Fast Algorithms for the Multi-dimensional Jacobi Polynomial Transform

James Bremer
Department of Mathematics
University of California, Davis, CA, USA
bremer@math.ucdavis.edu

Qiyuan Pang
Department of Mathematics
Tsinghua University, China
ppangqqyz@foxmail.com

Haizhao Yang
Department of Mathematics
National University of Singapore, Singapore
haizhao@nus.edu.sg

January 23, 2019

Abstract

We use the well-known observation that the solutions of Jacobi's differential equation can be represented via non-oscillatory phase and amplitude functions to develop a fast algorithm for computing multi-dimensional Jacobi polynomial transforms. More explicitly, it follows from this observation that the matrix corresponding to the discrete Jacobi transform is the Hadamard product of a numerically low-rank matrix and a multi-dimensional discrete Fourier transform (DFT) matrix. The application of the Hadamard product can be carried out via $O(1)$ fast Fourier transforms (FFTs), resulting in a nearly optimal algorithm to compute the multidimensional Jacobi polynomial transform.

Keywords. Multi-dimensional Jacobi polynomial transform, nonuniform transforms, non-oscillatory phase function, randomized low-rank approximation, fast Fourier transform.

1 Introduction

The one-dimensional forward discrete Jacobi transform consists of evaluating an expansion of the form

$$f(x) = \sum_{\nu=1}^{n} \alpha_\nu P_{\nu-1}^{(a,b)}(x), \tag{1}$$

where $P_\nu^{(a,b)}$ denotes the order-ν Jacobi polynomial of the first kind corresponding to parameters a and b, at a collection of points $\{x_i\}_{i=1,\dots,n} \subset (-1,1)$. The one-dimensional inverse discrete Jacobi transform is the process of computing the coefficients $\{\alpha_\nu\}_{i=1,\dots,n}$ in an expansion of the form (1) given its values at a collection of distinct points $\{x_i\}_{i=1,\dots,n} \subset (-1,1)$. For the sake of brevity, we will generally drop the adjective "discrete" and simply use the terms "forward Jacobi transform" and "inverse Jacobi transform" when referring to these operations. Often, the points $\{x_i\}_{i=1,\dots,n} \subset (-1,1)$ are the nodes of the n-point Gauss-Jacobi quadrature rule

$$\int_{-1}^{1} p(x)(1+x)^a(1+x)^b dx \approx \sum_{\nu=1}^{n} p(x_\nu)\omega_\nu \tag{2}$$

that is exact when p is a polynomial of degree less than or equal to $2n-1$. However, it is useful to consider more general sets of points as well. In the case in which the points $\{x_i\}_{i=1,\ldots,n} \subset$ are the nodes of the Gauss-Jacobi quadrature rule, we say the corresponding transform and inverse transform are uniform; otherwise, we describe them as nonuniform. To form an orthonormal matrix to represent the transform for numerical purpose, we will rescale the transform in (1) with quadrature weights later.

Many methods for rapidly applying the Jacobi transform and various special cases of the Jacobi transform have been developed. Examples include the algorithms of [1, 2, 3, 4] for applying the Legendre transform, and those of [5] for forming and evaluating expansions for Jacobi polynomials in general. See also, [6] and its references for extensive information on numerical algorithms for forming and manipulating Chebyshev expansions. Almost all such algorithms can be placed into one of two categories. Algorithms in the first category, such as [7, 8, 9, 10, 11], make use of the structure of the connection matrices which take the coefficients in the expansion of a function in terms of one set of Jacobi polynomials to the coefficients in the expansion of the same function with respect to a different set of Jacobi polynomials. They typically operate by computing the Chebyshev coefficients of an expansion and then applying a connection matrix or a series of connection matrices to obtain the coefficients in the desired expansion. The computation of the Chebyshev expansion can be carried out efficiently in $O(n \log n)$ operations via the nonuniform FFT (NUFFT) [12, 13] and the application of each connection matrix can be performed in a number of operations which grows linearly or quasi-linearly via the fast multipole method (FMM) [3, 14, 15, 5] or the fast eigendecomposition of semiseparable matrices [16, 17, 5].

The second class of algorithms, of which [18] is a major example, make use of butterfly methods and similar techniques for the application of oscillatory matrices. Most algorithms in this category, including that of in [18], require precomputation with $O(n^2)$ running times. However, [19] describes an algorithm based on this approach whose total running time is $O(n \log^2(n))$, when both parameters a and b are in the interval $(-1/2, 1/2)$. It makes use of the observation that the solutions of Jacobi's differential equation can be accurately represented via non-oscillatory phase and amplitude functions to apply the Jacobi transform rapidly. That certain differential equations admit non-oscillatory phase functions has long been known [20]; a general theorem was established in [21] and a numerical method for the computation of non-oscillatory phase functions was given in [22]. In the case of Jacobi's differential equation, there exists a smooth amplitude function $M^{(a,b)}(t,\nu)$ and a smooth phase function $\psi^{(a,b)}(t,\nu)$ such that

$$\tilde{P}_\nu^{(a,b)}(t) = M^{(a,b)}(t,\nu)\cos(\psi^{(a,b)}(t,\nu)) \tag{3}$$

and

$$\tilde{Q}_\nu^{(a,b)}(t) = M^{(a,b)}(t,\nu)\sin(\psi^{(a,b)}(t,\nu)), \tag{4}$$

where $\tilde{P}_\nu^{(a,b)}$ and $\tilde{Q}_\nu^{(a,b)}$ are referred as the modified Jacobi functions of the first and second kind, respectively. They are defined in terms of the Jacobi functions of the first and second kinds $P_\nu^{(a,b)}$ and $Q_\nu^{(a,b)}$ (see [20] for definitions) via the formulas

$$\tilde{P}_\nu^{(a,b)}(t) = C_\nu^{(a,b)} P_\nu^{(a,b)}(\cos(t)) \sin\left(\frac{t}{2}\right)^{a+\frac{1}{2}} \cos\left(\frac{t}{2}\right)^{b+\frac{1}{2}} \tag{5}$$

and

$$\tilde{Q}_{\nu}^{(a,b)}(t) = C_{\nu}^{(a,b)}\ Q_{\nu}^{(a,b)}(\cos(t))\sin\left(\frac{t}{2}\right)^{a+\frac{1}{2}}\cos\left(\frac{t}{2}\right)^{b+\frac{1}{2}}, \tag{6}$$

where

$$C_{\nu}^{(a,b)} = \sqrt{(2\nu+a+b+1)\frac{\Gamma(1+\nu)\Gamma(1+\nu+a+b)}{\Gamma(1+\nu+a)\Gamma(1+\nu+b)}}. \tag{7}$$

The normalization constant is chosen to ensure the $L^2(0,\pi)$ norm of $\tilde{P}_{\nu}^{(a,b)}$ is 1 when ν is an integer. Indeed, the set $\{\tilde{P}_{j}^{(a,b)}\}_{j=0}^{\infty}$ is an orthonormal basis for $L^2(0,\pi)$. The change of variables $x=\cos(t)$ makes the singularities in phase and amplitude functions for Jacobi's differential equation more tractable. Obviously, there is no substantial difference in treating expansions of the type (1) and those of the form

$$f(t) = \sum_{\nu=1}^{n} \alpha_{\nu}\tilde{P}_{\nu-1}^{(a,b)}(t) = \sum_{\nu=1}^{n}\alpha_{\nu}M^{(a,b)}(t,\nu)\cos(\psi^{(a,b)}(t,\nu)), \tag{8}$$

and we will consider the latter form here, more specially, the latter form with scaling by quadrature weights. Moreover, we will denote by $\{t_k\}_{k=1,\dots,n}$ and $\{w_k\}_{k=1\dots,n}$ the nodes and weights of the trigonometric Gauss-Jacobi quadrature rule

$$\int_0^{\pi} f(\cos(t))\cos^{2a+1}\left(\frac{t}{2}\right)\sin^{2b+1}\left(\frac{t}{2}\right)\ dt \approx \sum_{k=1}^{n} f(\cos(t_k))\cos^{2a+1}\left(\frac{t_k}{2}\right)\sin^{2b+1}\left(\frac{t_k}{2}\right)w_k \tag{9}$$

obtained by applying the change of variables $x=\cos(t)$ to (2). Again, we refer to transforms which use the set of points $\{t_k\}_{k=1,\dots,n}$ as uniform and those which use some other set points as nonuniform.

