version of the naive algorithm.

From our previous timing diagrams (see Figures 13 and 14), it is apparent that this semi-naive algorithm may be faster than the Driscoll-Healy algorithm if this factor 8 speedup is introduced. Brief experiments verified this for our implementation of the Legendre polynomial ($m = 0$) case. A straightforward implementation of the Driscoll-Healy algorithm as described in [20] requires ($3B \log^2 B - 7B \log B + 7B - 1$) multiplications; this exceeds the number of multiplications in the semi-naive algorithm whenever $B < 1024$. At $B = 1024$, the Driscoll-Healy algorithm requires about 10% fewer multiplications, but depending on the machine environment, the semi-naive algorithm may still be faster - we have found this to be the case on DEC Alpha and DEC 5000 workstations. However, some simple variations on the Driscoll-Healy algorithm can give large constant reductions in the complexity, moving the theoretical breakpoint back to smaller values of bandwidth $B$. In the following descriptions of these algorithms, for simplicity we assume $m = 0$; the extensions to arbitrary $m$ are straightforward. We will also simplify the notation by referring to the Driscoll-Healy algorithm as the DH algorithm.

[2] A slight modification produces this same result for odd $m$ - see the proof of the Bypass Theorem

### DH-Mid Algorithm

One variation on the DH algorithm reduces the constant factor by approximately half. Instead of initially computing the sequence $Z_0$ of length $2B$, compute instead the sequences $Z_{B/2}$, $Z_{B/2+1}$, each of length $B$. Then $Z_{3B/4}$ and $Z_{3B/4+1}$ are computed as before, but by using the three-term recurrence "in reverse", i.e.

$$P_{l-1}(x) = \frac{2l+1}{l} x P_l(x) - \frac{l+1}{l} P_{l+1}(x), \tag{81}$$

the sequences $Z_{B/4}, Z_{B/4+1}$ can be computed by applying a reverse shifted Legendre mask to $Z_{B/2}, Z_{B/2+1}$ instead of $Z_0, Z_1$ as in the DH algorithm. A savings is realized because instead of applying a Fourier transform operator to each of $Z_0, Z_1, Z_{B/2}$ and $Z_{B/2+1}$ in order to effect fast convolution with shifted Legendre masks, only $Z_{B/2}$ and $Z_{B/2+1}$ need to be transformed. At the next level, only $Z_{B/4}, Z_{B/4+1}, Z_{3B/4}$, and $Z_{3B/4+1}$ need to be transformed to compute $Z_j$, $j = B/8, B/8 + 1, 3B/8, 3B/8 + 1, 5B/8, 5B/8 + 1, 7B/8, 7B/8 + 1$, whereas in the DH algorithm we would also need to tranform $Z_0, Z_1, Z_{B/2}$ and $Z_{B/2+1}$. This trend continues as the algorithm recurses down to the base case. The number of multiplications required for this algorithm, which we will label DH-Mid, is

$$\frac{3B}{4} \log^2 B + \frac{17B}{4} \log B - \frac{38B}{4} + 2. \tag{82}$$

### DHK-Mid Algorithm

Another variation on DH-Mid is to recurse down to a level where the most recently computed $Z_l$ sequences are of some length $2k$, and then switch over to a semi-naive type method to compute the remaining Legendre coefficients. For example, suppose the recursion halts after we have computed a group of length $2k$ sequences which includes $Z_r, Z_{r+1}$ and $Z_{r+k}, Z_{r+k+1}$. By using a semi-naive method and the Fourier transforms of shifted Legendre and reverse shifted Legendre functions, $Z_r, Z_{r+1}$ can be used to compute the Legendre coefficients $Z_j(0)$, $j = r-k/2+1, r-k/2+2, \ldots, r-1, r+2, \ldots, r+k/2-1$. Similarly, $Z_{r+k}, Z_{r+k+1}$ can be used to compute $Z_j(0)$, $j = r + k/2, r + k/2 + 1, \ldots, r + k - 1, r + k + 2, \ldots, r + k + k/2 - 1$. This algorithm, which we label DHK-Mid, uses

$$\frac{3B}{4} \log^2 B - \frac{3B}{4} \log^2 k + \frac{17B}{4} \log B - \frac{15B}{4} \log k - 5B + 2 + \frac{Bk}{4} + \frac{k^2}{2} + \frac{3B}{2} - \frac{4B}{k} - 2k + 2 \tag{83}$$

multiplications. The trivial case is when $k = 2$, at which point DHK-Mid is just the DH-Mid algorithm. The value of $k = 2^j$ which minimizes Eq. (83) for various bandwidths $B$ is tabulated in Figure 15.

| *Bandwidth B* | *k* | *Multiplications* |
|---|---|---|
| 32 | 16 | 524 |
| 64 | 32 | 1692 |
| 128 | 32 | 4724 |
| 256 | 32 | 12964 |
| 512 | 64 | 33636 |
| 1K | 64 | 84292 |
| 2K | 64 | 207620 |

Figure 15: Tabulation of values for $k$ which minimize the number of multiplications in the DHK-Mid algorithm for a range of bandwidths B.

### Hybrid Algorithm

Finally, at some problem sizes a hybrid algorithm may be the fastest method. In this approach, the semi-naive algorithm is used to compute Legendre coefficients from $Z_0(0)$ to $Z_{r-1}(0)$, and then the DHK-Mid algorithm is used to compute the remaining Legendre coefficients $Z_r(0)$ to $Z_{B-1}(0)$. A natural choice for $r$ is $B/2$, since the DHK-Mid portion of the algorithm operates on sequences which have lengths equal to powers of 2.

Figure 16 graphs the $\log_2$ of the number of multiplications for each of these algorithms relative to the number of multiplications for the seminaive algorithm. From Figure 16, one can see that in order to achieve minimum runtime, it may be necessary to use different algorithms at different problem sizes. We emphasize again that Figure 16 is only a guideline since the actual runtime performance depends on environmental constraints, particularly machine architecture. In our implementation on a DEC Alpha workstation, we found that the semi-naive algorithm was the best choice for bandwidths from 32 to 128, and that the hybrid algorithm was best for bandwidths of 256, 512, and 1024. As $m$ increases from zero, one would expect the semi-naive performance to degrade relative to the various derivatives of the DH algorithm; on the other hand, as $m$ increases, the need to use stability bypass operations may affect the performance of the DH type algorithms.

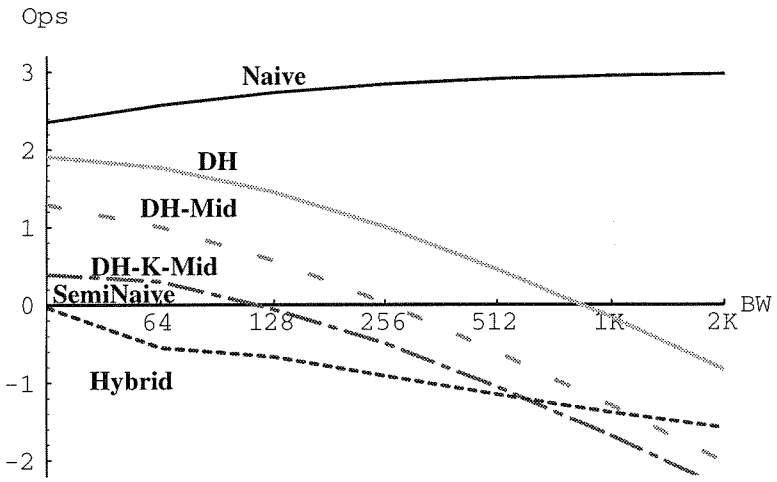



Figure 16: Comparison of the $\log_2$ of the number of multiplications for a variety of algorithms relative to the semi-naive algorithm. The hybrid algorithm has been restricted to using a semi-naive method to compute Legendre coefficients from $Z_0(0)$ to $Z_{B/2-1}(0)$.

