Rapid Application of the Spherical Harmonic Transform via Interpolative Decomposition Butterfly Factorization

James Bremer
Department of Mathematics, University of California, Davis, California, USA

Ze Chen
Department of Mathematics, National University of Singapore, Singapore

Haizhao Yang
Department of Mathematics, Purdue University, USA

February 9, 2021

Abstract

We describe an algorithm for the application of the forward and inverse spherical harmonic transforms. It is based on a new method for rapidly computing the forward and inverse associated Legendre transforms by hierarchically applying the interpolative decomposition butterfly factorization (IDBF). Experimental evidence suggests that the complexity of our method — including all necessary precomputations — is $\mathcal{O}\left(N^2 \log^3(N)\right)$ in terms of both flops and memory, where N is the order of the transform. This is nearly asymptotically optimal. Moreover, unlike existing algorithms which are asymptotically optimal or nearly so, the constants in the running time and memory costs of our algorithm are small enough to make it competitive with state-of-the-art $\mathcal{O}\left(N^3\right)$ methods at relatively small values of N (e.g., $N = 1024$). Numerical results are provided to demonstrate the effectiveness and numerical stability of the new framework.

Keywords. Spherical harmonic transform, Legendre transform, block partitioning, butterfly factorization, interpolative decomposition, randomized algorithm

AMS Classifications. 33C55, 42C10, 68W20, 15A23

1 Introduction

This paper is concerned with the efficient application of the forward and inverse spherical harmonic transforms (SHT). These transformations play an important role in many scientific computing applications, including in the fields of numerical weather prediction and climate modeling [24, 21, 23], and are significant components in many numerical algorithms. The forward SHT of degree N maps the coefficients in the expansion

$$f(\theta, \phi) = \sum_{k=0}^{2N-1} \sum_{m=-k}^{k} \beta_{k,m} \overline{P}_k^{|m|}(\cos(\theta)) e^{im\phi}, \tag{1}$$

where $\overline{P}_k^m(x)$ denotes the L^2 normalized associated Legendre function of order m and degree k, to the values of the expansion at a grid of discretization nodes formed from the tensor product of a $2N$-point Gauss-Legendre quadrature in the variable $x = \cos(\theta)$ and a $(4N-1)$-point trapezoidal quadrature rule in the variable ϕ. The inverse SHT is, of course, the mapping which takes the

values of the function $f(\theta, \phi)$ at the discretization nodes to the coefficients in the expansion. More explicitly, the expansion f is represented either via the coefficients in (1) or by its values at the set of points

$$\{(\theta_l, \phi_j) \quad : \quad l = 0, 1, \ldots, 2N-1, \quad j = 0, 1, \ldots, 4N-2\}, \tag{2}$$

where

$$-1 < \cos(\theta_0) < \cos(\theta_1) < \cdots < \cos(\theta_{2N-2}) < \cos(\theta_{2N-1}) < 1 \tag{3}$$

are the nodes of the $2N$-point Gauss-Legendre quadrature rule and $\phi_0, \phi_1, \ldots, \phi_{4N-3}, \phi_{4N-2}$ are the equispaced nodes on $(0, 2\pi)$ given by the formula

$$\phi_j = \frac{2\pi(j + \frac{1}{2})}{4N-1}, \text{ for } j = 0, 1, \ldots, 4N-3, 4N-2. \tag{4}$$

The forward SHT maps the coefficients in (1) to the values of the expansion at the discretization nodes (2), and the inverse SHT takes the values of the expansion at the discretization nodes to the coefficients.

If we let

$$g(m, \theta) = \sum_{k=|m|}^{2N-1} \beta_{k,m} \overline{P}_k^{|m|}(\cos(\theta)), \tag{5}$$

then (1) can be written as

$$f(\theta, \phi) = \sum_{m=-2N+1}^{2N-1} g(m, \theta) e^{im\phi}. \tag{6}$$

From (6), it is clear that given the values of $g(m, \theta)$ for each $m = -2N+1, \ldots, 2N+1$ and each $\theta_0, \ldots, \theta_{2N-1}$, the values of $f(\theta, \phi)$ at the discretization nodes (2) can be computed in $\mathcal{O}\left(N^2 \log(N)\right)$ operations by applying the fast Fourier transform $\mathcal{O}(N)$ times. Similarly, the inverse of this operation, which takes the values of $f(\theta, \phi)$ to those of $g(m, \theta)$, can be calculated in $\mathcal{O}\left(N^2 \log(N)\right)$ operations using $\mathcal{O}(N)$ fast Fourier transforms.

We will refer to the mapping which, for a fixed m, takes the coefficients in the expansion (5) to the values of $g(m, \theta)$ at the $\mathcal{O}(N)$ discretization nodes in θ as the forward associated Legendre transform (ALT). The inverse mapping, which takes the values of $g(m, \theta)$ to the coefficients in the expansion, will be referred to as the inverse ALT. The naive approach to applying one of these transforms requires $\mathcal{O}\left(N^2\right)$ operations, and using such an approach leads to an SHT with an $\mathcal{O}\left(N^3\right)$ operation count.

There is a large literature devoted to accelerating the application of the associated Legendre transform (we review it in Section 1.1). However, existing algorithms leave much to be desired. The most widely used class of methods allow for the application of the ALT in $\mathcal{O}\left(N \log^{\kappa}(N)\right)$ operations, but only after an $\mathcal{O}\left(N^2\right)$ precomputation phase. Existing algorithms that have quasilinear complexity (when all necessary precomputations are taken into account) have such poor constants in their running times that they are slower than the former class of methods at practical values of N. Indeed, the current state-of-the-art method appears to be [20], which has very favorable constants but requires a precomputation phase whose complexity is $\mathcal{O}\left(N^2\right)$.

In this paper, we propose a new method for applying the forward ALT whose total running time, including both the precomputation and application phases, is $\mathcal{O}\left(N \log(N) \times (r(N))^2\right)$, where $r(N)$ is a bound on the ranks of certain blocks of the transformation matrix. We conjecture that $(r(N))^2$ grows as $\mathcal{O}\left(\log^2(N)\right)$. Assuming this is correct, the total running time of our algorithm for applying the ALT is $\mathcal{O}\left(N \log^3(N)\right)$. Proving a rigorous bound on $r(N)$ appears to be quite difficult, and, for now, we are relying on experimental evidence regarding the running time of our

algorithm. Assuming our conjecture regarding the running time of our ALT is correct, the SHT can be applied using our ALT in $\mathcal{O}\left(N^2 \log^3(N)\right)$ time.

Our algorithm operates by hierarchically applying the interpolative decomposition butterfly factorization (BF) [15, 3] (a newly proposed nearly linear scaling butterfly algorithm [10, 12, 14, 9, 11]) and using randomized low-rank approximation to speed up the calculation of the ALT.

Butterfly algorithms are a collection of techniques for rapidly applying the matrices which result from discretizing oscillatory integral operators. They exploit the fact that these matrices have the complementary low-rank property (see [10] for a definition). A large class of special function transforms are integral operators of the appropriate type [14], and consequently can be applied rapidly using butterfly algorithms. Indeed, in the special case $m = 0$, the ALT can be applied via standard butterfly algorithms in $\mathcal{O}(N \log(N))$ time. These results do not, however, extend to the case $m > 0$. In that event, the associated Legendre functions are not oscillatory on the entire domain of interest. Instead, $\tilde{P}_n^m(\cos(\theta))$ is nonoscillatory on the interval

$$\left(0, \arcsin\left(\frac{\sqrt{m^2 - 1/4}}{n + 1/2}\right)\right) \tag{7}$$

and oscillatory on

$$\left(\arcsin\left(\frac{\sqrt{m^2 - 1/4}}{n + 1/2}\right), \frac{\pi}{2}\right) \tag{8}$$

(see Figure 1, which contains plots of $\tilde{P}_n^m(\cos(\theta))$ for two different pairs of the parameters n and m). As a consequence, the integral operator associated with the ALT when $m > 0$ is not of the purely oscillatory type whose discretizations have the complementary low rank property.

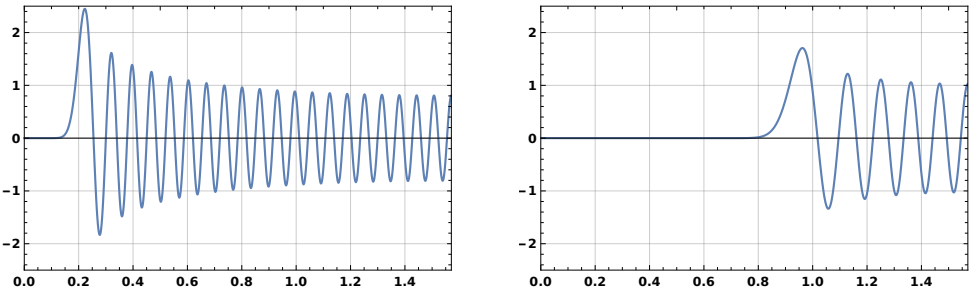


Figure 1: On the left is a plot of the L^2 normalized associated Legendre function $\tilde{P}_n^m(\cos(\theta))$ in the case $n = 100$ and $m = 20$. On the right is a plot of $\tilde{P}_n^m(\cos(\theta))$ when $n = 100$ and $m = 80$.