In [19], it was observed that the second summation in (8) could be interpreted as the application of a structured matrix that is the real part of a Hadamard product of a NUFFT matrix and a numerically low-rank matrix. This leads to a method for its computation which takes quasi-linear time [13, 23, 24]. Expansions in terms of the functions of the second kind $\{\tilde{Q}_{j}^{(a,b)}\}_{j=0}^{\infty}$ can be handled similarly. This non-oscillatory representation method will be described in more detail in Section 2.2.

In this paper, we generalize the one-dimensional Jacobi polynomial transform of [19] to the multi-dimensional case. The transformation matrix in the multi-dimensional case is still the real part of a Hadamard product of a multi-dimensional NUFFT matrix and a numerically low-rank matrix in the form of tensor products. However, the numerical rank of the low-rank matrix might increase quickly in the dimension following the algorithm in [19] and hence the multi-dimensional NUFFT in [12, 13] might not be sufficiently efficient. To obtain a viable multi-dimensional Jacobi polynomial transform, we reformulate the Hadamard product into a new one, which is the Hadamard product of an over-sampled Fourier transform matrix and a numerically low-rank matrix, and propose a fast randomized SVD to achieve near optimality in the low-rank approximation. This leads to an efficient implementation of two-dimensional and three-dimensional Jacobi polynomial transforms. Besides, as will be demonstrated numerically, the new method proposed in this paper is more robust than the method in [19] and works for a and b in the regime $(-1,1)$, other than the regime $(-\frac{1}{2},\frac{1}{2})$ in [19].

The remainder of the paper is organized as follows. In Section 2, we will first briefly introduce the fast SVD via randomized sampling and the theory of non-oscillatory phase functions. In Section

3, a variant of the fast one-dimensional Jacobi polynomial transform in [19] is given; it is followed by a description of our fast multi-dimensional Jacobi polynomial transform. In Section 4, numerical results which demonstrate the efficiency of our algorithm are described. Finally, we will conclude our discussion in Section 5.

2 Preliminaries

In this section, we will revisit the linear scaling randomized SVD introduced in [25] and the non-oscillatory phase and amplitude functions in [19] to make the presentation self-contained.

2.1 Approximate SVD via Randomized Sampling

For a numerically low-rank matrix $Z \in \mathbb{C}^{N\times N}$ with an $O(1)$ algorithm to evaluate an arbitrary entry, [25] introduced an $O(Nr^2)$ algorithm to construct a rank-r approximate SVD, $Z \approx U_0\Sigma_0 V_0^*$, from $O(r)$ randomly selected rows and columns of Z, where * denotes the conjugate transpose, U_0 and $V_0 \in \mathbb{C}^{N\times r}$, $\Sigma_0 \in \mathbb{C}^{r\times r}$ is a diagonal matrix with approximate singular values in the diagonal part in a descent order.

Here, we adopt the standard notation for a submatrix in MATLAB: given a row index set I and a column index set J, $Z_{I,J} = Z(I,J)$ is the submatrix with entries from rows in I and columns in J; we also use ":" to denote the entire columns or rows of the matrix, i.e., $Z_{I,:} = Z(I,:)$ and $Z_{:,J} = Z(:,J)$. With these handy notations, we briefly introduce the randomized SVD as follows.

Algorithm 2.1. *Approximate SVD via randomized sampling*

1. *Let Π_{col} and Π_{row} denote the important columns and rows of Z that are used to form the column and row bases. Initially $\Pi_{col} = \emptyset$ and $\Pi_{row} = \emptyset$.*

2. *Randomly sample rq rows and denote their indices by S_{row}. Let $I = S_{row} \cup \Pi_{row}$. Here $q = O(1)$ is a multiplicative oversampling parameter. Perform a pivoted QR decomposition of $Z_{I,:}$ to get*

$$Z_{I,:}P = QR, \tag{10}$$

where P is the resulting permutation matrix and $R = (r_{ij})$ is an $O(r) \times n$ upper triangular matrix. Define the important column index set Π_{col} to be the first r columns picked within the pivoted QR decomposition.

3. *Randomly sample rq columns and denote their indices by S_{col}. Let $J = S_{col} \cup \Pi_{col}$. Perform a pivoted LQ decomposition of $Z_{:,J}$ to get*

$$PZ_{:,J} = LQ, \tag{11}$$

where P is the resulting permutation matrix and $L = (l_{ij})$ is an $m \times O(r)$ lower triangular matrix. Define the important row index set Π_{row} to be the first r rows picked within the pivoted LQ decomposition.

4. *Repeat steps 2 and 3 a few times to ensure Π_{col} and Π_{row} sufficiently sample the important columns and rows of Z.*

5. *Apply the pivoted QR factorization to $Z_{:,\Pi_{col}}$ and let Q_{col} be the matrix of the first r columns of the Q matrix. Similarly, apply the pivoted QR factorization to $Z^*_{\Pi_{row},:}$ and let Q_{row} be the matrix of the first r columns of the Q matrix.*

6. *We seek a middle matrix M such that $Z \approx Q_{col} M Q_{row}^*$. To solve this problem efficiently, we approximately reduce it to a least-squares problem of a smaller size. Let S_{col} and S_{row} be the index sets of a few extra randomly sampled columns and rows. Let $J = \Pi_{col} \cup S_{col}$ and $I = \Pi_{row} \cup S_{row}$. A simple least-squares solution to the problem*

$$\min_M \| Z_{I,J} (Q_{col})_{I,:} M (Q_{row}^*)_{:,J} \|, \tag{12}$$

gives $M = (Q_{col})_{I,:}^{\dagger} Z_{I,J} (Q_{row}^*)_{:,J}^{\dagger}$, where $(\cdot)^{\dagger}$ stands for the pseudo-inverse.

7. *Compute an SVD $M \approx U_M \Sigma_M V_M^*$. Then the low-rank approximation of $Z \approx U_0 S_0 V_0^*$ is given by*

$$U_0 = Q_{col} U_M; \Sigma_0 = \Sigma_M; V_0 = V_M^* Q_{row}^*. \tag{13}$$

In our numerical implementation, iterating Steps 2 and 3 twice is empirically sufficient to achieve accurate low-rank approximations via Algorithm 2.1. Similar arguments as in [26] for a randomized CUR factorization can be applied to quantify the error and success probability rigorously for Algorithm 2.1. But at this point, we only focus on the application of Algorithm 2.1 to the fast Jacobi polynomial transform without theoretical analysis.

Note that our goal in the Jacobi polynomial transform is to construct a low-rank approximation of the low-rank matrix, i.e., $Z \approx UV^*$ with U and $V \in \mathbb{C}^{N \times r}$, up to a fixed relative error ε, rather than a fixed rank. Algorithm 2.1 can also be embedded into an iterative process that gradually increases the rank parameter r to achieve the desired accuracy. We denote the rank parameter as r_ϵ when it is large enough to obtain the accuracy ϵ.

When Z origins from the discretization of a smooth function $Z(x, y)$ at the grid points $\{x_i\}_{1 \le i \le N}$ and $\{y_i\}_{1 \le i \le N}$, i.e., $Z_{ij} = Z(x_i, y_j)$, another standard method for constructing a low-rank factorization of $Z \approx UV^*$ is Lagrange interpolation at Chebyshev grids in x or y. For example, let $C_\mu := \{\mu_i\}_{1 \le i \le r}$ denote the set of r Chebyshev grids in the domain of x, $U \in \mathbb{R}^{N \times r}$ be the matrix consisting of the i-th Lagrange polynomial in x corresponding to μ_i as its i-th column, and $V \in \mathbb{C}^{N \times r}$ be the matrix such that V_{ij} is equal to $\bar{Z}(\mu_j, y_i)$, where $\bar{\cdot}$ denotes the conjugate operator, then $Z \approx UV^*$ by the Lagrange interpolation. Usually, an oversampling parameter q is used via setting $r = q r_\epsilon$. Then a rank-r_ϵ truncated SVD of $U \approx U_0 \Sigma_0 V_0^*$ gives a compressed rank-r_ϵ approximation of $Z \approx U_0 (V V_0 \Sigma_0)^*$, where $U_0 \in \mathbb{C}^{N \times r_\epsilon}$ and $V V_0 \Sigma_0 \in \mathbb{C}^{N \times r_\epsilon}$.