## 6.2 Parallelizability

As remarked in Section 3.5, the divide and conquer nature of the fast forward transform would appear to make it a ready candidate for parallelization. In this subsection we expand on this observation and show that the direct implementation of the fast NALT algorithm is indeed well-adapted for parallel architectures, in the sense that it may be efficiently mapped to a hypercube network. Adaptations of the following discussion should show that the same is true of the the modified algorithm of Section 5.2, but we defer this more involved discussion to future work.

The main result of this section is the following Theorem.

**Theorem 9** *The fast normalized associated Legendre transform of a sequence* $\mathbf{s}$ *of length* $2B = 2^r$ *can be computed on an* $(r+1)$*-dimensional hypercube in* $O(r^2)$ *steps using a leveled algorithm. Consequently, there exists a work-optimal parallel algorithm for computing the fast normalized associated Legendre transform.*

Loosely speaking, Theorem 9 indicates that the fast NALT algorithm admits an efficient parallel implementation on a hypercube network.

The proof of Theorem 9 requires some definitions and descriptions which we now detail.

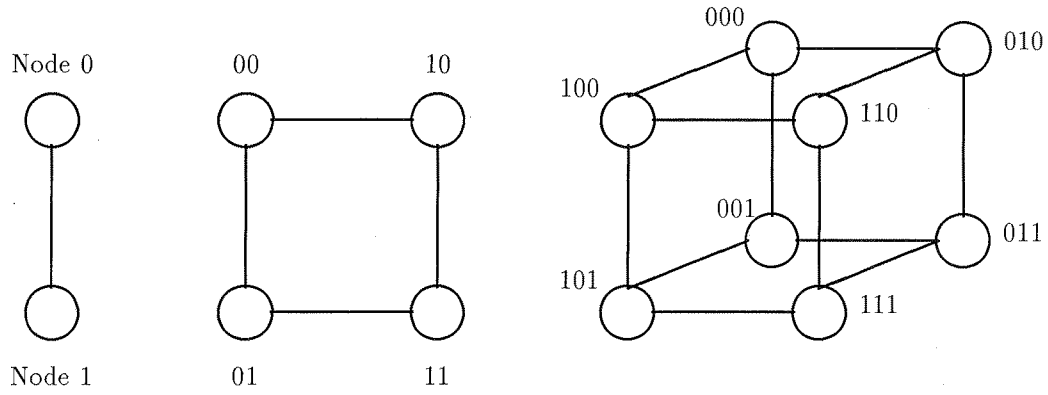
### 6.2.1 Hypercube networks

The hypercube network is powerful, general purpose network for designing and implementing parallel algorithms. A large number of popular networks, such as arrays, binary trees, and networks derived from the hypercube can be efficiently simulated with only a constant factor slowdown. Furthermore, *any* $N$-node bounded-degree network can be simulated on a hypercube with at most an $O(\log\ N)$ factor slowdown [35]. Conversely, hypercube networks can be simulated efficiently on other networks [48]. Hypercube networks are recursively structured and have small diameter $(\log\ N)$, allowing for efficiency in routing. Thus, despite some drawbacks such as logarithmic node degree, it remains a popular choice as a design model. To set up notation, following [48], we now describe the hypercube network.

The $r$-dimensional hypercube has $2^r$ nodes, where each node is labeled by a unique $r$-bit binary string. The node labeled $d_r d_{r-1} \ldots d_1$ is connected by an edge to every node that is indexed with a string that differs in exactly one bit position. Edges which connect nodes differing in bit position $i$ are said to be in *dimension* $i$ of the hypercube. We refer to such edges as the set of *$i$-dimensional* edges. Examples of the 1-, 2-, and 3-dimensional hypercubes are shown in Figure 17.

The nodes of a hypercube are naturally identified with processors, and edges with the communication links between them – we will switch between these identities often during the discussion and without notification since the context and meaning should be clear. We note also that the $r$-dimensional hypercube can be recursively defined by connecting an edge between the $i$-th node of one $(r-1)$-dimensional hypercube and the $i$-th node of another $(r-1)$-dimensional hypercube [35]. We will use this recursive structure when designing algorithms.

Highly structured algorithms are usually candidates for parallel implementations. A large class of such hypercube algorithms is known as the class of **leveled** algorithms, defined as those algorithms which use only one dimension of hypercube edges at a time, but in arbitrary order. A subset of the leveled algorithms is the set of **normal** algorithms, which require that adjacent dimensions are used at consecutive routing steps, i.e., if dimension $i$ is the most recently used edge dimension, then the next edge dimension must be $(i \pm 1) \bmod r$. **Fully normal** algorithms require that all $r$ dimensions of the edges are used in consecutive order. A leveled algorithm can be executed on a hypercube without any routing delay, which simplifies implementation.




(a) r = 1 (b) r = 2 (c) r = 3

Figure 17: Hypercubes of various dimensions. Note the recursive structure, i.e., a hypercube of dimension $j$ is formed from two hypercubes of dimension $j-1$.

### 6.2.2 A Parallel Fast Legendre Transform Algorithm

To prove Theorem 9 consider again the algorithm described in Section 3.3. The fast forward transform is essentially a succession of convolutions followed by projections. The former are computed using FFT's (already known to be well-adapted to hypercube networks ([35, 53]) and the latter are essentially data routings. Thus, proof of Theorem 9 amounts to a careful organization of the computation on hypercube.

Throughout this discussion we assume that the dimensions of the structures we are working with are integer powers of 2. In Section 3.3, the fast NALT algorithm is represented as a series of matrix-vector multiplications. We identify a vector $\mathbf{s} = (s_0, \ldots s_{N-1})^T$ with the finite sequence $\mathbf{s} = s(0), \ldots s(N-1)$. In an $r$-dimensional hypercube algorithm for matrix-vector multiplications, we assume that the sequence $\mathbf{s}$ is stored such that element $s(j)$ is initially located at the node labeled with the binary representation of $j$, i.e., $j = d_r^j, d_{r-1}^j \ldots d_1^j$. The complement of a binary digit $d_r^j$ is denoted $\overline{d}_r^j$, and the bitwise complement of a hypercube node label $j = d_r^j, d_{r-1}^j \ldots d_1^j$ is denoted $\overline{j} = \overline{d}_r^j, \overline{d}_{r-1}^j \ldots \overline{d}_1^j$. Many of the subproblems which make up the parallel Legendre transform algorithm compute some permutation $\mathbf{s}_P$ of the sequence $\mathbf{s}$; we consider such a permutation operation to be completed when under the mapping $s(j) \xrightarrow{P} s_P(k)$, the element $s(j)$ initially located at the **source** node labeled $j$ is routed to the **destination** node labeled $k$. Projection operations, which extract some subsequence $\mathbf{s}_m$ of length $m$ from a sequence $\mathbf{s}_n$ of length $n$, will be considered complete when the the elements of $\mathbf{s}_m$ are routed to the nodes labeled $0 \ldots m-1$. A sequence element $s(j)$ is considered **lost** if it has not yet reached its destination node. We assume that each edge is a two-way communication channel; i.e., a value sent out along the $i$-dimensional edge won't collide with an incoming value on this same edge. Finally, we assume that any constants which are used in binary arithmetic operations as part of a hypercube algorithm are stored or generated locally at the proper node, i.e., no routing steps are used to rout constant data.