In order to overcome this difficulty we apply the following methodology:

- We hierarchically partition the transformation matrix into purely oscillatory and purely non-oscillatory blocks (see Figure 2 (b)).
- In the purely nonoscillatory blocks, the corresponding matrix is numerically low-rank, and hence its application to a vector can be accelerated to obtain linear scaling by randomized low-rank approximation algorithms.
- The matrices corresponding to purely oscillatory blocks admit complementary low-rank structures, the application of which to a vector can be accelerated via butterfly algorithms. We use the relatively new interpolative decomposition butterfly factorization (IDBF) [15], which

yields nearly linear scaling in the degree N of the ALT transform in both precomputation and application.

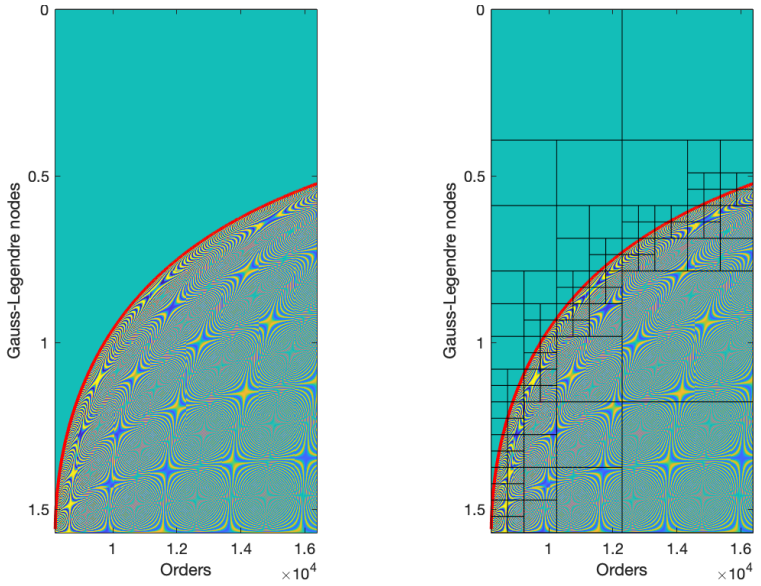


Figure 2: An illustration of the partitioning process of an odd ALT matrix when $N = 8192$ and order $m = 8192$. (a) The odd matrix with a piecewise continuous curve (red color) indicating the positions of turning points. (b) The hierarchically partitioned blocks of the odd matrix.

The scheme relies heavily on the algorithm of [2] for the numerical evaluation of the associated Legendre functions. That algorithm allows each evaluation to be performed in time independent of the parameters n and m. If standard methods for calculating $\tilde{P}_n^m$, which have running times which grow with the parameters n and m, were used instead, the running time of our algorithm for applying the ALT would no longer scale as $\mathcal{O}\left(N \log^3(N)\right)$.

1.1 Related works

There has been a significant amount of research aimed at accelerating the associated Legendre Transform in order to more rapidly apply the spherical harmonic transform. In [5], an algorithm for applying the ALT which results in an SHT whose running time is $\mathcal{O}\left(N^2 \log(N)^2\right)$ is described. However, this algorithm suffers from increasing numerical instability as N increases. In [13] and [16], asymptotically optimal schemes for the ALT which are numerically stable are described, but the constants in their running time make them unappealingly slow for practical values of N. The contribution [19] introduces a scheme based on the fast multiple method. It can apply the SHT in $\mathcal{O}\left(N^2 \log(N)\right)$ operations after a precomputation phase, the direct evaluation time of which is $\mathcal{O}\left(N^3\right)$ and could be reduced to nearly linear scaling in theory. However, this faster variant of the precomputation portion of the algorithm must be executed in extended precision arithmetic, which would most likely make it slow in applications.

The most widely-used algorithm today appears to be that of [20]. It uses the butterfly transform described in [14] to evaluate the ALT. Each ALT takes $\mathcal{O}\left(N^2\right)$ and $\mathcal{O}\left(N \log^3(N)\right)$ operations in the precomputation and application, respectively. This, of course, results in an SHT with a cost of $\mathcal{O}\left(N^3\right)$ for precomputation and $\mathcal{O}\left(N^2 \log^3(N)\right)$ for application. A highly-optimized computational

package based on [20] was developed in [17]. It is widely used and most likely represents the current state-of-the-art for rapidly applying the SHT. Though the application phase of this algorithm is nearly optimal, its precomputation phase is still prohibitively expensive when N is large.

In [24] an algorithm for applying the ALT which bears some similarities to our scheme was proposed. It operates by partitioning the transformation matrix in the manner shown in Figure 3. The application phase of the resulting algorithm has lower complexity than that used in [20] and yields somewhat improved accuracy (roughly an extra digit of precision). However, the method of [24] still requires a precomputation phase whose running time behaves as $O(N^3)$.

In [18], an algorithm that makes use of a rapid transformation between spherical harmonic expansions and bivariate Fourier series via the butterfly transform and hierarchically off-diagonal low-rank matrix decompositions. Although the application time of this algorithm $\mathcal{O}\left(N^2 \log^2(N)\right)$, it requires a precomputation whose running time grows as $\mathcal{O}\left(N^3 \log(N)\right)$.

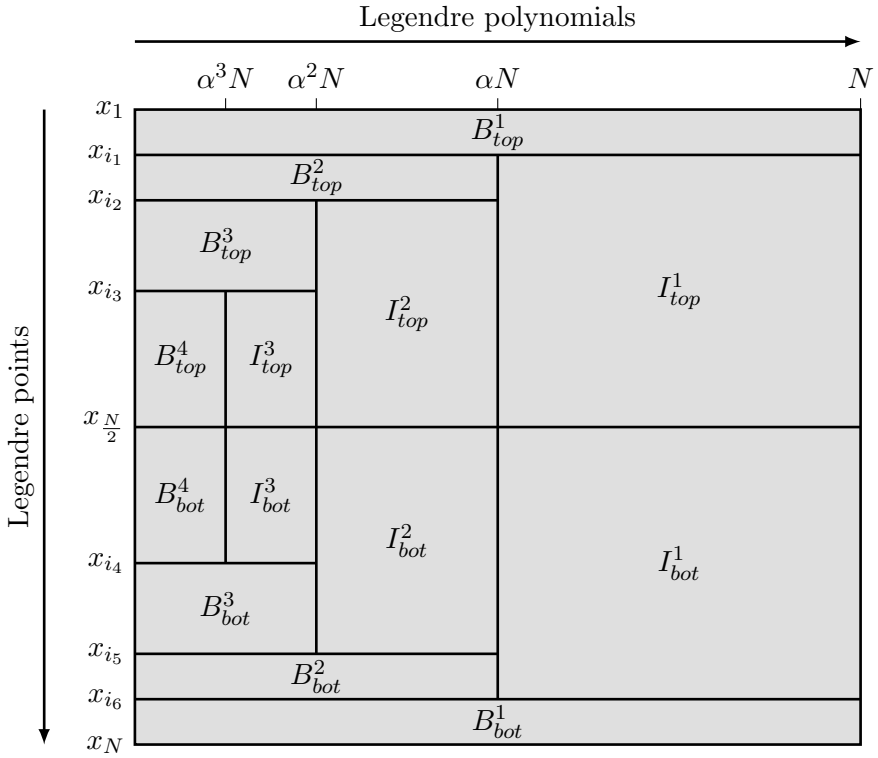


Figure 3: Block partitioning of the Legendre-Vandermonde matrix in [24], when $N = 1024$. x_i, $i = 1, 2, \ldots, N$, are the Legendre points. The parameters $i_1, i_2, \ldots, i_6$ and α are the computed partitioning coefficients, which are able to divide the Legendre-Vandermonde matrix into boundary parts (denoted by symbol B) and internal (denoted by symbol I) parts. The internal parts can be compressed by the BF while the boundary parts are directly computed for the corresponding matvecs.

1.2 Outline of this paper

The rest of the paper is organized as follows. In Section 2, we discuss existing low-rank matrix factorization techniques. Section 3 proposes a new algorithm for applying the Legendre Transform which is based on these factorization techniques. In Section 4, we discuss the computational

complexity of our algorithm. Again, we do not have a rigorous bound on its running time, but we estimate it under an assumption on the behavior of the ranks of certain subblocks of the matrix discretizing the ALT. Section 5 describes several numerical experiments conducted to assess the efficiency of the proposed algorithm.

For simplicity, we adopt MATLAB notations for the algorithm described in this paper: given row and column index sets I and J, $K(I, J)$ is the submatrix with entries from rows in I and columns in J; the index set for an entire row or column is denoted as " : ".