The fast nonuniform FFT in [13] and the one-dimensional Jacobi polynomial transform in [19] adopted low-rank approximation via Lagrange interpolation to deal with the low-rank term in their Hadamard products of low-rank and (nonuniform) FFT matrices without an extra truncated SVD. In this paper, we propose to use the randomized SVD via random sampling to obtain nearly optimal rank in the low-rank approximation.

2.2 Non-oscillatory phase and amplitude functions

Given a pair of parameters a and b in $\left(-\frac{1}{2}, \frac{1}{2}\right)$, and a maximum degree of interest $N_{\max} > 27$, we revisit the fast algorithms in [19] for constructing non-oscillatory phase and amplitude functions $\psi^{(a,b)}(t, \nu)$ and $M^{(a,b)}(t, \nu)$ such that

$$\tilde{P}_\nu^{(a,b)}(t) = M^{(a,b)}(t, \nu) \cos\left(\psi^{(a,b)}(t, \nu)\right) \tag{14}$$

and

$$\tilde{Q}_\nu^{(a,b)}(t) = M^{(a,b)}(t, \nu) \sin\left(\psi^{(a,b)}(t, \nu)\right) \tag{15}$$

for $t \in [\frac{1}{N_{\max}}, \pi - \frac{1}{N_{\max}}]$ and $\nu \in (27, N_{\max}]$. The polynomials with t and ν out of these ranges can be evaluated by the well-known three-term recurrence relations or various asymptotic expansions; the lower bound of 27 was chosen to obtain optimal numerical performance.

Next, we present a few facts regarding the phase and amplitude functions related to Jacobi's differential equations. It is well known that the functions $\tilde{P}_\nu^{(a,b)}$ and $\tilde{Q}_\nu^{(a,b)}$ satisfy the second order differential equation

$$y''(t) + q_\nu^{(a,b)}(t)y(t) = 0, \tag{16}$$

where

$$q_\nu^{(\alpha,\beta)}(t) = \left(\nu + \frac{\alpha+\beta+1}{2}\right)^2 + \frac{\frac{1}{4}-\alpha^2}{4\sin\left(\frac{t}{2}\right)^2} + \frac{\frac{1}{4}-\beta^2}{4\cos\left(\frac{t}{2}\right)^2}. \tag{17}$$

We refer to Equation (16) as Jacobi differential equation. Following the derivation in [19], we can show that the pair $\{\tilde{P}_\nu^{(a,b)}, \tilde{Q}_\nu^{(a,b)}\}$ of real-valued, linearly independent solutions of (16) satisfies

$$\tilde{P}_\nu^{(a,b)}(t) = \sqrt{W}\, \frac{\cos\left(\psi^{(a,b)}(t,\nu)\right)}{\sqrt{\left|\partial_t \psi^{(a,b)}(t,\nu)\right|}}, \tag{18}$$

and

$$\tilde{Q}_\nu^{(a,b)}(t) = \sqrt{W}\, \frac{\sin\left(\psi^{(a,b)}(t,\nu)\right)}{\sqrt{\left|\partial_t \psi^{(a,b)}(t,\nu)\right|}}, \tag{19}$$

where W is the necessarily positive constant Wronskian of the pair $\{\tilde{P}_\nu^{(a,b)}, \tilde{Q}_\nu^{(a,b)}\}$, and $\psi^{(a,b)}(t,\nu)$ is the non-oscillatory phase function of the pair $\{\tilde{P}_\nu^{(a,b)}, \tilde{Q}_\nu^{(a,b)}\}$ satisfying

$$\psi^{(a,b)}(t,\nu) = C + \int_{\sigma_1}^{t} \frac{W}{(\tilde{P}_\nu^{(a,b)}(s))^2 + (\tilde{Q}_\nu^{(a,b)}(s))^2}\, ds \tag{20}$$

with C an appropriately chosen constant related to σ_1. Note that $\partial_t \psi^{(a,b)}(t,\nu) > 0$ since $W > 0$. Hence, the non-oscillatory amplitude function of $\{\tilde{P}_\nu^{(a,b)}, \tilde{Q}_\nu^{(a,b)}\}$ can be defined as

$$M^{(a,b)}(t,\nu) = \sqrt{\frac{W}{\left|\partial_t \psi^{(a,b)}(t,\nu)\right|}} = \sqrt{\frac{W}{\partial_t \psi^{(a,b)}(t,\nu)}}. \tag{21}$$

Through straightforward computation, it can be verified that the square $N^{(a,b)}(t,\nu) = \left(M^{(a,b)}(t,\nu)\right)^2$ of the amplitude function satisfies the third order linear ordinary differential equation (ODE)

$$\partial_{ttt} N^{(a,b)}(t,\nu) + 4q_\nu^{(a,b)}(t)\partial_t N^{(a,b)}(t,\nu) + 2(q_\nu^{(a,b)}(t))' N^{(a,b)}(t,\nu) = 0 \quad \text{for all} \quad \sigma_1 < t < \sigma_2. \tag{22}$$

Obviously,

$$\left(M^{(a,b)}(t,\nu)\right)^2 = \left(P_\nu^{(a,b)}(t)\right)^2 + \left(Q_\nu^{(a,b)}(t)\right)^2. \tag{23}$$

Hence, we can specify the values of $\left(M^{(a,b)}(t,\nu)\right)^2$ and its first two derivatives in t at a point on the interval $[\frac{1}{N_{\max}}, \pi - \frac{1}{N_{\max}}]$ for any ν using various asymptotic expansion for $P_\gamma^{(a,b)}$ and $Q_\gamma^{(a,b)}$. Afterwards, we uniquely determine $M^{(a,b)}(t,\nu)$ via solving the ODE (22) using a variant of the integral equation method of [12] (or any standard method for stiff problems).

To obtain the values of $\psi^{(a,b)}(t,\nu)$, we first calculate the values of

$$\frac{d}{dt}\psi^{(a,b)}(t,\nu) \tag{24}$$

via (21). Next, we obtain the values of the function $\tilde{\psi}$ defined via

$$\tilde{\psi}(t) = \int_{\alpha_1}^{t} \frac{d}{ds}\psi^{(a,b)}(s,\nu)\; ds \tag{25}$$

at any point in $[\frac{1}{N_{\max}}, \pi - \frac{1}{N_{\max}}]$ via spectral integration. There is an unknown constant connecting $\tilde{\psi}$ with $\psi^{(a,b)}(t,\nu)$; that is,

$$\psi^{(a,b)}(t,\nu) = \tilde{\psi}(t) + C. \tag{26}$$

To evaluate C, we first use a combination of asymptotic and series expansions to calculate $\tilde{P}_\nu^{(a,b)}$ at the point α_1. Since $\widetilde{\psi}(\alpha_1) = 0$, it follows that

$$\tilde{P}_\gamma^{(a,b)}(\alpha_1) = M^{(a,b)}(\alpha_1,\gamma)\cos(C), \tag{27}$$

and C can be calculated in the obvious fashion.

The above discussion has introduced an algorithm to evaluate $\psi^{(a,b)}(t,\nu)$ and $M^{(a,b)}(t,\nu)$ in the whole domain $(t,\nu) \in [\frac{1}{N_{\max}}, \pi - \frac{1}{N_{\max}}] \times (27, N_{\max}]$. To achieve a fast algorithm, we note that it is enough to conduct calculation on selected important grid points of $[\frac{1}{N_{\max}}, \pi - \frac{1}{N_{\max}}] \times (27, N_{\max}]$ through the above calculation, since we can evaluate $\psi^{(a,b)}(t,\nu)$ and $M^{(a,b)}(t,\nu)$ via interpolation with their values on the important grid points.

It is well known that a polynomial $p(t)$ (here $p(t)$ is either $\tilde{P}_\nu^{(a,b)}(t)$ or $\tilde{Q}_\nu^{(a,b)}(t)$) can be evaluated in a numerically stable fashion using the barcyentric Chebyshev interpolation formula [27]

$$p(t) = \sum_{j=1}^{k} \frac{w_j}{t - x_j} p(x_j) \Bigg/ \sum_{j=1}^{k} \frac{w_j}{t - x_j}, \tag{28}$$

where $x_1, \dots, x_k$ are the nodes of the k-point Chebyshev grid on a sufficiently small interval (σ_1, σ_2) (such that $p(t)$ is a polynomial of degree at most $k-1$) and

$$w_j = \begin{cases} (-1)^j, & 1 < j < k; \\ (-1)^j\, \frac{1}{2}, & \text{otherwise.} \end{cases} \tag{29}$$

Hence, it is sufficient to evaluate $\psi^{(a,b)}(t,\nu)$ and $M^{(a,b)}(t,\nu)$, which lead to $\tilde{P}_\nu^{(a,b)}(t)$ and $\tilde{Q}_\nu^{(a,b)}(t)$, at a tensor product of Chebyshev grid points in t and ν, and calculate them at an arbitrary location (t,ν) via a bivariate piecewise barcyentric Chebyshev interpolation.