As presented in Section 3.3, the fast NALT algorithm is built as a sequence of FFT's, projections and diagonal matrix multiplications of geometrically decreasing size. The geometric decrease will be reflected in the partitioning of the successive matrix-vector multiplications to hypercubes of decreasing dimension within the original network. Since the FFT is (by now) well-known to be well-adapted to parallel architectures, we are essentially left with showing that at each step the current data vector

can be routed efficiently to the appropriate configuration on the hypercube. To begin, we show that two fundamental data movements may be performed using fully normal algorithms.

**Lemma 3** *Define the reversal of a sequence* $\mathbf{s}$ *of length* $N$ *to be the sequence* $\mathbf{s}_{rev}$*, where*

$$s(j) \xrightarrow{rev} s_{rev}(N-1-j)$$

*. Then a sequence* $\mathbf{s}$ *of length* $N = 2^r$ *can be reversed using a fully normal algorithm on an* $r$*-dimensional hypercube in* $O(r)$ *steps.*

**Proof:** Under a sequence reversal, $s(j)$ gets mapped to $s_{rev}(N-1-j)$. Since the binary representation of $N-1$ is the $r$-digit string $11\ldots1$, then the label $N-1-j$ of the destination node is the bitwise complement $\overline{j}$ of the source node $j$. For the hypercube algorithm to be fully normal, the element $s(j)$ at the source node $d_r^j, d_{r-1}^j \ldots d_1^j$ needs to be routed to the destination node $\overline{d}_r^j, \overline{d}_{r-1}^j \ldots \overline{d}_1^j$. In the first step of the algorithm, rout every element at level 0 along the $r$-dimensional edge; this routs the sequence element at the node $d_r^j, d_{r-1}^j \ldots d_1^j$ to the node $\overline{d}_r^j d_{r-1}^j \ldots d_1^j$. On the next step rout every element along the $(r-1)$-dimensional edge, which maps the element at node $\overline{d}_r^j d_{r-1}^j \ldots d_1^j$ to the node $\overline{d}_r^j \overline{d}_{r-1}^j \ldots d_1^j$. Continue routing along consecutively decreasing edges. After $r$ routing steps, the element $s(j)$ originally located at $d_r^j, d_{r-1}^j \ldots d_1^j$ will be located at $\overline{d}_r^j, \overline{d}_{r-1}^j \ldots \overline{d}_1^j$, thus achieving the reversed sequence using a fully normal algorithm.

■

**Lemma 4** *Given a sequence* $\mathbf{s}$ *of length* $N = 2^r$*, the cyclic right shifted sequence* $\mathbf{s}_i$*, where*

$$s(j) \xrightarrow{+1 \bmod N} s_i(j+1 \bmod N)$$

*, can be computed using a fully normal algorithm in* $O(r)$ *steps on an* $r$*-dimensional hypercube.*

**Proof:** Assume element $s(j)$ is initially located at the source node labeled $j$. Initially, all elements $s(j)$ located at even-numbered nodes, i.e., nodes with labels $d_r^j d_{r-1}^j \ldots d_2^j 0$, can reach their destinations by routing along the 1-dimensional edge. By routing *all* elements along the 1-dimensional edge, then half of them reach their destination in only one step. After this step, every lost element is located at an even-numbered node. Lost elements located at nodes labeled such that the digit $d_2^j = 0$ can now reach their destination by routing along the 2-dimensional edge. By routing every lost element at an even-numbered node along this edge, half of the lost elements reach their destination. Now only the elements located at nodes with labels $j \equiv 0 \bmod 4$ are lost, and those with the digit $d_3^j = 0$ can reach their destinations by routing along the 3-dimensional edge. If we rout all lost elements at nodes with labels $j \equiv 0 \bmod 4$, and then rout all lost elements at nodes labeled $j \equiv 0 \bmod 8$, then those labeled $j \equiv 0 \bmod 16$, etc., then at every step half of the elements get routed to their destination. So after $r$ steps, every element has reached its destination. Since all routing dimensions were used in consecutive order, this algorithm is fully normal.

■

**Lemma 5** *A diagonal matrix can be applied to a vector on a hypercube using only one local multiplication at each node.*

As discussed in Section 3.3, the NALT uses a fast cosine transform as part of the implementation of the input operator $\mathcal{I}$. Recall that this is effected by computing the even extension of the input sequence and then applying an FFT.

**Lemma 6** *A sequence* s *of length* $N = 2^r$ *can be evenly extended (cf. 39) using a leveled algorithm on an* $(r+1)$*-dimensional hypercube in* $O(r)$ *steps.*

**Proof:** Assume element $s(j)$ is initially located at node $j$. For all $j$, rout a copy of $s(j)$ to node $s(N+j)$ using the $(r+1)$-dimensional edge. Reverse the sequence elements $s(N), s(N+1), \ldots, s(2N-1)$ using the algorithm in Lemma 3. Then do a cyclic right 1-shift on the elements at nodes labeled $N$ through $2N-1$, which by Lemma 4 takes $O(r)$ steps. Set the value at $s(N)$ to zero, and double the value at $s(0)$.

■

**Lemma 7** *Given a sequence* s *of length* $2N = 2^r$*, the sequence* $\mathbf{s}_N$ *obtained by performing a cyclic right* $N$*-shift on* s*, i.e.,* $s(j) \xrightarrow{N} s_N(j+N \bmod 2N)$*, can be computed on an* $r$*-dimensional hypercube using one* $r$*-dimensional routing step.*

**Proof:** Under a cyclic right $N$-shift, element $s(j)$ gets mapped to the node labeled $(j+N) \bmod 2N$. Since $s(j)$ is initially located at the source node labeled $j = d_r^j, d_{r-1}^j \ldots d_1^j$, then under this mapping its destination node is labeled $\overline{d}_r^j d_{r-1}^j \ldots d_1^j$. This can be accomplished with a single routing step along the $r$-dimensional edge.

■

**Lemma 8** *The (Inverse) Discrete Fourier transform of a sequence* s *of length* $N = 2^r$ *can be computed in* $O(r)$ *steps on an* $r$*-dimensional hypercube using a fully normal algorithm.*

**Proof:** See any one of [53, 35, 15] for either butterfly or hypercube algorithms which perform such an amazing feat. Section 3.7 of [35] states this result for the parallel FFT.