2 Low rank factorizations and butterfly algorithms

In this section, we discuss several existing algorithms which exploit rank deficiency to rapidly apply certain classes of matrices. These are used by the algorithm of Section 3 for the application of the ALT. Subsection 2.1 outlines the linear scaling interpolative decomposition (ID) method, which is an important tool for the interpolative decomposition butterfly factorization (IDBF) discussed in Subsection 2.2. Subsection 2.3 describes low-rank approximation via randomized sampling.

2.1 Linear scaling interpolation decomposition

This subsection reviews the algorithm of [15] for the construction of interpolative decompositions.

A column interpolative decomposition, which will abbreviate by *cID*, of $A \in \mathbb{C}^{m\times n}$ is a factorization of the form

$$A \approx A(:, q)V, \tag{9}$$

where q is an index set specifying k columns of A and V is a $k \times n$ matrix. The set q is called the *skeleton* index set, and the rest of the indices are called *redundant* indices. The matrix V is called the column interpolation matrix. The algorithm described in this section takes as input a desired precision ϵ and adaptively determines k such that

$$\|A - A(:, q)V\|_2 \leq \epsilon. \tag{10}$$

The numerical rank of A to precision ϵ is defined via

$$k_\epsilon = \min \left\{ \operatorname{rank}(X) : X \in \mathbb{C}^{m\times n}, \|A - X\|_2 \leq \epsilon \right\}, \tag{11}$$

and it is the optimal possible value of k. In most cases, the algorithm of this section forms factorizations with k equal to or only slightly larger than k_ϵ.

The algorithm also takes as input a parameter r_k, which we refer to as the "adaptive rank," which serves as an upper bound for the rank of A. It proceeds by first constructing an index set s containing $t \cdot r_k$ rows of A chosen from the Mock-Chebyshev grids as in [22, 8, 1] or randomly sampled points. Here, t is an oversampling parameter.

We next compute a rank revealing QR decomposition of $A(s, :)$. That is, we decompose $A(s, :)$ as

$$A(s, :)\Lambda \approx QR = Q[R_1\ R_2], \tag{12}$$

where the columns of $Q \in \mathbb{C}^{tk\times k}$ are an orthonormal set in $\mathbb{C}^m$, $R \in \mathbb{C}^{k\times n}$ is upper trapezoidal, and $\Lambda \in \mathbb{C}^{n\times n}$ is a carefully chosen permutation matrix such that $R_1 \in \mathbb{C}^{k\times k}$ is nonsingular. The value of k is chosen so that the L^2 error in the approximation (12) is somewhat smaller than ϵ. We now define

$$T = (R_1(1 : k, 1 : k))^{-1}[R_1(1 : k, k + 1 : tk)\ R_2(1 : k, :)] \in \mathbb{C}^{k\times(n-k)}, \tag{13}$$

such that

$$A(s,q) = QR_1(1:k, 1:k).$$

Then

$$A(s,:) \approx A(s,q)V \tag{14}$$

with $V = [I, T]\Lambda$ and the approximation error determined by the error in the rank-revealing QR decomposition. Moreover,

$$A \approx A(:,q)V \tag{15}$$

with an approximation error coming from the QR truncation and the error incurred in (9) when performing interpolation from the subsampled rows of A using the interpolation matrix V. When the obtained accuracy is insufficient, the procedure is repeated with an increased k. Using the steps outlined above, the construction of this factorization requires $\mathcal{O}\left(nk^2\right)$ operations and $\mathcal{O}\left(nk\right)$ storage.

A row interpolative decomposition (abbreviated *rID*) of the form

$$A \approx UA(q,:) \tag{16}$$

can be constricted in a similar fashion in $\mathcal{O}\left(mk^2\right)$ operations using $\mathcal{O}\left(mk\right)$ storage. We refer to U as the *row interpolation matrix*.

2.2 Interpolative decomposition butterfly factorization

In this section, we briefly discuss the properties of the interpolative decomposition butterfly factorization, and the algorithm of [15] for producing it. We refer the reader to [15] for a detailed discussion.

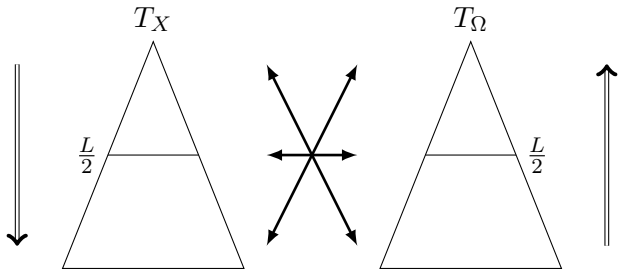


Figure 4: Trees of the row and column indices. Left: T_X for the row indices X. Right: T_Ω for the column indices Ω. The interaction between $A \in T_X$ and $B \in T_\Omega$ starts at the root of T_X and the leaves of T_Ω.

We first recall the definition of a complementary low-rank matrix given in [10]. Suppose that $K \in \mathbb{C}^{N\times N}$. We denote the set of rows of K by X and the set of columns of K by Ω. We introduce two trees T_X and T_Ω that are generated by bisecting the sets X and Ω recursively, and the elements of T_X and T_Ω consist of subsets of X and Ω, respectively. Assume that both trees have the same depth $L = \mathcal{O}\left(\log\left(N\right)\right)$ with the top-level being level 0 and the bottom one being level L (see Figure 4 for an illustration). The top level of T_X (and T_Ω) contains all the indices in X (and Ω), while each leaf at the bottom level contains $O(1)$ indices. Then, the matrix K is said to satisfy the *complementary low-rank property*, if the following property holds: for any level ℓ, any node $A \in T_X$ at level ℓ, and any node $B \in T_\Omega$ at level $L - \ell$, the submatrix $K(A, B)$, obtained by restricting K to the rows indexed by the points in A and the columns indexed by the points in B,

is numerically low-rank. By numerically low-rank, we mean that the ranks of the submatrices grow no more quickly than $\log^{\kappa}(N)$ with the size of the matrix K. In many cases of interest, $\kappa = 0$ — that is, the ranks of the submatrices are bounded by a constant independent of N. See Figure 5 for an illustration of the complementary low-rank property.

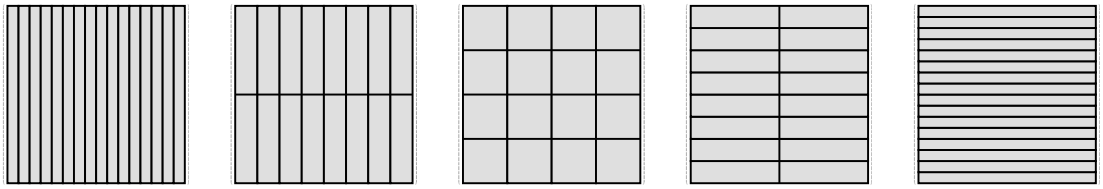


Figure 5: Hierarchical decomposition of the row and column indices of a 16×16 matrix. The dyadic trees T_X and T_Ω have roots containing 16 rows and 16 columns respectively, and their leaves containing only a single row or column. The partition above indicates the complementary low-rank property of the matrix, and assumes that each submatrix is rank-1.

An interpolative decomposition butterfly factorization (IDBF) of a complementary low-rank matrix K is a factorization of the form

$$K \approx U^L U^{L-1} \cdots U^h S^h V^h \cdots V^{L-1} V^L, \tag{17}$$

where L is the number of levels in the trees T_X and T_Ω, $h = L/2$ and each of the matrices U^l and V^l is sparse with $\mathcal{O}(N)$ entries. The number of levels L in this decomposition is on the order of $\log(N)$, where N is the dimension of K. This factorization is obtained by constructing interpolation decompositions of the low rank blocks of K using the algorithm of the preceding section. The IDBF algorithm takes as input the parameter r_k which is an estimate of the maximum possible ranks of the low-rank blocks.

Given an $N \times N$ matrix K, or equivalently an $\mathcal{O}(1)$ algorithm to evaluate an arbitrary entry of K, the algorithm of [15] constructs this data-sparse representation of K in $\mathcal{O}\left(N \log^{\kappa+1}(N)\right)$ operations using $\mathcal{O}\left(N \log^{\kappa+1}(N)\right)$ storage. Once this factorization has been constructed, the matrix K can be applied in $\mathcal{O}\left(N \log^{\kappa+1}(N)\right)$ operations.

2.3 Low-rank approximation by randomized sampling

In this section, we discuss an existing linear complexity algorithm for constructing an approximate singular value decomposition (SVD) of a matrix.