More explicitly, let m_ν be the least integer such that $3^{m_\nu} \geq N_{\max}$, and let m_t be equal to twice the least integer l such that $\frac{\pi}{2} 2^{-l+1} \leq \frac{1}{N_{\max}}$. Next, we define $\beta_j = \max\left\{3^{j+2}, N_{\max}\right\}$ for $j = 1, \dots, m_\nu$, $\alpha_i = \frac{\pi}{2} 2^{i - m_t/2}$ for $i = 1, \dots, m_t/2$, and $\alpha_i = \pi - \frac{\pi}{2} 2^{m_t/2+1-i}$ for $i = m_t/2+1, \dots, m_t$. Now we let

$$\tau_1, \dots, \tau_{M_t} \tag{30}$$

denote the 16-point piecewise Chebyshev gird on the intervals

$$(\alpha_1, \alpha_2), (\alpha_2, \alpha_3), \dots, (\alpha_{m_t-1}, \alpha_{m_t}) \tag{31}$$

with $M_t = 15(m_t - 1) + 1$ points in total; let

$$\gamma_1, \dots, \gamma_{M_\nu} \tag{32}$$

denote the nodes of the 24-point piecewise Chebyshev grid on the intervals

$$(\beta_1, \beta_2), (\beta_2, \beta_3), \dots, (\beta_{m_\nu - 1}, \beta_{m_\nu}) \tag{33}$$

with $M_t = 23(m_\nu - 1) + 1$ points in total. The piecewise Chebyshev grids (30) and (32) form a tensor product

$$\{(\tau_i, \gamma_j) : i = 1, \dots, M_t,\ j = 1, \dots, M_\nu\}. \tag{34}$$

Then $\psi^{(a,b)}(t,\nu)$ and $M^{(a,b)}(t,\nu)$ can be obtained via barcyentric Chebyshev interpolations in t and ν via their values at the tensor product restricted to the piece $(\alpha_i, \alpha_{i+1}) \times (\beta_j, \beta_{j+1})$ that (t,ν) belongs to for some i and j.

Here we summarize the operation complexity of the whole algorithm above. A detailed discussion can be found in [19]. ODE solvers are applied for $\mathcal{O}(\log(N_{\max}))$ values of ν, and the solution of the solver is evaluated at $\mathcal{O}(\log(N_{\max}))$ values of t, making the total running time of the procedure just described $\mathcal{O}\left(\log^2(N_{\max})\right)$.

Once the values of $\psi_\nu^{(a,b)}$ and $M_\nu^{(a,b)}$ are ready at the tensor product of the piecewise Chebyshev grids (30) and (32), they can be evaluated for any t and ν via repeated application of the barycentric Chebyshev interpolation formula in the same number of operations which is independent of ν and t.

3 Fast Jacobi polynomial transforms

Without loss of generality, we assume that $\psi^{(a,b)}(t,\nu)$, $M^{(a,b)}(t,\nu)$, $P_\nu^{(a,b)}$, and $Q_\nu^{(a,b)}$ can be evaluated in $O(1)$ operations for any $t \in (0,\pi)$ and $\nu \in [0, N_{\max}]$. A fast algorithm for rapidly computing Gauss-Jacobi quadrature nodes and weights in the Jacobi polynomial transform in (8) has also been introduced in [19]. We refer the reader to [19] for details and assume that quadrature nodes and weights are available in this section.

3.1 One-dimensional transform and its inverse

Here we propose a new variant of the one-dimensional Jacobi polynomial transform for $\tilde{P}_j^{(a,b)}(t)$ in [19]. The new algorithm simplifies the discussion of the fast algorithm. The transform for $\tilde{Q}_j^{(a,b)}(t)$ is similar. For our purpose, we consider the n^{th} order uniform forward Jacobi polynomial transform with a scaling, that is, calculate the vector of values

$$\begin{pmatrix} f(t_1)\sqrt{w_1} \\ f(t_2)\sqrt{w_2} \\ \vdots \\ f(t_n)\sqrt{w_n} \end{pmatrix} \tag{35}$$

given the vector

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} \tag{36}$$

of the coefficients in the expansion

$$f(t) = \sum_{j=0}^{n-1} \alpha_j \tilde{P}_j^{(a,b)}(t); \tag{37}$$

here, $t_1, \ldots, t_n, w_1 \ldots, w_n$ are the nodes and weights of the n-point trigonometric Gauss-Jacobi quadrature rule corresponding to the parameters a and b. The "nonuniform" forward Jacobi transform does not require trigonometric Gauss-Jacobi quadrature nodes and weights.

In the uniform transform, the properties of the trigonometric Gauss-Jacobi quadrature rule and the weighting by square roots in (35) ensure that the $n \times n$ matrix $W\mathcal{J}_n^{(a,b)}$ taking (36) to (35) is orthogonal, where W is the $n \times n$ matrix

$$W = \begin{pmatrix} \sqrt{w_1} & 0 & 0 & 0 \\ 0 & \sqrt{w_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sqrt{w_n} \end{pmatrix}, \tag{38}$$

and $\mathcal{J}_n^{(a,b)}$ is the $n \times n$ matrix representing the sum in (37). Hence, the inverse transform is a simple matvec by the transpose of $W\mathcal{J}_n^{(a,b)}$. In the nonuniform case, $\mathcal{J}_n^{(a,b)}$ is usually a very ill-conditioned matrix and the inverse transform requires solving a challenging system of linear equations, which will be discussed in a separate paper in the future.

It follows from (14) that the (j,k)-th entry of $\mathcal{J}_n^{(a,b)}$ is

$$\left(\mathcal{J}_n^{(a,b)}\right)_{jk} = \begin{cases} M^{(a,b)}(t_j, k-1)\cos\left(\psi^{(a,b)}(t_j, k-1)\right), & \text{if } k-1 > 27, \\ \tilde{P}_{k-1}^{(a,b)}(t_j), & \text{otherwise}, \end{cases} \tag{39}$$

since the part consisting of low-degree polynomials is computed via three-term recurrence relations or various asymptotic expansions. Let $\mathcal{V}_n^{(a,b)} \in \mathbb{R}^{n\times 28}$ be the submatrix of $\mathcal{J}_n^{(a,b)}$ corresponding to the part of polynomials with degree less than 28, and $\mathcal{G}_n^{(a,b)} \in \mathbb{R}^{n\times n-28}$ be the rest of $\mathcal{J}_n^{(a,b)}$, then

$$\mathcal{J}_n^{(a,b)} = \left[\mathcal{V}_n^{(a,b)}, \mathcal{G}_n^{(a,b)}\right].$$

Note that $\mathcal{G}_n^{(a,b)}$ is the real part of a discrete Fourier integral transform that can be evaluated with a quasi-linear complexity by the algorithm of [23] for applying Fourier integral transforms and a special case of that in [24] for general oscillatory integral transforms. Motivated by this observation, the algorithm in [19] constructs a low-rank matrix $\mathcal{A}_n^{(a,b)} \in \mathbb{C}^{n\times(n-28)}$ whose (j,k)-th entry is

$$M^{(a,b)}(t_j, k+27)\exp\left(\mathfrak{i}\left(\psi^{(a,b)}(t_j, k+27) - (k+27)t_j\right)\right). \tag{40}$$

Denote the $n \times (n-28)$ nonuniform FFT matrix, whose (j,k)-th entry is

$$\exp\left(i(k+27)t_j\right), \tag{41}$$

as $\mathcal{N}_n$. Then

$$\mathcal{J}_n^{(a,b)} = \left[\mathcal{V}_n^{(a,b)}, \Re(\mathcal{A}_n^{(a,b)} \otimes \mathcal{N}_n)\right], \tag{42}$$

where "$\otimes$" denotes the Hadamard product. It can be immediately seen from (42) that $\mathcal{J}_n^{(a,b)}$ can be applied through a small number of NUFFTs [12, 13] and a simple direct summation for $\mathcal{V}_n^{(a,b)}$ in a quasi-linear scaling of n.