■

To continue, recall the definition of the Kronecker product of two matrices. From [53], if $A \in \mathbf{C}^{p \times q}$ and $B \in \mathbf{C}^{m \times n}$, then the *Kronecker* product $A \bigotimes B$ is the $p$-by-$q$ block matrix

$$A \bigotimes B = \begin{pmatrix} a_{00}B & \cdots & a_{0,q-1}B \\ \vdots & & \vdots \\ a_{p-1,0}B & \cdots & a_{p-1,q-1}B \end{pmatrix} \in \mathbf{C}^{pm \times qn}. \tag{84}$$

**Lemma 9** *Applying the matrix* $I_k \bigotimes \mathcal{F}_{N/k}$*, where* $N = 2^r$*,* $k = 2^t$*,* $I_k$ *is the* $k \times k$ *identity matrix, and* $\mathcal{F}_{N/k}$ *is the* $N/k \times N/k$ *DFT matrix of (11), to a sequence* s *of length* $N$ *can be accomplished with a normal algorithm in* $O(r-t)$ *steps on an* $r$*-dimensional hypercube.*

**Proof:** On an $r$-dimensional hypercube, an $N$-point FFT is computed by first computing two FFTs of size $N/2$ on the two $(r-1)$ dimensional hypercubes which recursively define the full $r$-dimensional hypercube, and combining the results as per [53, 35, 15]. By following the recursion down $t$ steps, one finds that $k$ FFTs, each of size $N/k$, are computed on $k$ hypercubes of dimension $log\ (N/k) = r-t$, which is exactly what is computed by applying the matrix $I_k \bigotimes \mathcal{F}_{N/k}$ to the input sequence s. Each FFT takes only $O(log\ (N/k)) = O(r-t)$ steps, and on the $r$-dimensional hypercube, each of these FFTs is computed in parallel with the others, thus effecting application of $I_k \bigotimes \mathcal{F}_{N/k}$ to s in only $O(r-t)$ steps.

■

**Lemma 10** *The* $\frac{2B}{2^j} \times \frac{2B}{2^{j-1}}$ *projection matrix* $\pi_j$ *of (10) can be applied to a sequence* **s** *of length* $\frac{2B}{2^{j-1}} = 2^r$ *on an* $r$*-dimensional hypercube using a normal algorithm consisting of only two routing steps.*

**Proof:** Let $N = \frac{2B}{2^{j-1}} = 2^r$. Then $\pi_j$ is the $\frac{N}{2} \times N$ projection matrix which routs the middle $\frac{N}{2}$ elements of **s**, i.e., $s(N/4) \ldots s(3N/4-1)$, to the first $N/2$ nodes of the hypercube. The source node labels of the elements $s(\frac{N}{2}), s(\frac{N}{2}+1) \ldots s(\frac{3N}{4}-1)$ differ from their destination node labels only in the digits $d_r$ and $d_{r-1}$. The source node labels of the elements $s(\frac{N}{4}) \ldots s(\frac{N}{2}-1)$ differ from the labels of their destination nodes only in the digit $d_{r-1}$. Thus, by first routing the elements $s(N/4) \ldots s(3N/4-1)$ along their $(r-1)$-dimensional edge, elements $s(N/4) \ldots s(N/2-1)$ reach their destination, and elements $s(N/2) \ldots s(3N/4-1)$ differ from their destinations only in the digit $d_r$. Routing these remaining lost elements on the $r$-dimensional edge routs them to their destinations.

■

**Lemma 11** *Each of the* $N \times N$ *tridiagonal matrices* $\begin{pmatrix} a_j(k) & b_j(k) \\ c_j(k) & d_j(k) \end{pmatrix}$ *of (11), where* $N = 2^r$*, can be applied to a sequence* **s** *using a normal algorithm on an* $r$*-dimensional hypercube in* $O(1)$ *steps.*

**Proof:** We will denote the entries of the tridiagonal matrix by $a_{ij}$. Note that for any column of the matrix, the row indices of non-zero entries differ by $N/2$. We assume that sequence element $s(j)$ is located at node $j$. Our goal is to store the value $(a_{jj} \cdot s(j)) + (a_{j,j+N/2 \bmod N} \cdot s(j + N/2 \bmod N))$ at node $j$. To do this, first prestore the matrix elements $a_{jj}$ and $a_j, j + \frac{N}{2} \bmod N$ at the hypercube node labeled $j$. Rout a copy of $s(j)$ along the $r$-dimensional edge, then read in the incoming value on this edge, which must be $s(j + N/2 \bmod N)$. We now have all the data we need at each node, so compute $(a_{jj} \cdot s(j)) + (a_{j,j+N/2 \bmod N} \cdot s(j + N/2 \bmod N))$ to complete the operation.

■

**Lemma 12** *The matrix* $\mathcal{I}$ *of (8) can be applied to a sequence* **s** *of length* $2B = 2^r$ *on an* $(r+1)$*-dimensional hypercube using a leveled algorithm in* $O(r)$ *steps.*

**Proof:** Application of $\mathcal{I}$ to **s** computes the sequences $Z_m^m$ and $Z_{m+1}^m$. On the hypercube we will compute them consecutively, and place the elements of $Z_m^m$ in the first $2B$ nodes, and place $Z_{m+1}^m$ in the second $2B$ nodes. From [38], the sequence $Z_m^m$ can be obtained by $Z_m^m = \pi_B \cdot \mathcal{F}_c \cdot E \cdot P_m^m \cdot \mathbf{s}$, where $E$ is the $4B \times 2B$ operator of (44) which computes the even extension of a sequence of length $2B$, and $\mathcal{F}_c$ is a $4B \times 4B$ DFT matrix which differs from the usual DFT matrix in that the output sequence of Fourier coefficients is ordered by $\hat{s}(-2B), \hat{s}(-2B+1), \ldots \hat{s}(2B-1)$ instead of the usual $\hat{s}(0), \ldots \hat{s}(2B-1), \hat{s}(-2B), \hat{s}(-2B+1) \ldots \hat{s}(-1)$. By Lemma 5, the matrix-vector product $P_m^m \cdot \mathbf{s}$ can be accomplished in one local step. By Lemma 6, the operator $E$ can be applied using a leveled algorithm in $O(r)$ steps. By Lemmas 7 and 8, $\mathcal{F}_c$ can be applied using normal algorithms in $O(r)$ steps by first computing the FFT and then performing a cyclic right $N$-shift. By Lemma 10, $\pi_j$ is effected using a normal algorithm consisting of two steps, giving a total of $O(r)$ steps to compute $Z_m^m$ using a series of leveled and normal algorithms, which clearly constitutes a leveled algorithm. By a similar argument, $Z_{m+1}^m$ can be computed in $O(r)$ steps. However, an additional routing step is necessary since the elements of $Z_{m+1}^m$ are located in the first $2B$ nodes instead of the second $2B$ nodes. A single routing step along the $r$-dimensional edge performs the appropriate mapping while maintaining the leveled property.

■

**Lemma 13** *The $4B \times 4B$ matrix $\mathcal{M}_j$, where $4B = 2^r$, can be applied to a vector in $O(r-j)$ steps using a leveled algorithm on an $r$-dimensional hypercube.*

**Proof:** Partition the hypercube into $2^j$ hypercubes, each of dimension $(r-j)$. We only need to consider the application of a single block $\begin{pmatrix} \Pi_j \\ \Pi_j \cdot \mathcal{T}_j(k) \end{pmatrix}$ of $\mathcal{M}_j$, since we claim that application of this block can be carried out on a single hypercube of dimension $r-j$, which corresponds with the dimensions of each $2^{r-j} \times 2^{r-j}$ block. Since there are $2^j$ blocks, and $2^j$ hypercubes, or subcubes, the entire matrix $\mathcal{M}_j$ can be applied using an $r$-dimensional hypercube. First we will compute the application of the upper half of the block $\Pi_j$, followed by the application of the lower half $\Pi_j \cdot \mathcal{T}_j(k)$. By Lemma 10, the upper half operator $\Pi_j$ is effected by a normal algorithm in only 2 single-dimensional routing steps on edges of dimension $(r-j)$ and $(r-j-1)$, which are within the bounds of the subcube. To compute the lower half of the block, first consider the factorization (11)

$$\mathcal{T}_j(k) = \begin{pmatrix} \mathcal{F}^{-1} & \bigcirc \\ \bigcirc & \mathcal{F}^{-1} \end{pmatrix} \begin{pmatrix} a_j(k) & b_j(k) \\ c_j(k) & d_j(k) \end{pmatrix} \begin{pmatrix} \mathcal{F} & \bigcirc \\ \bigcirc & \mathcal{F} \end{pmatrix}. \tag{85}$$