Suppose that $A \in \mathbb{C}^{m \times n}$ has singular values

$$|\sigma_1| \geq |\sigma_2| \geq \cdots \geq |\sigma_l|, \tag{18}$$

where $l = \min(n, m)$. A rank-r singular value decomposition of A is a factorization of the form

$$A \approx U \Sigma V^T, \tag{19}$$

where $U \in \mathbb{C}^{m \times r}$ is orthogonal, $\Sigma \in \mathbb{R}^{r \times r}$ is diagonal, $V \in \mathbb{C}^{n \times r}$ is orthogonal and

$$\left\| A - U \Sigma V^T \right\|_2 = \sigma_{k+1}. \tag{20}$$

The construction of a factorization of this form is a notoriously expensive calculation. However, using randomized algorithms, approximate SVDs of the same form with slightly lower accuracy can

be rapidly constructed. That is, randomized algorithms result in factorizations of the form (19) for which

$$\left\|A - U\Sigma V^T\right\|_2 \tag{21}$$

is no longer equal to the optimal value σ_{k+1}, but is instead slightly larger.

One of the first practical randomized algorithms for constructing approximate SVDs was proposed in [7]. It operates by applying a random transform to the matrix A and requires $\mathcal{O}(nmk)$ operations. By using a special random transform which can be applied rapidly via the FFT, a variant of the algorithm of [7] which requires $\mathcal{O}(nm\log(k))$ can be obtained.

In [6], a method which operates by randomly sampling $\mathcal{O}(1)$ rows and columns of the input matrix is described. It only requires $\mathcal{O}(m+n)$ operations and $\mathcal{O}(m+n)$ storage. Here, we denote this algorithm as Function `randomizedSVD` and it is presented in Algorithm 1. Assuming the whole low-rank matrix A is known, the input of Function `randomizedSVD` is A, $\mathcal{O}(1)$ randomly sampled row indices $\mathcal{R}$ and column indices $\mathcal{C}$, as well the parameter r. Equivalently, it can also be assumed that $A(\mathcal{R},:)$ and $A(:,\mathcal{C})$ are known as the inputs. The outputs are the matrices $U \in \mathbb{C}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$, and $V \in \mathbb{C}^{n\times r}$ which give an approximate SVD (19).

In Function `randomizedSVD`, for simplicity, given any matrix $K \in \mathbb{C}^{s\times t}$, Function `qr(K)` performs a pivoted QR decomposition $K(:,P) = QR$, where P is a permutation vector of the t columns, Q is a unitary matrix, and R is an upper triangular matrix with positive diagonal entries in decreasing order. Function `randperm(m,r)` denotes an algorithm that randomly selects r different samples in the set $\{1,2,\ldots,m\}$.

In most cases, to obtain higher accuracy, we add an oversampling parameter q and we sample rq rows and columns and only generate a rank r truncated SVD in the penultimate line in Algorithm 1. Larger q results in better stability of Algorithm 1. In our numerical experiments, Algorithm 1 is stable with high probability for $q >= 2$, and $q = 2$ is empirically sufficient to achieve accurate low-rank approximations.

Function $[U,\Sigma,V] \leftarrow$ `randomizedSVD(`$A,\mathcal{R},\mathcal{C},r$`)`

- $[m,n] \leftarrow \mathrm{size}(A)$
- $P \leftarrow$ `qr(`$A(\mathcal{R},:)$`)` ; $\Pi_{col} \leftarrow P(1:r)$ // $A(\mathcal{R},P) = QR$
- $P \leftarrow$ `qr(`$A(:,\mathcal{C})^T$`)` ; $\Pi_{row} \leftarrow P(1:r)$ // $A(P,\mathcal{C}) = R^TQ^T$
- $Q \leftarrow$ `qr(`$A(:,\Pi_{col})$`)` ; $Q_{col} \leftarrow Q(:,1:r)$ // $A(P,\Pi_{col}) = QR$
- $Q \leftarrow$ `qr(`$A(\Pi_{row},:)^T$`)` ; $Q_{row} \leftarrow Q(:,1:r)$ // $A(\Pi_{row},P) = R^TQ^T$
- $S_{row} \leftarrow$ `randperm(`m,r`)` ; $I \leftarrow [\Pi_{row}, S_{row}]$
- $S_{col} \leftarrow$ `randperm(`n,r`)` ; $J \leftarrow [\Pi_{col}, S_{col}]$
- $M \leftarrow (Q_{col}(I,:))^\dagger A(I,J)\left(Q_{row}^T(:,J)\right)^\dagger$ // $(\cdot)^\dagger$: `pseudo-inverse`
- $[U_M,\Sigma_M,V_M] \leftarrow \mathrm{svd}(M)$
- $U \leftarrow Q_{col}U_M$; $\Sigma \leftarrow \Sigma_M$; $V \leftarrow Q_{row}V_M$

Algorithm 1: Randomized sampling for a rank-r approximate SVD with $\mathcal{O}(m+n)$ operations, such that $A \approx U\Sigma V^T$.

3 Algorithm for the application of the ALT

The principal content of this section is a description of our block partitioning algorithm based on IDBF and low-rank approximation by randomized sampling for applying the forward ALT. Before we present this, however, we briefly discuss certain background information regarding the

associated Legendre functions and the associated Legendre transform which we exploit. As we observe in Section 3.1.1, the inverse ALT can be applied in essentially the same fashion as the forward ALT.

3.1 Background

3.1.1 The relationship between the forward and inverse associated Legendre transforms

For fixed N and $|m| \leq N$, the forward ALT consists of computing the values of the sum

$$g(m,\theta) = \sum_{k=|m|}^{2N-1} \beta_{k,m} \overline{P}_k^{|m|}(\cos(\theta)), \tag{22}$$

at the nodes of the $2N$-point Gauss-Legendre quadrature rule. We let

$$x_0 = \cos(\theta_0), x_1 = \cos(\theta_1), \ldots, x_{2N-1} = \cos(\theta_{2N-1})$$

and

$$w_0, w_1, \ldots, w_{2N-1}$$

denote the nodes and weights of this quadrature. There are $(2N-|m|)$ coefficients in the expansion (22) and $2N$ target points, so this amounts to applying the $2N \times (2N-|m|)$ matrix whose ij entry is

$$\overline{P}_j^{|m|}(\cos(\theta_i))$$

to the vector

$$\begin{pmatrix} \beta_{|m|,m} \\ \beta_{|m|+1,m} \\ \vdots \\ \beta_{2N-1,m} \end{pmatrix}. \tag{23}$$

of coefficients.

It is well-known that for $k \geq |m|$, $\overline{P}_k^{|m|}(x)$ is a polynomial of degree $k-|m|$, and that the functions

$$\left\{ \overline{P}_k^{|m|}(x) : k = |m|, \ldots, 2N-1 \right\} \tag{24}$$

form an orthonormal basis in the space of polynomials of degree no larger than $2N-1$. The $2N$-point Gauss-Legendre quadrature rule exactly integrates the product of any two polynomials of degree $2N-1$. In particular, it follows that when the $(2N-|m|) \times 2N$ matrix whose ij entry is

$$\overline{P}_i^{|m|}(\cos(\theta_j))w_j$$

is applied to the vector

$$\begin{pmatrix} g(m,\theta_0) \\ g(m,\theta_1) \\ \vdots \\ g(m,\theta_{2N-1}) \end{pmatrix}$$

the result is the vector of coefficients (23). In other words, due to the orthonormality of the associated Legendre polynomials and the method used to discretize spherical harmonic transforms,

the matrix B which discretizes the inverse ALT is related to the matrix A discretizing the forward ALT via the formula

$$B = A^T W, \tag{25}$$

where W is a diagonal matrix. The methodology described in this section for applying the forward ALT can be easily used to apply its transpose, and hence also the inverse ALT.

3.1.2 Odd and even Legendre transform matrices

It is well-known (see, for instance, Chapter 14 of [4]) that $\overline{P}_k^m(x)$ is odd when $k - |m|$ is odd, and even when $k - |m|$ is even. This, together with the fact that the Gauss-Legendre quadrature nodes are symmetric around 0, allows us to reduce the cost of applying the forward ALT by a factor of 2.

More explicitly, the sum (22) can be rewritten as

$$g(\theta, m) = g_1(\theta, m) + g_2(\theta, m), \tag{26}$$

where g_1 and g_2 are defined via the formulas

$$g_1(\theta, n) = \sum_{0 \leq k \leq 2N-|m|-1,\, k \text{ is odd}} \beta_{k+|m|,m} \overline{P}_{k+|m|}^{|m|}(\cos(\theta)) \tag{27}$$

and

$$g_2(\theta, n) = \sum_{0 \leq k \leq 2N-|m|-1,\, k \text{ is even}} \beta_{k+|m|,m} \overline{P}_{k+|m|}^{|m|}(\cos(\theta)). \tag{28}$$

Because of the symmetry of the Gauss-Legendre nodes, we have

$$g_1(\theta_l, m) = -g_1(\theta_{2N-1-l}, m) \quad \text{and} \quad g_2(\theta_l, m) = g_2(\theta_{2N-1-l}, m) \tag{29}$$

for $l = 0, 1, \ldots, 2N-1$. Therefore, we can reduce the cost of applying the forward ALT by only computing the values of (22) and (23) at the nodes $\theta_0, \theta_1, \ldots, \theta_{N-1}$ and using these to compute $g(m, \theta)$ at each of the Gauss-Legendre nodes.