More specifically, let a low-rank approximation of $\mathcal{A}_n^{(a,b)}$ be

$$\mathcal{A}_n^{(a,b)} \approx \sum_{j=1}^{r} u_j v_j^T, \tag{43}$$

where u_j and v_j are column vectors for each j. Then the transform from (36) to (35) can be approximated by the following sum,

$$W\mathcal{J}_n^{(a,b)}\alpha \approx W\mathcal{V}_n^{(a,b)}\alpha_{\mathcal{V}} + W\Re(\sum_{j=1}^{r} D_{u_j}\mathcal{N}_n D_{v_j}\alpha_{\mathcal{G}}), \tag{44}$$

where $\alpha_{\mathcal{V}}$ is the subvector of α with the first 28 entries, $\alpha_{\mathcal{G}}$ is the subvector of α with the rest entries, D_u denotes a diagonal matrix with a column vector u on its diagonal. The formula (44) can be carried out by r NUFFTs in $O(rn\log n)$ arithmetic operations.

In practice, inspired by the NUFFT in [13], we can replace the Hadamard product of a low-rank matrix and a NUFFT matrix in (42) with a Hadamard product of a low-rank matrix and a DFT matrix, the matvec of which can be carried out more efficiently with a few numbers of FFTs.

Consider another matrix $\mathcal{B}_n^{(a,b)} \in \mathbb{C}^{n\times(n-28)}$ whose (j,k)-th entry is defined via

$$M^{(a,b)}(t_j, k+27)\exp\left(\mathrm{i}(\psi^{(a,b)}(t_j, k+27) - 2\pi\frac{[t_j n/2\pi]}{n}(k+27))\right), \tag{45}$$

where $[x]$ denotes the integer nearest to x. Then we have

$$\mathcal{J}_n^{(a,b)} = \left[\mathcal{V}_n^{(a,b)}, \Re(\mathcal{B}_n^{(a,b)} \otimes \mathcal{F}_n)\right], \tag{46}$$

where $\mathcal{F}_n$ is an $n \times (n-28)$ matrix whose (j,k)-th entry is

$$\exp\left(i(2\pi\frac{[t_j n/2\pi]}{n}(k+27))\right). \tag{47}$$

It's obvious that $\mathcal{F}_n$ in (46) is a row permutation of an inverse DFT matrix. Note that the difference between the phase functions of $\mathcal{A}_n^{(a,b)}$ and $\mathcal{B}_n^{(a,b)}$ is less than π. Hence, $\mathcal{B}_n^{(a,b)}$ is also a low-rank matrix and we can apply the randomized SVD in Algorithm 2.1 to construct a low-rank approximation of $\mathcal{B}_n^{(a,b)}$ in $O(r^2 n)$ operations.

Suppose we have constructed an approximate rank-r SVD of $\mathcal{B}_n^{(a,b)}$ up to a desired accuracy using Algorithm 2.1; that is, suppose that we have computed the factorization

$$\mathcal{B}_n^{(a,b)} \approx USV, \tag{48}$$

where $U \in \mathbb{C}^{n\times r}$, $V \in \mathbb{C}^{r\times(n-28)}$, and $S \in \mathbb{R}^{r\times r}$ is a positive definite diagonal matrix. By rearranging the factors above, we have

$$\mathcal{B}_n^{(a,b)} \approx (US^{\frac{1}{2}})(S^{\frac{1}{2}}V) = u_1 v_1^T + \cdots + u_r v_r^T, \tag{49}$$

where u_i and v_i^T denote the i^{th} column vector of $US^{\frac{1}{2}}$ and the i^{th} row vector of $S^{\frac{1}{2}}V$, respectively, and T denotes the matrix transpose. Once (49) is ready, we have

$$W\mathcal{J}_n^{(a,b)}\alpha \approx W\mathcal{V}_n^{(a,b)}\alpha_{\mathcal{V}} + W\Re(((\sum_{j=1}^{r} u_j v_j^T) \otimes \mathcal{F}_n)\alpha_{\mathcal{G}}) = W\mathcal{V}_n^{(a,b)}\alpha_{\mathcal{V}} + W\Re(\sum_{j=1}^{r} D_{u_j}\mathcal{F}_n D_{v_j}\alpha_{\mathcal{G}}), \tag{50}$$

where D_u denotes a diagonal matrix with u on its diagonal. Formula (50) indicates that the Jacobi polynomial transform can be evaluated efficiently via r inverse FFTs, which requires $O(rn \log n)$ arithmetic operations and $O(rn)$ memory.

Compared to (44) used in the original fast Jacobi transform in [19], the number of inverse FFTs in (50) has been optimized. Thus Formula (50) would take fewer operations to compute a multi-dimensional transform. Note that, compared to the original method in [19], though our new method using (50) would cost more time to construct a low-rank approximation, the optimized rank of the low-rank approximation would accelerate the application of the multi-dimensional Jacobi transform. In addition, as we mentioned previously, the new method works in a larger range of a and b. In Section 4, we will provide numerical comparisons to demonstrate the superior of the new method over the method in [19].

The fast inverse Jacobi transform in the uniform case can be carried out in a similar manner, since $W\mathcal{J}_n^{(a,b)}$ is an orthonormal matrix. In fact, the inverse transform can be computed via

$$\alpha_{\mathcal{V}} = \left(\mathcal{V}_n^{(a,b)}\right)^T W^T W f,$$

and

$$\alpha_{\mathcal{G}} \approx \Re(((\sum_{j=1}^{r} u_j v_j^T) \otimes \mathcal{F}_n)^T W^T W f) = \Re(\sum_{j=1}^{r} D_{v_j} \mathcal{F}_n^T D_{u_j} W^T W f), \tag{51}$$

where $\mathcal{F}_n^T$ is a permutation of the DFT matrix. Therefore, the inverse Jacobi polynomial transform can also be computed via r FFTs.

It is worth emphasizing that the fast algorithm introduced in this section also works for non-uniform Jacobi polynomial transforms since the low-rankness of $\mathcal{B}_n^{(a,b)}$ is independent of the samples in t and the quadrature weights in the uniform Jacobi polynomial transform. There will be numerical examples later to verify this. The inverse transform in the non-uniform case requires solving a highly ill-conditioned linear system, which will be reported in a separate paper in the future.

3.2 Two-dimensional transform and its inverse

In this section, we extend our algorithm to the two-dimensional case, using the new method developed in the last section. We only focus on the transform for $\tilde{P}_\nu^{(a,b)}(t)$; the one for $\tilde{Q}_{k-1}^{(a,b)}(t)$ is similar. We will adopt the MATLAB notation $A(:)$ as a vector resulting from reshaping the matrix A into a vector. We also use similar notations as in Section 3.1, e.g., $\mathcal{J}_{n,\cdot}^{(a,b)}$, $\mathcal{V}_{n,\cdot}^{(a,b)}$, $\mathcal{G}_{n,\cdot}^{(a,b)}$, $\mathcal{B}_{n,\cdot}^{(a,b)}$, $\mathcal{F}_{n,\cdot}$, and $W_\cdot$ denote corresponding matrices analogous to their counterparts in Section 3.1, respectively, with "$\cdot$" specifying a variable x or y in the spatial domain. In the rest of this section, we always assume a low-rank approximation

$$\mathcal{B}_{n,\cdot}^{(a,b)} = \sum_{i=1}^{r_\cdot} u_{\cdot,i} v_{\cdot,i}^T \tag{52}$$

has been obtained by Algorithm 2.1 up to a desired accuracy for "$\cdot$" as x or y.