By Lemma 7 and Lemma 8, the $\mathcal{F}$ operators can be applied in $O(r-j)$ steps using a normal algorithm on the hypercube with routing dimensions of $r-j$ or less. By Lemma 11, the tridiagonal matrix $\begin{pmatrix} a_j(k) & b_j(k) \\ c_j(k) & d_j(k) \end{pmatrix}$ can be applied in $O(1)$ steps using only one single dimensional routing step on the $(r-j)$-dimensional edge. Again by Lemmas 7 and 8, the $\mathcal{F}^{-1}$ operators can be applied in $O(r-j)$ steps using edge dimensions of $(r-j)$ or less, giving a total of $O(r-j)$ steps for application of $\mathcal{T}_j$. Application of $\Pi_j$ takes two routing steps, and an additional routing step of dimension $(r-j-1)$ on the output of $\Pi_j$ is necessary to map these elements to their appropriate locations, which is accomplished in one single-dimensional routing step. Thus, each block is applied in $O(r-j)$ steps using a leveled algorithm on each subcube, and each block can be computed in parallel, effecting the application of $\mathcal{M}_j$ in only $O(r-j)$ steps.

■

The last lemma concerns the operator $\mathcal{O}$. This a $B \times 4B$ matrix used for extracting the Legendre coefficients and normalizing if necessary. Our concern is with the extraction, or projection, operation only, because normalization involves multiplication by a constant. After application of the last $\mathcal{M}_i$ matrix, the output sequence, which we will label $Z_{out}$, is a sequence of length $4B$ consisting of the $B$ sequences $Z_l^m$, each of which has been reduced to length 4 by the projection operators $\pi_i$. It turns out that the third element of each $Z_l^m$ sequence is the Legendre coefficient we need. Equivalently, the elements of $Z_{out}$ with index $j \equiv 2 \bmod 4$ are the Legendre coefficients. We will consider the operation $\mathcal{O}$ effected when these Legendre coefficients have been routed to the first $B$ nodes of the hypercube.

**Lemma 14** *The $B \times 4B$ operator $\mathcal{O}$, where $4B = 2^r$, can be applied to a sequence in $O(r)$ steps on an $r$-dimensional hypercube using a fully normal algorithm.*

**Proof:** First we will describe the algorithm, then prove that it works. As a first step, the destination of every element is computed locally at the source node labeled $j$ by $dest(j) = j \ div \ 4$. The routing sequence begins by comparing the source label digit $d_1^j$ with the destination digit $d_1^{dest}$, and routing on the 1-dimensional edge if $d_1^j \neq d_1^{dest}$. The next step compares the $d_2$ digit of the current location

of the element with the destination digit $d_2^{dest}$, routing along the 2-dimensional edge if necessary. By consecutively increasing the edge dimension $i$ and routing if $d_i \neq d_i^{dest}$, all elements get routed to their destinations in $r$ routing steps and without delay. Note that if the routing had begun on the $r$-dimensional edge and consecutively decreased to dimension 1, there would be routing delays.

The proof that the above algorithm works will use induction. For the base case, let $B = 2$; then $Z_{out}$ has length 8. The elements $Z_{out}(2)$ and $Z_{out}(6)$, located at the nodes labeled 010 and 110, need to be routed to destination nodes labeled 000 and 001, respectively. On the first routing step, the $Z_{out}(6)$ element gets routed to the node labeled 111, and the $Z_{out}(2)$ remains at its current location. On the next routing step, $Z_{out}(2)$ gets routed to the node labeled 000, thus reaching its destination, and $Z_{out}(6)$ gets routed from 111 to 101. On the final routing step, $Z_{out}(6)$ gets routed from 101 to 001, thus completing the operation $\mathcal{O}$ using a fully normal algorithm.

For the inductive case, we first make the observation that after the first 1-dimensional routing step, every element $Z_{out}(j)$ is routed to a unique node with label $k$ that is equivalent to the destination label modulo 2, i.e., $k \equiv dest(j) \bmod 2$. Similarly, after the 2-dimensional routing step, every element is routed to a unique node with label equivalent to $dest(j) \bmod 4$. Thus, after $r-1$ routing steps, each Legendre coefficient $Z_{out}(j)$ is at a node $k$ such that $k \equiv dest(j) \bmod 2^{r-1}$, which we take to be the inductive case. Since the original source nodes of the first $B/2$ Legendre coefficient elements have $d_r = 0$, and since these elements are currently located at nodes having labels with $d_r = 0$, then these elements are at their destinations. The remaining lost Legendre coefficients $Z_{out}(j)$ are currently located at nodes labeled $10d_{r-2}^j \ldots d_1^j$ which differ from their destinations only in the most significant digit $d_r$. Clearly, a single routing step on the $r$-dimensional edge routs each Legendre coefficient $Z_{out}(j)$ to a unique node labeled $k$ such that $k \equiv dest(j) \bmod 2^r$, which completes the routing operation without delay.

■

Now we are in a position to state the main result of this section.

**Proof:** As a matrix factorization (8), the fast Legendre transform is written

$$\tilde{\mathcal{P}}_B^m \cdot \mathbf{s} = \mathcal{O}\mathcal{M}_{r-2} \cdots \mathcal{M}_1 \mathcal{M}_0 \mathcal{I} \cdot \mathbf{s}. \tag{86}$$

By Lemmas 12, 13, and 14, application of any of the matrices on the right hand side of (86) is effected on an $(r+1)$-dimensional hypercube using at most $O(r)$ steps. Each of the matrices is applied using leveled algorithms. Therefore, applying the matrices in the specified order takes at most $O(r^2)$ steps using a series of leveled algorithms, which itself constitutes a leveled algorithm.

■

## 6.3 Two applications

### 6.3.1 Computation of the bispectrum and triple correlation

The techniques of multiple correlations and higher order spectra have only recently been developed for nonabelian Lie groups and their homogeneous spaces by R. Kakarala [45]. Of particular interest is the triple correlation and its associated Fourier transform, the bispectrum.

For functions on the line, the triple correlation is the integral of the product of the function with two independently shifted copies of itself. The resulting function on $\mathbf{R}^2$ determines the original function up to translation. The usefulness of computing the triple correlation derives from the fact that it is (1) insensitive to additive Gaussian noise; (2) retains most of the phase information of the underlying signal and (3) is invariant under translation of the underlying signal. This makes it useful in recovering

a signal from multiple observations in situations in which the signal may be translating on a noisy background.

Kakarala has been able to generalize many of the results for functions on the line to arbitrary locally compact groups and their homogeneous spaces. For particular examples of interest such as the sphere, a suitably defined triple correlation of a band-limited function is again unique up to translation, (assuming that the Fourier coefficients are nonsingular) and insensitive to additive Gaussian noise. This suggests possible applications for global rotational motion compensation and Kakarala goes on to suggest possible applications to imaging the heart [12].

The techniques which we have developed for fast, reliable spherical convolution admit almost immediate application to fast, reliable computation of the triple correlation or bispectrum on the sphere. A detailed explanation of this is beyond the intended scope of this paper. However, the following abreviated discussion should give some indication of our ideas.