Computing the sum (27) at each of the N positive Gauss-Legendre nodes amounts to applying an $N \times \left(N - \lceil \frac{|m|}{2} \rceil\right)$ matrix, which we refer to as the odd ALT matrix. Computing (27) at each of the N positive Gauss-Legendre nodes amounts to applying an $N \times \left(N - \lfloor \frac{|m|}{2} \rfloor\right)$, which we refer to as the even ALT matrix.

3.2 A block partitioning scheme

When $|m| > 0$, the associated Legendre function $\overline{P}_k^{|m|}(\cos(\theta))$ has a single turning point (or the first inflection point) on the interval $(0, \pi/2)$. Its location is given by the formula

$$t_{k,m}^* = \arcsin\left(\frac{\sqrt{m^2 - 1/4}}{k + 1/2}\right). \tag{30}$$

See, for instance, Chapter 14 of [4] for details. On the interval $(0, t_{k,m}^*)$, $\overline{P}_k^{|m|}(\cos(\theta))$ is nonoscillatory and on $(t_{k,m}^*, \pi/2)$ it is oscillatory. We can view (30) as defining a piecewise continuous curve that divides the odd and even ALT matrices into oscillatory and nonoscillatory regions. We will refer to this as the "turning point curve". Any subblock of these matrices that intersects this curve will have a high rank. As a consequence of this, the even and odd ALT matrices do not have the

complementary low-rank property. Figure 2 (a) shows an example of an odd ALT matrix with a graph of this piecewise continuous curve overlaid on top of it.

We use the following procedure to hierarchically partition the even and odd and ALT matrices into blocks each of which will either be of small dimension or will have complementary low-rank property. We will denote the resulting collection of subblocks by $\mathcal{B}_s$. Since the shape of the matrices are not square when $m \neq 0$, we initially take $\mathcal{B}_s$ to consist of blocks each of which consist of all columns of the matrix and b rows, where

$$b = \begin{cases} \lfloor \frac{N}{N - \lceil \frac{|m|}{2} \rceil} \rceil & \text{for odd matrices,} \\ \lfloor \frac{N}{N - \lfloor \frac{|m|}{2} \rfloor} \rceil & \text{for even matrices.} \end{cases} \tag{31}$$

Each of these blocks is nearly square. The symbol $\lfloor x \rceil$ means the nearest integer to x. Next, we repeatedly apply the following procedure. We split each block in $\mathcal{B}_s$ which intersects the piecewise curve defined by (30) into a 2×2 grid of subblocks. We stop this procedure only once all blocks which contain turning points have either fewer than n_0 rows or columns, where n_0 is a specified parameter. This makes the maximum partition level is $L = \log\left(\frac{N}{n_0}\right)$. For each partition level ℓ of $\mathcal{B}_s$, the turning point curve intersects no more than $2^\ell - 1$ submatrices. Therefore, this procedure takes at most $\mathcal{O}(N)$ operations to partition the odd and the even matrices into submatrices, since

$$\mathcal{O}\left(\sum_{\ell=1}^{L}\left(2^\ell - 1\right)\right) \sim \mathcal{O}\left(\frac{2N}{n_0} - \log\left(\frac{N}{n_0}\right) - 2\right) \sim \mathcal{O}(N). \tag{32}$$

At level ℓ, the sub-matrix is of size $\mathcal{O}(\frac{N}{2^\ell}) \times \mathcal{O}(\frac{N}{2^\ell})$ and there are $\mathcal{O}(2^\ell)$ such sub-matrices either as an oscillatory block or a non-oscillatory block. See Figure 2 (b), which shows an example of an odd ALT matrix that has been partitioned into blocks by this procedure.

Finally, we estimate the cost for applying matrix blocks that intersect with the turning point curve. The maximum partition level is $L = \log\left(\frac{N}{n_0}\right)$ and the turning point curve intersects with no more than $2^\ell - 1$ submatrices for each partition level ℓ. Each box has size at most $n_0 \times n_0$ implying that the cost of applying all these blocks is

$$\mathcal{O}\left(n_0^2 L\left(2^L - 1\right)\right) = \mathcal{O}(N \log(N)). \tag{33}$$

3.3 Factorization and application of matrix blocks

In each of the partitioned matrices, there are three types of blocks: oscillatory blocks, non-oscillatory blocks, and the blocks which intersect the turning point curve. We deal with each of these different kinds of blocks by different approaches. In the following discussion, we assume that, for a fixed dimension N of the ALT, the ranks of all low rank matrices are bounded. We denote the least upper bound by $r(N)$.

1. For an oscillatory block $\mathcal{B}^o$ with the size $N_0 \times N_0$, we use the IDBF to construct a factorization

$$\mathcal{B}^o \approx U^L U^{L-1} \cdots U^h S^h V^h \cdots V^{L-1} V^L, \tag{34}$$

where $L = \mathcal{O}(\log(N_0))$ and $h = \frac{L}{2}$. This takes only $\mathcal{O}\left((r(N))^2 N_0 \log(N_0)\right)$ operations and memory. After factorization, we can apply the subblock B^o with $\mathcal{O}\left((r(N))^2 N_0 \log(N_0)\right)$ operations and memory.

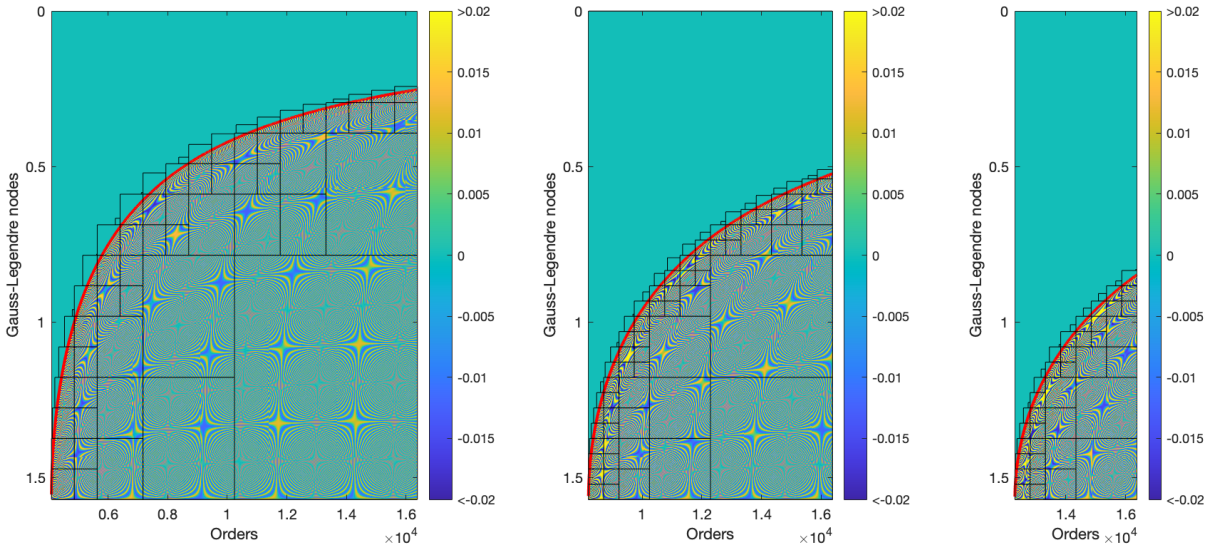
2. In the non-oscillatory region, the entries of the odd and even ALT matrices can be of extremely small magnitudes. Therefore, when processing a non-oscillatory block $\mathcal{B}^n$ with the size $N_0 \times N_0$, we first take the largest subblock $\mathcal{B}^{n'}$ which does not contain any elements of magnitude smaller than machine precision. Next, we use the algorithm of Section 2.3 to construct a low rank factorization of the form

$$\mathcal{B}^{n'} \approx U \Sigma V^T \tag{35}$$

with $\mathcal{O}\left((r(N))^2 N_0\right)$ operations and $\mathcal{O}\left(r(N) N_0\right)$ memory. After factorization, the application of $\mathcal{B}^n$ requires $\mathcal{O}\left(r(N) N_0\right)$ operations and memory.

3. For a block $\mathcal{B}^t$ including turning points, we also let $\mathcal{B}^{t'}$ be a smaller submatrix which excludes as many entries whose magnitudes are smaller than machine precision as possible. These blocks are applied to a vector through a standard matrix-vector multiplication with no made attempt to accelerate it. The operation and memory complexity for these blocks are bounded by the number of nonzero entries in these blocks, which $\mathcal{O}(N \log N)$ for an ALT matrix of size $N \times N$ according to (33).