Given locations $\{x_i\}_{i=1,\cdots,n} \subset (0,\pi)$ and $\{y_i\}_{i=1,\cdots,n} \subset (0,\pi)$, with no substantial difference, the forward and inverse two-dimensional Jacobi polynomial transforms arise in the following Jacobi expansion

$$f(x_i, y_j) = \sum_{k=1}^{n}\sum_{\ell=1}^{n} \alpha(k,\ell) \tilde{P}_{k-1}^{(a,b)}(x_i) \tilde{P}_{\ell-1}^{(a,b)}(y_j), \quad \text{for } i,j = 1,\cdots,n, \tag{53}$$

where α denotes an expansion coefficients matrix. The forward and inverse transform can be defined analogously as in the one-dimensional case. When both $\{x_i\}_{i=1,\cdots,n}$ and $\{y_i\}_{i=1,\cdots,n}$ are exactly the nodes of the trigonometric Gauss-Jacobi quadrature rule, the corresponding transform is referred as the uniform transform. With additional weight matrices W_x and W_y, the forward transformation matrix $(W_x\mathcal{J}_{n,x}^{(a,b)})\odot(W_y\mathcal{J}_{n,y}^{(a,b)})$ taking $\alpha(:)$ to $(W_x\odot W_y)f(:)$ is orthogonal, where "$\odot$" denotes the Kronecker product. When $\{x_i\}_{i=1,\cdots,n}$, $\{y_i\}_{i=1,\cdots,n}$, and weights are not given by the trigonometric Gauss-Jacobi quadrature rule, we refer the corresponding transform as a non-uniform transform.

With the low-rank approximations in (52) available, we have

$$\begin{aligned}
&\left[(W_x\mathcal{J}_{n,x}^{(a,b)})\odot(W_y\mathcal{J}_{n,y}^{(a,b)})\right]\alpha(:)\\
&\approx(W_x\odot W_y)([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}])\alpha(:)\\
&+\sum_{i=1}^{r_y}\Re((W_x\odot W_y)([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot D_{u_{y,i}})(I_n\odot\mathcal{F}_{n,y})(I_n\odot D_{v_{y,i}})\alpha(:))\\
&+\sum_{i=1}^{r_x}\Re((W_x\odot W_y)(D_{u_{x,i}}\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}])(\mathcal{F}_{n,x}\odot I_n)(D_{v_{x,i}}\odot I_n)\alpha(:))\\
&+\sum_{i=1}^{r_x}\sum_{j=1}^{r_y}\Re((W_x\odot W_y)(D_{u_{x,i}}\odot D_{u_{y,j}})(\mathcal{F}_{n,x}\odot\mathcal{F}_{n,y})(D_{v_{x,i}}\odot D_{v_{y,j}})\alpha(:)),
\end{aligned}\tag{54}$$

where $\mathbf{0}$ is an $n\times(n-28)$ zero matrix. In our numerical implement, since $\mathbf{0}$ does not contribute to the numerical result, there are just $O(n)$ entries of $\alpha(:)$ needed to be applied in the first three terms above. Formula (54) indicates that 2D uniform Jacobi polynomial transforms can be evaluated via $O((r_x+r_y)n)$ 1D inverse FFTs and $O(r_xr_y)$ 2D inverse FFTs, which results in $O(r_xr_yn^2\log(n))$ arithmetic operations and $O(r_xr_yn^2)$ memories. We would like to emphasize that this algorithm also works for 2D non-uniform forward transform since it is actually a tensor form of the one-dimensional transform. There will be numerical examples later to verify this.

The fast 2D inverse Jacobi transform in the uniform case can be carried out in a similar manner, since both $W_x\mathcal{J}_{n,x}^{(a,b)}$ and $W_y\mathcal{J}_{n,y}^{(a,b)}$ are orthonormal matrices. In fact, the 2D inverse transform can be computed via

$$\begin{aligned}
&\alpha(:)=\left[(\mathcal{J}_{n,x}^{(a,b)})^T\odot(\mathcal{J}_{n,y}^{(a,b)})^T\right]\tilde{f}(:)\\
&\approx([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]^T\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]^T)\tilde{f}(:)\\
&+\sum_{i=1}^{r_y}\Re((I_n\odot D_{v_{y,i}})(I_n\odot\mathcal{F}_{n,y}^T)([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]^T\odot D_{u_{y,i}})\tilde{f}(:))\\
&+\sum_{i=1}^{r_x}\Re((D_{v_{x,i}}\odot I_n)(\mathcal{F}_{n,x}^T\odot I_n)(D_{u_{x,i}}\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]^T)\tilde{f}(:))\\
&+\sum_{i=1}^{r_x}\sum_{j=1}^{r_y}\Re((D_{v_{x,i}}\odot D_{v_{y,j}})(\mathcal{F}_{n,x}^T\odot\mathcal{F}_{n,y}^T)(D_{u_{x,i}}\odot D_{u_{y,j}})\tilde{f}(:)),
\end{aligned}\tag{55}$$

where $\tilde{f}(:):=(W_x\odot W_y)(W_x\odot W_y)f(:)$. In our numerical implement, since $\mathbf{0}$ does not contribute to the numerical result, we just need to compute $O(n)$ entries of $\alpha(:)$ for the first three terms above. Therefore, 2D inverse Jacobi polynomial transforms can also be evaluated via $O((r_x+r_y)n)$ 1D FFTs and $O(r_xr_y)$ 2D FFTs.

3.3 Three-dimensional transform and its inverse

In this section, we continue to extend our algorithm for 3D Jacobi polynomial transform and its inverse. Analogously, we just discuss the transform for $\tilde{P}_{\nu}^{(a,b)}(t)$ and the transform for $\tilde{Q}_{k-1}^{(a,b)}(t)$ is similar. The notations in Section 3.2 will be inherited here.

Given locations $\{x_i\}_{i=1,\cdots,n}$, $\{y_i\}_{i=1,\cdots,n}$, and $\{z_i\}_{i=1,\cdots,n}$ in $(0,\pi)$, with no substantial difference, the forward and inverse three-dimensional Jacobi polynomial transforms arise in the following Jacobi expansion

$$f(x_i,y_j,z_k)=\sum_{h=1}^{n}\sum_{\ell=1}^{n}\sum_{m=1}^{n}\alpha(k,\ell,m)\tilde{P}_{h-1}^{(a,b)}(x_i)\tilde{P}_{\ell-1}^{(a,b)}(y_j)\tilde{P}_{m-1}^{(a,b)}(z_k),\quad \text{for } i,j,k=1,\cdots,n, \tag{56}$$

where α denotes a three-dimensional tensor containing expansion coefficients. The forward and inverse transforms can be defined analogously to the three-dimensional case. When $\{x_i\}_{i=1,\cdots,n}$, $\{y_i\}_{i=1,\cdots,n}$, and $\{z_i\}_{i=1,\cdots,n}$ are all exactly the nodes of the trigonometric Gauss-Jacobi quadrature rule, the corresponding transform is referred as the uniform transform. Otherwise, we refer the corresponding transform as a non-uniform transform.

In the uniform transform, in order to take advantage of orthogonality, we consider the the tensor product $(W_x\mathcal{J}_{n,x}^{(a,b)})\odot(W_y\mathcal{J}_{n,y}^{(a,b)})\odot(W_y\mathcal{J}_{n,z}^{(a,b)})$ that takes $\alpha(:)$ to $(W_x\odot W_y\odot W_z)f(:)$. Once the related low-rank approximations,

$$\mathcal{B}_{n,\cdot}^{(a,b)}=\sum_{i=1}^{r_\cdot}u_{\cdot,i}v_{\cdot,i}^{T}, \tag{57}$$

have been obtained, we have

$$(W_x\odot W_y\odot W_z)f(:)=(W_x\odot W_y\odot W_z)(\mathcal{J}_{n,x}^{(a,b)}\odot\mathcal{J}_{n,y}^{(a,b)}\odot\mathcal{J}_{n,z}^{(a,b)})\alpha(:), \tag{58}$$