To get to the bispectrum on the sphere we must go through the bispectrum for functions on its cover $SO(3)$. If $f \in L^2(SO(3))$, then the triple correlation of $f$ is the function on $SO(3) \times SO(3)$ given by

$$a_{3,f}(s,t) = \int_{SO(3)} f(gs)f(gt)f(g)dg$$

where $dg$ denotes Haar measure on $SO(3)$. Assuming $f$ is integrable on $SO(3)$, then $a_{3,f}$ is integrable on $SO(3) \times SO(3)$.

The irreducible representations of $SO(3)$ are naturally indexed by nonnegative integers, one irreducible of dimension $2l+1$ for each $l \geq 0$, which we denote as $\Lambda_l$. Consequently, the irreducible representations of $SO(3) \times SO(3)$ are given by all possible tensor products $\Lambda_l \otimes \Lambda_{l'}$, so are indexed by all pairs $\{l, l'\}$ with $l \geq l' \geq 0$.

The Fourier transform of $f$ at $\Lambda_l$, denoted as $\hat{f}(l)$ is the integral

$$\hat{f}(l) = \int_{SO(3)} f(g)\Lambda_l(g)^\dagger dg$$

where $\dagger$ indicates conjugate transpose. The Fourier transform of $f$ is the collection $\{\hat{f}(l)\}_{l\geq 0}$.

Similarly, the Fourier transform of a function on $SO(3) \times SO(3)$ will be the analagously defined collection $\{\hat{f}(l,l')\}_{l'\geq l\geq 0}$. The bispectrum of $f$ is Fourier transform of $a_{3,f}$. In [45] Kakarala shows how the bispectrum may be computed from the Fourier transform of $f \in L^2(SO(3))$. For this we need to introduce one more piece of notation. Notice that $SO(3)$ has a natural embedding in $SO(3) \times SO(3)$ as the diagonal subgroup. Considered as such, each representation $\Lambda_{l',l}$ when restricted to the diagonal will be equivalent to a direct sum of appropriate $\Lambda_j$. Thus, there exists an invertible matrix $C_{l,l'}$ such that

$$\Lambda_{l,l'}(s,s) = C_{l,l'}\Big[\Lambda_{j_1(l,l')}(s) \oplus \Lambda_{j_2(l,l')}(s) \oplus \cdots \oplus \Lambda_{j_m(l,l')}(s)\Big]C_{l,l'}^\dagger$$

for suitable indices $j_i(l,l')$.

**Theorem** *([45], Lemma 3.2.3) With the notation as above,*

$$\widehat{a}_{3,f}(l,l') = \hat{f}(l) \otimes \hat{f}(l')C_{l,l'}\Big[\hat{f}(j_1(l,l'))^\dagger \oplus \hat{f}(j_2(l,l'))^\dagger \oplus \cdots \oplus \hat{f}(j_m(l,l'))^\dagger\Big]C_{l,l'}^\dagger \tag{87}$$

When $f \in L^2(SO(3))$ comes from a function on the sphere, (i.e. $f \in L^2(SO(3))$ is right $SO(2)$-invariant) then the matrix $\hat{f}(l)$ will have entries all 0 except, possibly, for a single column which (up to a normalization constant) will contain the associated Legendre transforms

$$\{\hat{f}(l,-l), \ldots, \hat{f}(l,0), \ldots, \hat{f}(l,l)\}.$$

Thus, if $f$ is band-limited, then the bispectrum will only involve a finite number of Fourier transforms. For each $l, l'$, $\widehat{a}_{3,f}(l, l')$ can then be computed directly as follows. Compute first the spherical harmonic expansion as described in Section 3. This precomputes all possible Fourier transforms $\hat{f}(l)$ for any $f \in L^2(S^2)$. The inner direct sum of the matrices

$$\left[\hat{f}(j_1(l,l'))^\dagger \oplus \hat{f}(j_2(l,l'))^\dagger \oplus \cdots \oplus \hat{f}(j_m(l,l'))^\dagger\right]$$

is then constructed by retrieving the appropriate associated Legendre transforms and organizing them together into a single sparse block diagonal matrix. This is then conjugated by the precomputed change of basis matrices $C_{l,l'}$. Finally the lefthand factor

$$\hat{f}(l) \otimes \hat{f}(l')$$

simply requires the computation of all possible pointwise products $\hat{f}(l,m)\hat{f}(l',m')$ organized as the appropriate single nonzero column in some suitably defined matrix. These matrices are then all multplied together, giving the relevant component of the bispectrum.

A more careful discussion of this computation will be the subject of a future paper.

### 6.3.2 Matched Filters

One simple application of the techniques of this paper may be found in certain problems of detection, estimation, and pattern matching for data defined on the sphere. This sort of data arises in geophysics, computer vision, or quality assurance for computer designed and manufactured parts.

A simple problem arising in this area may be stated as follows: Suppose we are considering a known signal or pattern in the directional data setting, described by a function, $f(\omega)$ on the sphere. In many situations, we are interested in determining the presence or absence of this signal in data coming from measurements of some real world phenomenon. This is often made more difficult by the presence of some random interference, or noise, in the measurements. In the simplest cases, we assume that one of two hypotheses obtains for the measured data, $\mathbf{y}(\omega)$:

- $H_0$: $\mathbf{y}(\omega) = \mathbf{n}(\omega)$
- $H_1$: $\mathbf{y}(\omega) = f(\omega) + \mathbf{n}(\omega)$,

where $\mathbf{n}(\omega)$ is a random process on the sphere representing the noise. Our task is then to devise an algorithm which takes a particular instance of the measured data and returns an assessment of whether or not the signal is present in the data.

A more interesting version of this problem occurs when, in addition to the additive noise, the pattern signal $f$ may have undergone a rotation which is unknown to us. That is, when $f$ is present, the measured data has the form

$$\mathbf{y}(\omega) = \Lambda(g)f(\omega) + \mathbf{n}(\omega),$$

where $g$ is an unknown element of $SO(3)$, and $\Lambda(g)$ is the associated operator, $\Lambda(g)f(\omega) = f(g^{-1}\omega)$. In this case we have the more involved detection and estimation problem; determine if a rotated version of the pattern is present, and if so, estimate the value of the rotation parameter $g$. We will concern ourselves with this question.

The intuitive approach to this involves a template matching operation. That is, one correlates the data with the pattern one is looking for, which amounts to forming the inner product of the data with a large number of shifted versions of the pattern. Those shifts which produce a large inner product, or correlation, between the pattern and the data are regarded as good indicators that there really is

a copy of the pattern shifted to the corresponding location and buried in the noise. This correlation process is known as matched filtering; it amounts to computing the function

$$\chi(g) = \int_{S^2} y(\omega)\Lambda(g)f\omega\, d\omega.$$

This matched filter is also indicated by a standard statistical analysis for these sorts of problems. This analysis yields optimal detection and estimation solutions involving a computation of the appropriate *likelihood function*, [33, 58, 54]. Basically, for a given measurement of data, the value of this function gives the likelihood of having made that particular observation given a certain hypothesis (signal present or signal absent) or a given value of the unknown parameter.