Figure 6 shows the boxes which result after as many elements of negligible magnitude as possible have been excluded. Empty blocks with 0×0 size are omitted in the figure and will not be utilized in the application step.



(a) $m = 4096$ (b) $m = 8192$ (c) $m = 12288$

Figure 6: A visualization of the partitioning procedure for the forward ALT matrix. In each case, $N = 8192$. From left to right, the orders of the transform are $m = 4096$, $m = 8192$ and $m = 12288$.

4 Computational complexity

A rigorous estimate of the computational complexity of our algorithm would seem to require a bound on the ranks of the subblocks of the odd and even ALT matrices which are in the oscillatory regions. To the author's knowledge, no such bounds are presently known, except in the special case $m = 0$ (such an estimate can be found in [24]). We can, however, develop an estimate on the

complexity of our algorithm in terms of both operation and memory assuming that the ranks of these boxes are bounded by a quantity depending on N, which we denote by $r(N)$.

We will assume that the matrix we are applying is an $N \times N$ odd ALT matrix which we will denote by A. The analysis for the even case is similar. We first observe that subdividing a matrix with the complementary low-rank property into subblocks and using the IDBF to apply each subblock separately has essentially the same asymptotic complexity as using the IDBF to apply the entire matrix. Recall that there are $L = \log\left(\frac{N}{n_0}\right)$ levels of subdivision. At level ℓ, the sub-matrix is of size $\mathcal{O}(\frac{N}{2^\ell}) \times \mathcal{O}(\frac{N}{2^\ell})$ and there are $\mathcal{O}(2^\ell)$ such sub-matrices as an oscillatory block for IDBF. Hence, the total factorization and application complexity in terms of both operation and memory is bounded by:

$$\sum_{\ell=1}^{\log\left(\frac{N}{n_0}\right)} \mathcal{O}(2^\ell)((r(N))^2 \mathcal{O}\left(\frac{N}{2^\ell}\log\left(\frac{N}{2^\ell}\right)\right) = \mathcal{O}\left((r(N))^2 N \log^2 N\right).$$

The complexity analysis is similar for non-oscillatory blocks. The numbers of blocks of different sizes are the same as those of oscillatory blocks. The factorization and application complexity in terms of operation and memory of each non-oscillatory block is cheaper than that of an oscillatory block. Therefore, the total cost for non-oscillatory blocks has a scaling smaller than that of oscillatory blocks. In fact, due to the very rapid decay of the associated Legendre functions as one moves away from the turning point into the nonoscillatory regime (see Figure 6), many non-oscillatory blocks are almost zero and, hence, we neglect their computation.

Based on extensive numerical experiments, some of which are presented in the following section, we conjecture that $(r(N))^2$ grows as $\mathcal{O}(\log(N))$. Of course, if this is correct, then our algorithm for applying the ALT requires $\mathcal{O}\left(N\log^3(N)\right)$ operations and can be used to apply the SHT in $\mathcal{O}\left(N^2\log^3(N)\right)$ operations.

We summarize the situation with the following theorem and conjecture:

Theorem 4.1. *The time and memory complexities of the algorithm of Section 3 for applying an odd or even ALT matrix of size $N \times N$ to a vector are*

$$\mathcal{O}\left((r(N))^2 N \log^2 N\right),$$

where $r(N)$ is the least upper bound for both the ranks of the non-oscillatory blocks which do not intersect the turning point curve and the ranks of the subblocks of the oscillatory portion of the matrix.

Conjecture 4.2. *The quantity $(r(N))^2$ grows as $\mathcal{O}(\log(N))$.*

5 Numerical results

This section presents several numerical experiments which demonstrate the efficiency of the proposed algorithm. Our code was written in MATLAB® and executed on a single 3.2GHz core. It is available as a part of the ButterflyLab package (`https://github.com/ButterflyLab/ButterflyLab`).

For the forward ALT, given an order m, we let $g^d(\theta)$ denote the results given by applying the discretized operator directly using a standard matrix-vector multiplication, and we let $g^b(\theta)$ denote the results obtained via the proposed algorithm. The accuracy of our method is estimated via the relative error defined by

$$\epsilon^{fwd} = \sqrt{\frac{\sum_{\theta\in S_1} |g^b(\theta) - g^d(\theta)|^2}{\sum_{\theta\in S_1} |g^d(\theta)|^2}}, \tag{36}$$

where S_1 is an index set containing 256 randomly sampled row indices of the non-zero part in the odd matrix or the even matrix.

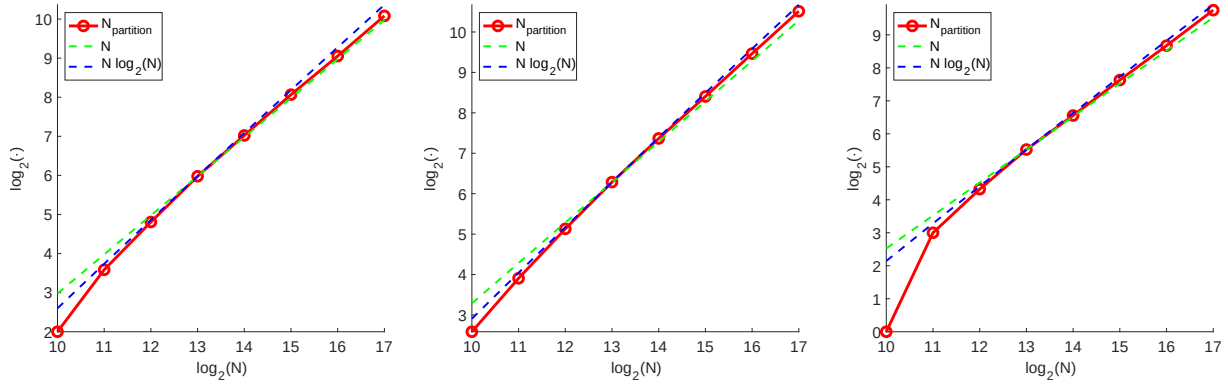
We use $a^d(k)$ and $a^b(k)$ denote the results obtained by applying the inverse ALT using a standard matrix-vector multiplication and via our algorithm, respectively. The definition of the error ϵ^{inv} in this case is

$$\epsilon^{inv} = \sqrt{\frac{\sum_{k\in S_2} |a^b(k) - a^d(k)|^2}{\sum_{k\in S_2} |a^d(k)|^2}}, \tag{37}$$

where S_2 is an index set containing 256 randomly sampled row indices of the odd matrix or the even matrix.

In all of our examples, the tolerance parameter ϵ for interpolative decompositions is set to 10^{-10}, the minimum length n_0 for the partitioned block is set to 512, and the rank parameter r for randomized SVD in low-rank phase matrix factorization is set to 30.

Number of the blocks: Our first experiment consists of counting the number of blocks which remain after those which contain no non-negligible elements are discarded. Figure 7 visualizes the results of this experiment for different N and with m set to be $0.5N$, N and $1.5N$. We observe that the number of remaining blocks scales nearly linearly as the problem size increases.



(a) $m = 0.5N$ (b) $m = N$ (c) $m = 1.5N$

Figure 7: Plots of the number of the remaining blocks a a function of N for $m = 0.5N$, $m = N$, and $1.5N$.

Selection of Mock-Chebyshev points or randomly selected points: Next, we compare the results of using Mock-Chebyshev points and randomly selected points for evaluating IDs in the IDBF process. The results are shown in Figure 8. In these experiments, the order parameter m is set to be equal to N and the adaptive rank r_k for IDBF is set to be 50, 100 or 150. We observe that the accuracy of results increases as the rank parameter r_k increases, and that the accuracy of IDs performed with Mock-Chebyshev points is higher than that of the IDs performed with randomly selected points. Moreover, we conclude that letting $r_k = 150$ suffices to achieve high-accuracy with Mock-Chebyshev points.

Thus, for the rest of experiments, we will use Mock-Chebyshev points as grids to compute IDs in the IDBF algorithm, and the adaptive rank r_k for IDBF will be fixed at 150.