where

$$\begin{aligned}
&(\mathcal{J}_{n,x}^{(a,b)}\odot\mathcal{J}_{n,y}^{(a,b)}\odot\mathcal{J}_{n,z}^{(a,b)})\alpha(:)\\
&\approx([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]\odot[\mathcal{V}_{n,z}^{(a,b)}\quad \mathbf{0}])\alpha(:)\\
&+\sum_{i=1}^{r_z}\Re(([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]\odot D_{u_{z,i}})(I_n\odot I_n\odot\mathcal{F}_{n,z})(I_n\odot I_n\odot D_{v_{z,i}})\alpha(:))\\
&+\sum_{i=1}^{r_y}\Re(([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot D_{u_{y,i}}\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}])(I_n\odot\mathcal{F}_{n,y}\odot I_n)(I_n\odot D_{v_{y,i}}\odot I_n)\alpha(:))\\
&+\sum_{j=1}^{r_y}\sum_{k=1}^{r_z}\Re(([\mathcal{V}_{n,x}^{(a,b)}\quad \mathbf{0}]\odot D_{u_{y,j}}\odot D_{u_{z,k}})(I_n\odot\mathcal{F}_{n,y}\odot\mathcal{F}_{n,z})(I_n\odot D_{v_{y,j}}\odot D_{v_{z,k}}))\alpha(:)\\
&+\sum_{i=1}^{r_x}\Re((D_{u_{x,i}}\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]\odot[\mathcal{V}_{n,z}^{(a,b)}\quad \mathbf{0}])(\mathcal{F}_{n,x}\odot I_n\odot I_n)(D_{v_{x,i}}\odot I_n\odot I_n)\alpha(:))\\
&+\sum_{j=1}^{r_x}\sum_{k=1}^{r_z}\Re((D_{u_{x,j}}\odot[\mathcal{V}_{n,y}^{(a,b)}\quad \mathbf{0}]\odot D_{u_{z,k}})(\mathcal{F}_{n,x}\odot I_n\odot\mathcal{F}_{n,z})(D_{v_{x,j}}\odot I_n\odot D_{v_{z,k}}))\alpha(:)\\
&+\sum_{j=1}^{r_x}\sum_{k=1}^{r_y}\Re((D_{u_{x,j}}\odot D_{u_{y,k}}\odot[\mathcal{V}_{n,z}^{(a,b)}\quad \mathbf{0}])(\mathcal{F}_{n,x}\odot\mathcal{F}_{n,y}\odot I_n)(D_{v_{x,j}}\odot D_{v_{y,k}}\odot I_n))\alpha(:)\\
&+\sum_{i=1}^{r_x}\sum_{j=1}^{r_y}\sum_{k=1}^{r_z}\Re((D_{u_{x,i}}\odot D_{u_{y,j}}\odot D_{u_{z,k}})(\mathcal{F}_{n,x}\odot\mathcal{F}_{n,y}\odot\mathcal{F}_{n,z})(D_{v_{x,i}}\odot D_{v_{y,j}}\odot D_{v_{z,k}})\alpha(:)).
\end{aligned} \tag{59}$$

In numerical implement, since $\mathbf{0}$ does not contribute to numerical result, there are just $O(n^2)$ entries of $\alpha(:)$ needed to be applied in the first seven terms above. Formula (59) indicates that 3D uniform Jacobi polynomial transform can be evaluated via $O((r_x + r_y + r_z)n^2)$ inverse FFTs, $O((r_xr_y + r_xr_z + r_yr_z)n)$ 2D inverse FFTs and $O(r_xr_yr_z)$ inverse 3D FFTs, which results in $O(r_xr_yr_zn^3\log(n))$ arithmetic operations and $O(r_xr_yr_zn^3)$ memories. Again, we would like to emphasize that this algorithm also works for 3D non-uniform forward transform. There will be numerical examples later to verify this.

The fast 3D inverse (uniform) Jacobi transform in the uniform case can be carried out in a similar manner, since $W_x\mathcal{J}_{n,x}^{(a,b)}$, $W_y\mathcal{J}_{n,y}^{(a,b)}$ and $W_z\mathcal{J}_{n,z}^{(a,b)}$ are all orthonormal matrices. In fact, the 3D inverse transform can be computed via

$$\begin{aligned}
&(\mathcal{J}_{n,x}^{(a,b)} \odot \mathcal{J}_{n,y}^{(a,b)} \odot \mathcal{J}_{n,z}^{(a,b)})^T \tilde{f}(:)\\
&\approx ([\mathcal{V}_{n,x}^{(a,b)} \quad \mathbf{0}]^T \odot [\mathcal{V}_{n,y}^{(a,b)} \quad \mathbf{0}]^T \odot [\mathcal{V}_{n,z}^{(a,b)} \quad \mathbf{0}]^T)\tilde{f}(:)\\
&+\sum_{i=1}^{r_z} \Re((I_n \odot I_n \odot D_{v_{z,i}})(I_n \odot I_n \odot \mathcal{F}_{n,z}^T)([\mathcal{V}_{n,x}^{(a,b)} \quad \mathbf{0}]^T \odot [\mathcal{V}_{n,y}^{(a,b)} \quad \mathbf{0}]^T \odot D_{u_{z,i}})\tilde{f}(:))\\
&+\sum_{i=1}^{r_y} \Re((I_n \odot D_{v_{y,i}} \odot I_n)(I_n \odot \mathcal{F}_{n,y}^T \odot I_n)([\mathcal{V}_{n,x}^{(a,b)} \quad \mathbf{0}]^T \odot D_{u_{y,i}} \odot [\mathcal{V}_{n,y}^{(a,b)} \quad \mathbf{0}]^T)\tilde{f}(:))\\
&+\sum_{j=1}^{r_y}\sum_{k=1}^{r_z} \Re((I_n \odot D_{v_{y,j}} \odot D_{v_{z,k}})(I_n \odot \mathcal{F}_{n,y}^T \odot \mathcal{F}_{n,z}^T)([\mathcal{V}_{n,x}^{(a,b)} \quad \mathbf{0}]^T \odot D_{u_{y,j}} \odot D_{u_{z,k}})\tilde{f}(:))\\
&+\sum_{i=1}^{r_x} \Re((D_{v_{x,i}} \odot I_n \odot I_n)(\mathcal{F}_{n,x}^T \odot I_n \odot I_n)(D_{u_{x,i}} \odot [\mathcal{V}_{n,y}^{(a,b)} \quad \mathbf{0}]^T \odot [\mathcal{V}_{n,z}^{(a,b)} \quad \mathbf{0}]^T)\tilde{f}(:))\\
&+\sum_{j=1}^{r_x}\sum_{k=1}^{r_z} \Re((D_{v_{x,j}} \odot I_n \odot D_{v_{z,k}})(\mathcal{F}_{n,x}^T \odot I_n \odot \mathcal{F}_{n,z}^T)(D_{u_{x,j}} \odot [\mathcal{V}_{n,y}^{(a,b)} \quad \mathbf{0}]^T \odot D_{u_{z,k}})\tilde{f}(:))\\
&+\sum_{j=1}^{r_x}\sum_{k=1}^{r_y} \Re((D_{v_{x,j}} \odot D_{v_{y,k}} \odot I_n)(\mathcal{F}_{n,x}^T \odot \mathcal{F}_{n,y}^T \odot I_n)(D_{u_{x,j}} \odot D_{u_{y,k}} \odot [\mathcal{V}_{n,z}^{(a,b)} \quad \mathbf{0}]^T)\tilde{f}(:))\\
&+\sum_{i=1}^{r_x}\sum_{j=1}^{r_y}\sum_{k=1}^{r_z} \Re((D_{v_{x,i}} \odot D_{v_{y,j}} \odot D_{v_{z,k}})(\mathcal{F}_{n,x}^T \odot \mathcal{F}_{n,y}^T \odot \mathcal{F}_{n,z}^T)(D_{u_{x,i}} \odot D_{u_{y,j}} \odot D_{u_{z,k}})\tilde{f}(:)).
\end{aligned} \tag{60}$$

where $\tilde{f}(:) = (W_x \odot W_y \odot W_z)^2 f(:)$.

In our numerical implement, since $\mathbf{0}$ does not contribute to the summation above, we just need to compute $O(n^2)$ entries of $\alpha(:)$ for the first seven terms above. Therefore, 3D inverse Jacobi polynomial transform can also be evaluated via $O((r_x+r_y+r_z)n^2)$ 1D FFTs, $O((r_xr_y+r_xr_z+r_yr_z)n)$ 2D FFTs, and $O(r_xr_yr_z)$ 3D FFTs.