For example, suppose we know that a rotated version of the signal is present, in the data process $\mathbf{y}(\omega)$, and we wish to know where it is. This is the same as estimating the rotation parameter $g$. We assume that the additive noise $\mathbf{n}(\omega)$ is Gaussian and white. The latter term refers to the covariance structure of the noise, implying first that the covariance $R(\omega_1, \omega_2) = \mathbf{E}[\mathbf{n}(\omega_1)\mathbf{n}(\omega_2)]$ is actually rotation independent, so that $R(\omega_1, \omega_2) = R(g\omega_1, g\omega 2)$ for any rotation $g \in SO(3)$. This property is sometimes referred to as "stationarity." A consequence of stationarity is that $R$ is determined by the values $R(\omega)$ $= R(\omega, \eta)$, for $\eta$ the north pole of the sphere. White noise is a particular stationary noise with point mass covariance; $R(\omega) = \sigma^2\delta_\eta(\omega)$. Strictly speaking, this requires the usual sorts of mathematical temporizing required when dealing with distributions; we'll assume that is familiar.

The likelihood $L(g)$ of a particular value of the parameter $g$ given the data $y$ is the probability density for the random variable $\mathbf{y} = \Lambda(g)f + \mathbf{n}$ evaluated at the particular observed measurement values $y_o$; this is the same as $p_{\mathbf{y}}(y_o; g) = p_{\mathbf{n}}(y_o - \Lambda(g)f)$. Using our assumptions on $\mathbf{n}$ and some limiting arguments, we obtain for our likelihood:

$$\begin{aligned} L(g) &\propto e^{-\frac{||y_o - \Lambda(g)f||_2^2}{2\sigma^2}} \\ &\propto e^{\frac{<y_o, \Lambda(g)f>}{\sigma^2}}, \end{aligned}$$

as the other terms which come from expanding the norm, $e^{-\frac{||y_o||_2^2}{2\sigma^2}}$ and $e^{-\frac{||\Lambda(g)f||_2^2}{2\sigma^2}} = e^{-\frac{||f||_2^2}{2\sigma^2}}$ are constant, independent of $g$. Thus the maximimum likelihood estimate of $g$ is

$$\text{Arg Max}_{g\in SO(3)} \int_{S^2} y(\omega)\Lambda(g)f\omega\, d\omega.$$

Let us consider now a simple case in which the pattern signal is rotationally symmetric. In fact, we take $f$ to be the analog of the normal density on the sphere, the Fisher von-Mises density $C_\kappa \exp(\kappa \cos\theta)$. Here, $\kappa$ is a concentration parameter, $C_\kappa$ a normalizing factor. In this case, the matched filter expression actually reduces to a function defined on the sphere, rather than the entire group, due to rotation invariance. Below we show the results of some experiments in which $f$ is rotated and buried in white noise, and then passed through a matched filter. The results are shown in Figure 18.

# 7 Conclusion

We have presented here an efficient and numerically reliable algorithm for convolution of band-limited functions on the 2-sphere. Our implementation shows that the run-time efficiency is better than the naive at useful problem sizes and appears to be readily parallelizable. The stability bypass technique offers a method for improving the numerical properties of the algorithm while maintaining asymptotic

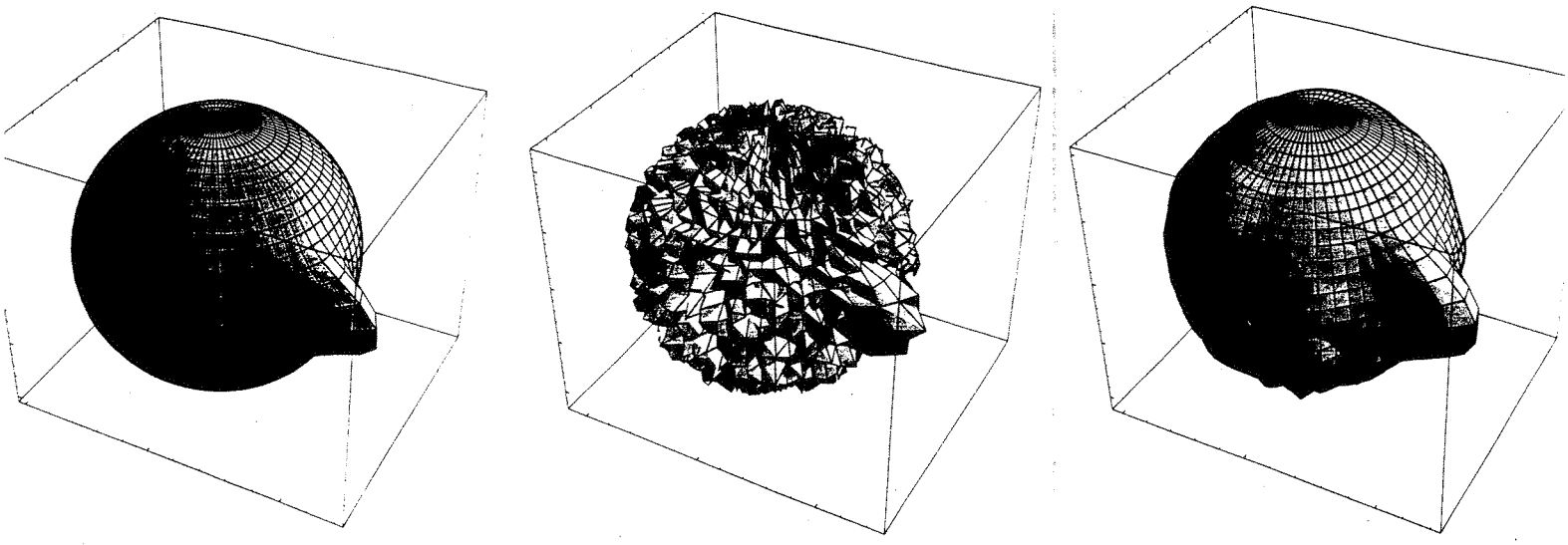


Figure 18: (a) The pattern signal $f(\omega)$ is the Fisher-von Mises density $C_\kappa \exp(\kappa \cos\theta)$, with concentration paremeter $\kappa = 64$, and rotated by an arbitrary rotation $g$ on the sphere; (b) $\Lambda(g)f(\omega)$ has been buried in additive white noise $n(\omega)$ to simulate noisy measured data $y(\omega)$; (c) The likelihood function $L(g)$ computed by using the fast convolution algorithm to convolve $y(\omega)$ with a matched filter. The position $g_{max}$ of the maximum value of $L(g)$ indicates the maximum likelihood estimate of the position of the pattern signal $f(\omega)$.

and run-time superiority over the naive computation. Since the fast algorithm as well as the stability bypass technique can be used for transforms onto any space of basis functions generated by a three-term recurrence rule, we have high hopes for being able to apply these techniques in a much more general setting. A natural place to look next would be at other useful compact groups and their quotients such as the higher spheres or the Lie groups $SU(n)$.

The possibility of efficient and reliable algorithms of this kind open the door for many types of potential applications. As we have indicated construction of matched filters for the is one such applications which may have applications to pattern recognition and computer vision. Another immediate application is to the numerical computation of the bispectrum for band-limited functions on the 2-sphere as well as other compact groups and their quotients. These methods have been identified as having potential applications for imaging the heart. In future work we hope to explore these and other applications more fully.

## References


[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform", *IEEE Transactions on Computers*, January 1974, pps. 90-93.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading (1976).

[3] B. Alpert and V. Rokhlin, A fast algorithm for the evaluation of Legendre expansions, *Research Report YALEU/DCS/RR-671*, Yale University, Department of Computer Science (1989).

[4] L. Auslander and R. Tolmieri, Is computing with the fast Fourier transform pure or applied mathematics?, *Bulletin of the American Mathematical Society*, 1 (New Series), 847-897 (1979).

[5] P. Barrucand and D. Dickinson, On the Associated Legendre Polynomials, in *Orthogonal Expansions and their Continous Analogues*, Southern Illinois University Press, Carbondale (1968).