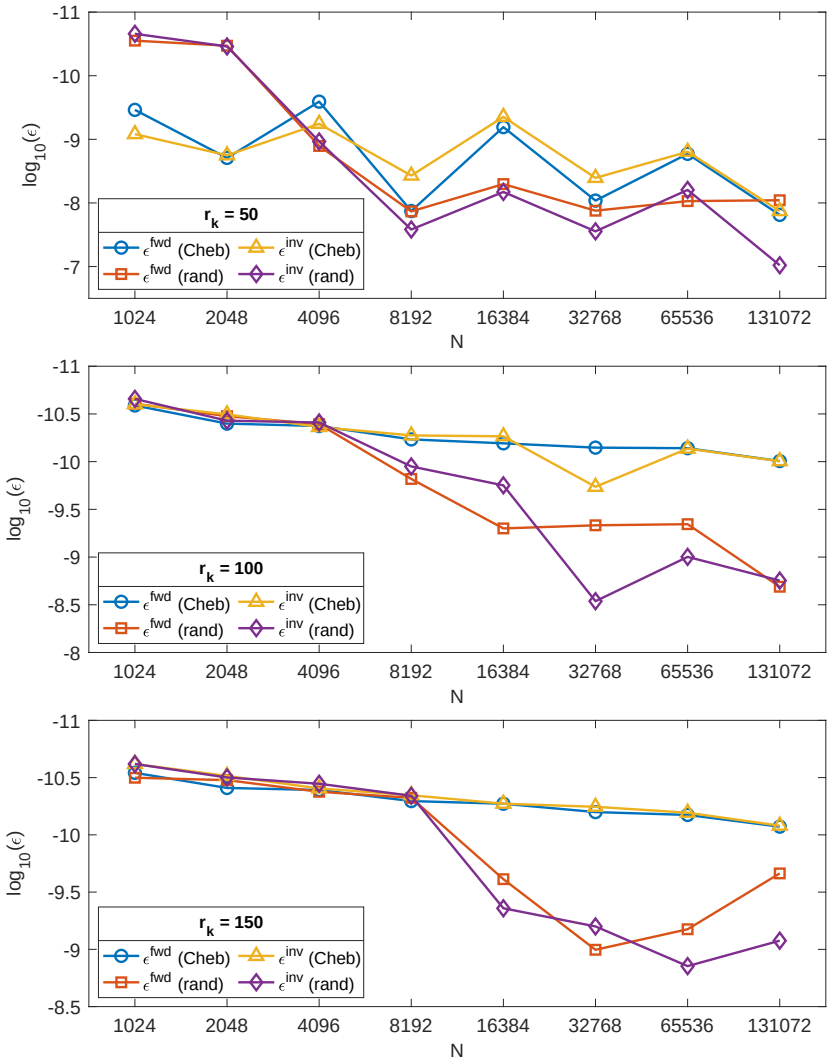


Figure 8: The results of experiments comparing the error in applying the associated Legendre transform when different grids of points are used to form interpolative decompositions in the IDBF algorithm. Here, N is the size of the matrix, and the order m is set to be N in each case. The adaptive rank r_k for IDBF is set to be 50, 100 and 150 from the top panel to the bottom panel. "Cheb" and "rand" represent IDs with Mock-Chebyshev points and randomly selected points, respectively.

Associated Legendre transforms of different orders: In these experiments, we measured the accuracy and efficiency of the proposed algorithm for various orders of m.

Figure 9 shows that the accuracy of the proposed algorithm is unaffected by the order m of the ALT, even though the accuracy decays slightly as the problem size increases. The slightly increasing error appears to be due to the randomness of the proposed algorithm in Subsection 2.3. As the problem size increases, the probability of capturing the low-rank matrix with a fixed rank parameter becomes slightly smaller.

Figure 10 visualizes the computational complexity of the factorizing and applying the forward and inverse ALT matrices. There, T_{fac}^{fwd} and T_{app}^{fwd} are the factorization time and the application time of the proposed algorithm for the forward ALT, respectively. And T_{mat}^{fwd} and T_{dir}^{fwd} are the time

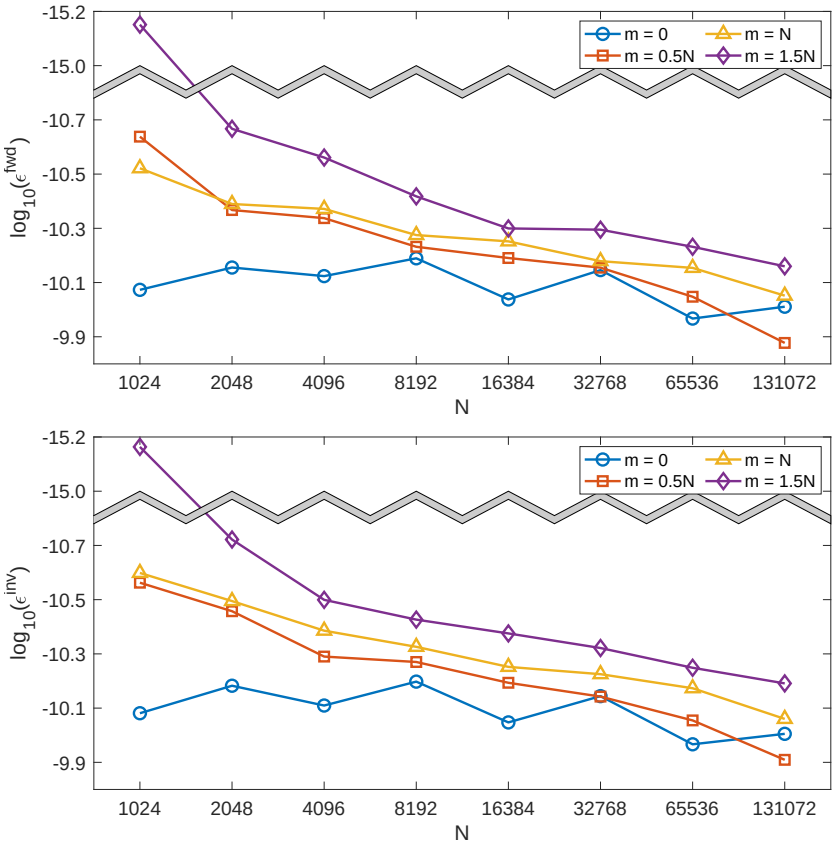


Figure 9: The errors in the application of associated Legendre transforms of different orders. The top panel shows the error ϵ^{fwd} of the forward ALT, and the bottom panel shows that error ϵ^{inv} of the inverse ALT. Here, N is the size of the matrix, and the order m was set to be 0, $0.5N$, N or $1.5N$. The adaptive rank r_k for IDBF was taken to be 150.

for constructing the normalized associated Legendre matrix and performing the matrix application directly. The definitions of T^{inv}_{fac}, T^{inv}_{app}, T^{inv}_{mat}, and T^{inv}_{dir} for the inverse ALT are analogous. We observe that the running times of these processes scale nearly linearly with the problem size.

Figure 11 compares the factorization time and the application time of the proposed algorithm with the brute force approach to applying the ALT (that is, direct application of the matrix discretizing the ALT). We observe a significant improvement at larger problem sizes.

6 Conclusion and future work

This paper introduces an algorithm for the application of the forward and inverse associated Legendre transforms. Experimental results suggest that its total running time, including both an application and a precomputation phase, is $\mathcal{O}\left(N \log^3(N)\right)$. Using this algorithm, the forward and inverse spherical harmonic transforms can be applied in $\mathcal{O}\left(N^2 \log^3(N)\right)$ time, assuming our conjecture regarding the running time of our algorithm is correct.

The blocked IDBF algorithm used here is extremely dependent on the method used to form interpolative decompositions. The most efficient and accurate method for forming such factorizations is still an ongoing topic of research, and the authors plan to develop improved versions of

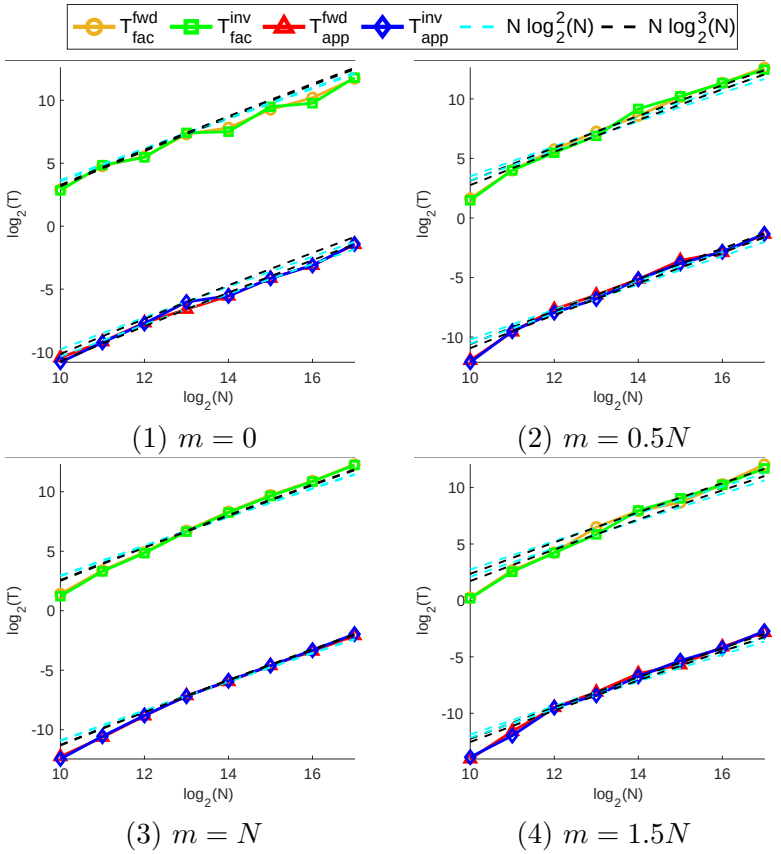


Figure 10: The computational complexity of the ALT for different orders m. Here, N is the size of the matrix, and order m is set to be 0, $0.5N$, N or $1.5N$. "Fac" and "App" represent the factorization time and the application time, respectively. All times are in seconds.

their algorithm which incorporate new developments.