[6] U. Baum, M. Clausen, B. Tietz, "Improved Upper Complexity Bounds for the Discrete Fourier Transform", in *Applicable Algebra in Engineering, Communication, and Computing*, Springer-Verlag (1991).

[7] T. Beth, On the computational complexity of the general discrete Fourier transform, *Theoretical Computer Science*, 51, 331-339 (1987).

[8] L. C. Biedenharn and J. D. Louck, *Angular Momentum in Quantum Mechanics*, Addison Wesley, Reading (1981).

[9] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numeric Problems* Elsevier, New York 1975.

[10] N. Bshouty, M. Kaminski, and D. Kirkpatrick. Addition requirements for matrix and transposed matrix products. *J. of Algorithms*, **9**:354-364, 1988.

[11] D. Calvetti, A stochastic roundoff error analysis for the fast Fourier transform, *Math. Comput.*, 56, no 194, 755-774 (1991).

[12] C. W. Chen and T. S. Huang, Epicardial motion and deformation estimation from coronary artery bifurcation points, in *Proc. of Third. Int. Conf. on Comp. Vision, Dec. 4-7, 1990*, IEEE Press, (1990), 456-460.

[13] M. Clausen and U. Baum, *Fast Fourier Transforms*, Wissenschaftsverlag, Manheim (1993).

[14] J. W. Cooley and J. W. Tukey, An algorithm for machine calculation of complex Fourier series, *Math. Comput.*, 19, 297-301 (1965).

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press, (1990).

[16] P. Diaconis, Average running time of the fast Fourier transform, *J. Algorithms*, 1, 187-208 (1980).

[17] P. Diaconis, *Group Representations in Probability and Statistics*, IMS, Hayward, California (1988).

[18] P. Diaconis and D. Rockmore, Efficient computation of the Fourier transform on finite groups, *Journal of the American Mathematical Society*, 3, no. 2, 297-332 (1990).

[19] G. A. Dilts, Computation of Spherical Harmonic Expansion Coefficients via FFT's, *Journal of Computational Physics*, vol 57, no. 3, 439-453 (1985).

[20] J. R. Driscoll and D. Healy. Computing Fourier transforms and convolutions on the 2-sphere. *Adv. in Appl. Math.*, To appear.

[21] J. R. Driscoll, D. Healy and D. Rockmore. Fast spherical transforms for distance transitive graphs. *Technical Report, Department of Mathematics and Computer Science, Dartmouth College*, 1989.

[22] W. Freeden, On integral formulas of the (unit) sphere and their application to numerical computation of integrals, *Computing*, 25, 131-146 (1980).

[23] F. A. Gilbert, Inverse problems for the earth's normal modes, in *Mathematical Problems in the Geophysical Sciences, Vol I*, American Mathematical Society, Providence (1971).

[24] N. C. Gallagher, G. L. Wise, and J. W. Allen, A novel approach for the computation of Legendre polynomial expansions, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-26, no. 1, 105-106 (1978).

[25] G. H. Golub and C. F. Van Loan, *Matrix Computations, second edition*, Johns Hopkins University Press, Baltimore (1989).

[26] D. Healy and P. Kim, Spherical deconvolution with application to geometric quality assurance, *Technical Report, Department of Mathematics and Computer Science, Dartmouth College*, (1993).

[27] D. Healy and P. Kim, An empirical Bayes approach to directional data and efficient computation on the sphere, *Technical Report, Department of Mathematics and Computer Science, Dartmouth College*, (1993).

[28] G. T. Herman (Ed.), *Image Reconstruction from Projections*, Springer-Verlag, New York (1979).

[29] R. Holmes, Signal processing on finite groups, *Technical Report 873, MIT, Lincoln Laboratory*, (1990).

[30] R. Holmes, Mathematical foundations of signal processing, II. *Technical Report 781, MIT, Lincoln Laboratory*, (1987).

[31] K. Kanatani, *Group-Theoretical Methods in Image Understanding*, Springer-Verlag, NY (1990).

[32] M. Karpovsky and E. Trachtenberg, Filtering in a communication channel by Fourier transforms over finite groups, in *Spectral Techniques and Fault Detection*, M. Karpovsky (ed.) Academic Press, NY (1985), 179-212.

[33] S. Kay, *Fundamentals of Statistical Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ 1993.

[34] K. Kobayashi, Solution of multi-dimensional neutron transport equation of the spherical harmonics method using the finite Fourier transformation and quadrature formula, *Transport Theory and Stat. Phys.*, 14, 63-81 (1985).

[35] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufman Publishers (1992).

[36] G. J. Martyna and B. J. Berne, Structure and energies of $Xe_n^-$: Many body polarization effects, *J. Chem. Phys.* (7) **90** (1989), 3744-3755.

[37] D. Maslen, *Fast Transforms and Sampling for Compact Groups*, Ph.D. Thesis, Department of Mathematics, Harvard University (1993).

[38] S. Moore, D. Healy, and D. Rockmore. Symmetry stabilization for polynomial evaluation and interpolation. *Lin. Alg. Appl.*, To appear.

[39] S. S. B. Moore and L. Wisniewski, Supermoduli Tree-based Cosine Transform Algorithms with Hardware Considerations, In preparation.

[40] S. D. Morgera, Efficient synthesis and implementation of large discrete Fourier transformations, *SIAM J. Comput.*, 9, 251-272 (1980).

[41] W. Neutsch, Optimal spherical designs and numerical integration on the sphere, *J. Comp. Phys.*, 51, 313-325 (1983).

[42] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, New York 1982.

[43] S. A. Orzag, Fast eigenfunction transforms, in *Science and Computers*, Academic Press, Orlando (1986).

[44] C. H. Papadimitriou, Optimality of the fast Fourier transform, *J. Assoc. Comput. Mach.*, 26, 95-102 (1979).

[45] K. Ramakrishna, *Triple Correlation on Groups*, Ph.D. Thesis, Dept. of Math. University of California, Irivine (1992).

[46] G. U. Ramos, Roundoff error analysis of the fast Fourier transform, *Math. Comp.*, **25** (1971), 757-768.

[47] L. Robin, *Fonctions Spheriques de Legendre et Fonctionns Spheroidales*, Gauthier-Villars, Paris (1959).

[48] E. J. Schwabe, *Efficient Embeddings and Simulations for Hypercubic Networks*, Ph.D. Thesis, MIT/LCS/TR-508, Massachusetts Institute of Technology, June 1991.

[49] J. T. Schwartz, Mathematics Addresses Problems in Computer Vision for Advanced Robotics, *SIAM News*, 18, 3 (1985).

[50] 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

[51] A. Tam, *Optimal Choice of Directions for the Reconstruction of an Object from its Plane Integrals* , Ph.D. Thesis, U.C. Berkeley (1982).

[52] R. Tolimieri, M. An, and C. Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, New York 1989.

[53] C. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, Philadelphia (1992).

[54] H. Van Trees, *Detection, Estimation and Modulation Theory, Vol I*, Wiley, New York, 1968.

[55] N. J. Vilenkin, *Special Functions and the Theory of Group Representations*, American Mathematical Society, Providence (1968).

[56] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice Hall, Englewood Cliffs (1963).

[57] S. Winograd, On computing the discrete Fourier transform, *Math. Comp*, 32, 175-199 (1978).

[58] P. M. Woodward, *Probability and Information Theory, with Applications to Radar*, Artech House, Dedham MA 1980.