Moreover, the authors are actively working on developing a rigorous bound on the ranks of blocks of the forward and inverse ALT matrices. Such a bound would enable a rigorous complexity estimate for the spherical harmonic transform.

Acknowledgments. The authors are grateful to the anonymous reviewers for their many helpful comments. The authors also thank Yingzhou Li for his discussion on block partitioning the oscillatory region of associated Legendre transform. J.B. was supported in part by NSF grants DMS-1418723 and DMS-2012487. Z. C. was partially supported by the Ministry of Education in Singapore under the grant MOE2018-T2-2-147. H. Y. was partially supported by NSF under the grant award DMS-1945029.

References

[1] J. P. Boyd and F. Xu. Divergence (Runge Phenomenon) for least-squares polynomial approximation on an equispaced grid and Mock Chebyshev subset interpolation. *Applied Mathematics and Computation*, 210(1):158 – 168, 2009.

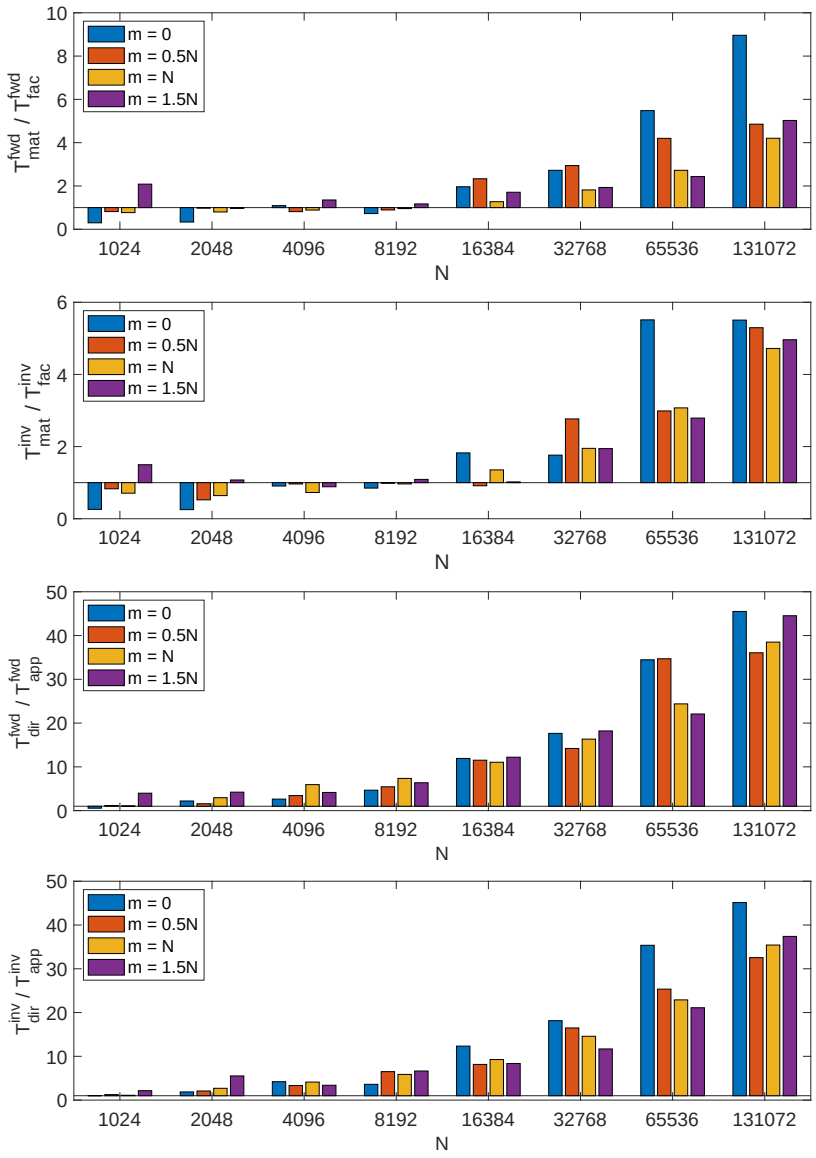


Figure 11: A comparison of the speed of the proposed algorithm for the ALT with the brute force approach. Here, N is the size of the matrix, and order m is 0, $0.5N$, N or $1.5N$. The adaptive rank r_k for IDBF is set to be 150. From the top to bottom, the charts give the ratios $T^{fwd}_{mat}/T^{fwd}_{fac}$, $T^{inv}_{mat}/T^{inv}_{fac}$, $T^{fwd}_{dir}/T^{fwd}_{app}$ and $T^{inv}_{dir}/T^{inv}_{app}$ are shown.

[2] J. Bremer. An algorithm for the numerical evaluation of the associated Legendre functions that runs in time independent of degree and order. *Journal of Computational Physics*, 360:15 – 38, 2018.

[3] Z. Chen, J. Zhang, K. L. Ho, and H. Yang. Multidimensional phase recovery and interpolative decomposition butterfly factorization. *Journal of Computational Physics*, 412:109427, 2020.

[4] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.22 of 2019-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[5] J. Driscoll and D. Healy. Computing Fourier transforms and convolutions on the 2-sphere. *Advances in Applied Mathematics*, 15(2):202 – 250, 1994.

[6] B. Engquist and L. Ying. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun. Math. Sci.*, 7(2):327–345, 2009.

[7] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[8] P. Hoffman and K. Reddy. Numerical Differentiation by High Order Interpolation. *SIAM Journal on Scientific and Statistical Computing*, 8(6):979–987, 1987.

[9] Y. Li and H. Yang. Interpolative butterfly factorization. *SIAM Journal on Scientific Computing*, 39(2):A503–A531, 2017.

[10] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying. Butterfly Factorization. *Multiscale Modeling & Simulation*, 13(2):714–732, 2015.

[11] Y. Liu, X. Xing, H. Guo, E. Michielssen, P. Ghysels, and X. S. Li. Butterfly factorization via randomized matrix-vector multiplications. *arXiv:2002.03400 [math.NA]*, 2020.

[12] E. Michielssen and A. Boag. A multilevel matrix decomposition algorithm for analyzing scattering from large structures. *Antennas and Propagation, IEEE Transactions on*, 44(8):1086–1093, Aug 1996.

[13] M. J. Mohlenkamp. A fast transform for spherical harmonics. *The Journal of Fourier Analysis and Applications*, 5(2):159–184, 1999.

[14] M. O'Neil, F. Woolfe, and V. Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. Anal.*, 28(2):203–226, 2010.

[15] Q. Pang, K. L. Ho, and H. Yang. Interpolative Decomposition Butterfly Factorization. *SIAM Journal on Scientific Computing*, 42(2):A1097–A1115, 2020.

[16] V. Rokhlin and M. Tygert. Fast Algorithms for Spherical Harmonic Expansions. *SIAM J. Sci. Comput.*, 27(6):1903–1928, Dec. 2005.

[17] D. S. Seljebotn. WAVEMOTH-FAST SPHERICAL HARMONIC TRANSFORMS BY BUTTERFLY MATRIX COMPRESSION. *The Astrophysical Journal Supplement Series*, 199(1):5, feb 2012.

[18] R. M. Slevinsky. Fast and backward stable transforms between spherical harmonic expansions and bivariate Fourier series. *Applied and Computational Harmonic Analysis*, 47(3):585–606, 2019.

[19] M. Tygert. Fast algorithms for spherical harmonic expansions, II. *Journal of Computational Physics*, 227(8):4260–4279, 2008.

[20] M. Tygert. Fast algorithms for spherical harmonic expansions, III. *Journal of Computational Physics*, 229(18):6181 – 6192, 2010.

[21] N. P. Wedi, M. Hamrud, and G. Mozdzynski. A Fast Spherical Harmonics Transform for Global NWP and Climate Models. *Monthly Weather Review*, 141(10):3450–3461, 2013.

[22] H. Yang. A unified framework for oscillatory integral transforms: When to use NUFFT or butterfly factorization? *Journal of Computational Physics*, 388:103–122, Jul 2019.

[23] F. Yin, G. Wu, J. Wu, J. Zhao, and J. Song. Performance Evaluation of the Fast Spherical Harmonic Transform Algorithm in the YinHe Global Spectral Model. *Monthly Weather Review*, 146(10):3163–3182, 2018.

[24] F. Yin, J. Wu, J. Song, and J. Yang. A High Accurate and Stable Legendre Transform Based on Block Partitioning and Butterfly Algorithm for NWP. *Mathematics*, 7, 10 2019.