4 Numerical results

This section presents several numerical examples to demonstrate the effectiveness of the algorithms proposed above. Section 4.1 provides a comparison of the original method in [19] named as method **CHEB** using (44) in Section 3.1, and our new method named as **RS** using (50) in Section 3.1 to demonstrate the superiority of our new method. In Section 4.2, we apply our new method to three examples corresponding to 1D, 2D, and 3D Jacobi polynomial transforms with parameters $a = b = 0.25$, respectively. All implementations are in MATLAB® on a server computer with 28

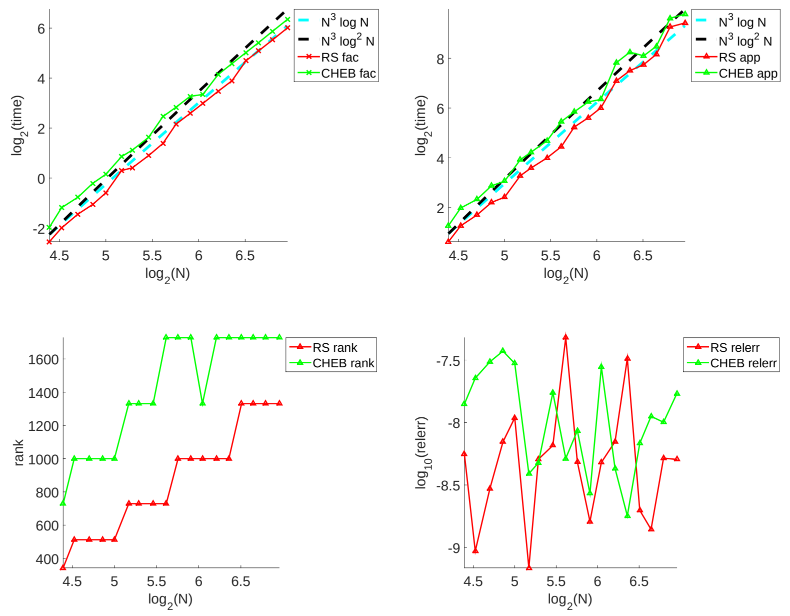

Figure 1: Numerical results for the comparison of the CHEB and RS method for 3D forward uniform transform. Top: the factorization and application time from left to right, respectively. Bottom left and right: the numerical ranks of the low-rank matrices provided by CHEB and RS, and the relative errors of the fast matvec compared to the exact summation.

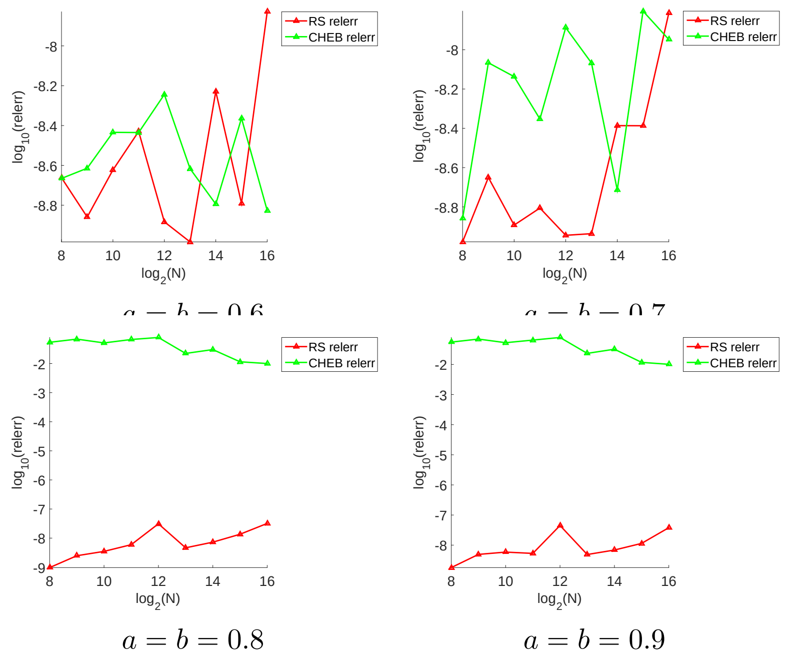

Figure 2: Numerical accuracy of the CHEB and RS algorithms for 1D forward uniform transform when a and b are assigned as 0.6, 0.7, 0.8, and 0.9, respectively.

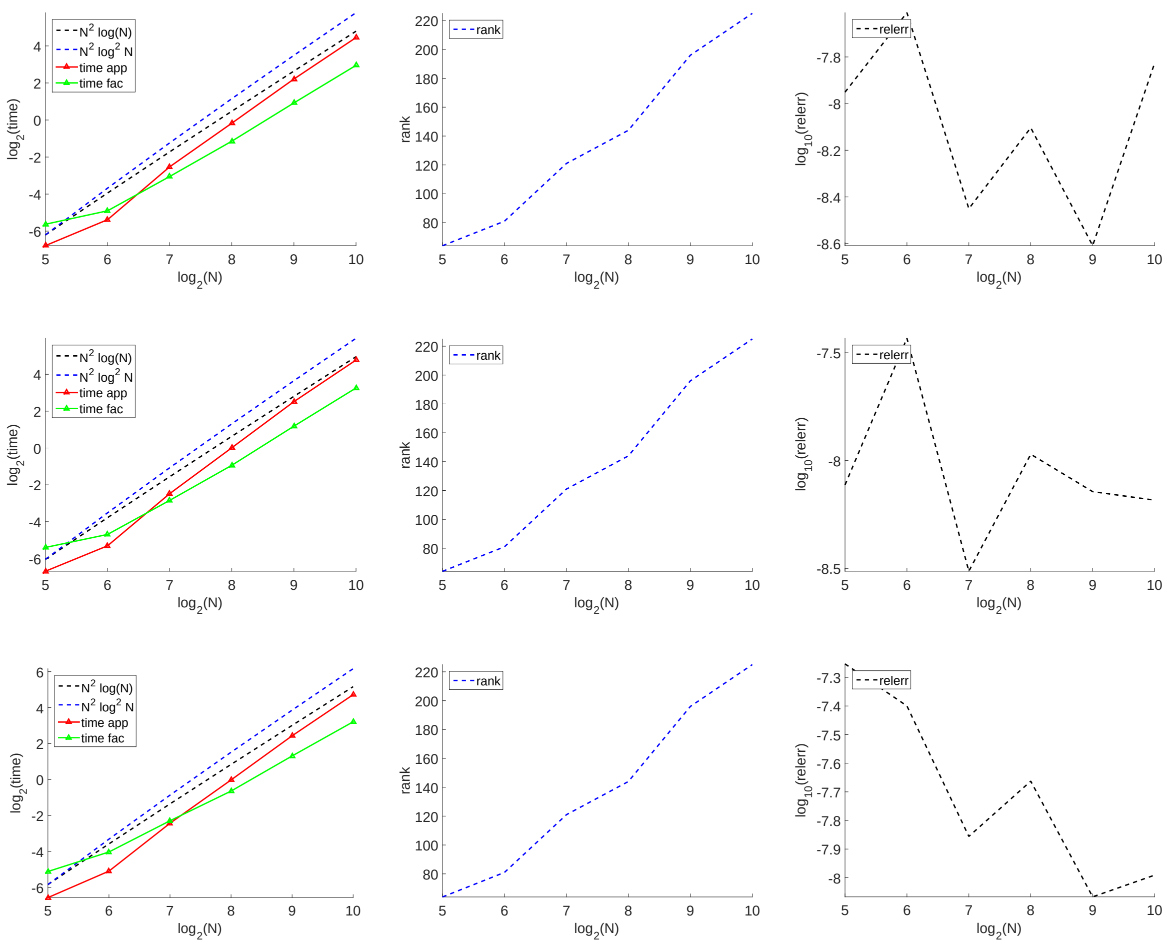

Figure 3: Numerical results for the 2D uniform forward transform (the first row), the 2D uniform inverse transform (the second row), and the 2D nonuniform forward transform. The running time in second, the numerical rank of the low-rank matrix in (52), and the relative error of the fast algorithm compared to the exact summation are visualized from left to right columns.

processors and 2.6 GHz CPU. We have made our code, including that for all of the experiments described here, available on GitLab at the following address:

`https://gitlab.com/FastOrthPolynomial/Jacobi.git`

In our numerical results, N denotes the number of grid points per dimension; "fac" and "app" stand for the factorization time for setting up the algorithm and the application time for applying the algorithm, respectively; "relerr" means the relative error of the fast matvec. The rank parameter r and the accuracy parameter ϵ in all low-rank factorization algorithms are set to $\lceil 2\log_2(N)\rceil$ and $1e-8$, respectively, where $\lceil\cdot\rceil$ is the ceiling function. To make fair comparisons, the oversampling parameter q in Algorithm 2.1 for the RS method is set such that $q \times r$ is approximately equal to the number of piecewise Chebyshev grids in (30) used in the CHEB method. We perform the same experiment 10 times and summarize the average statistics in the following figures.

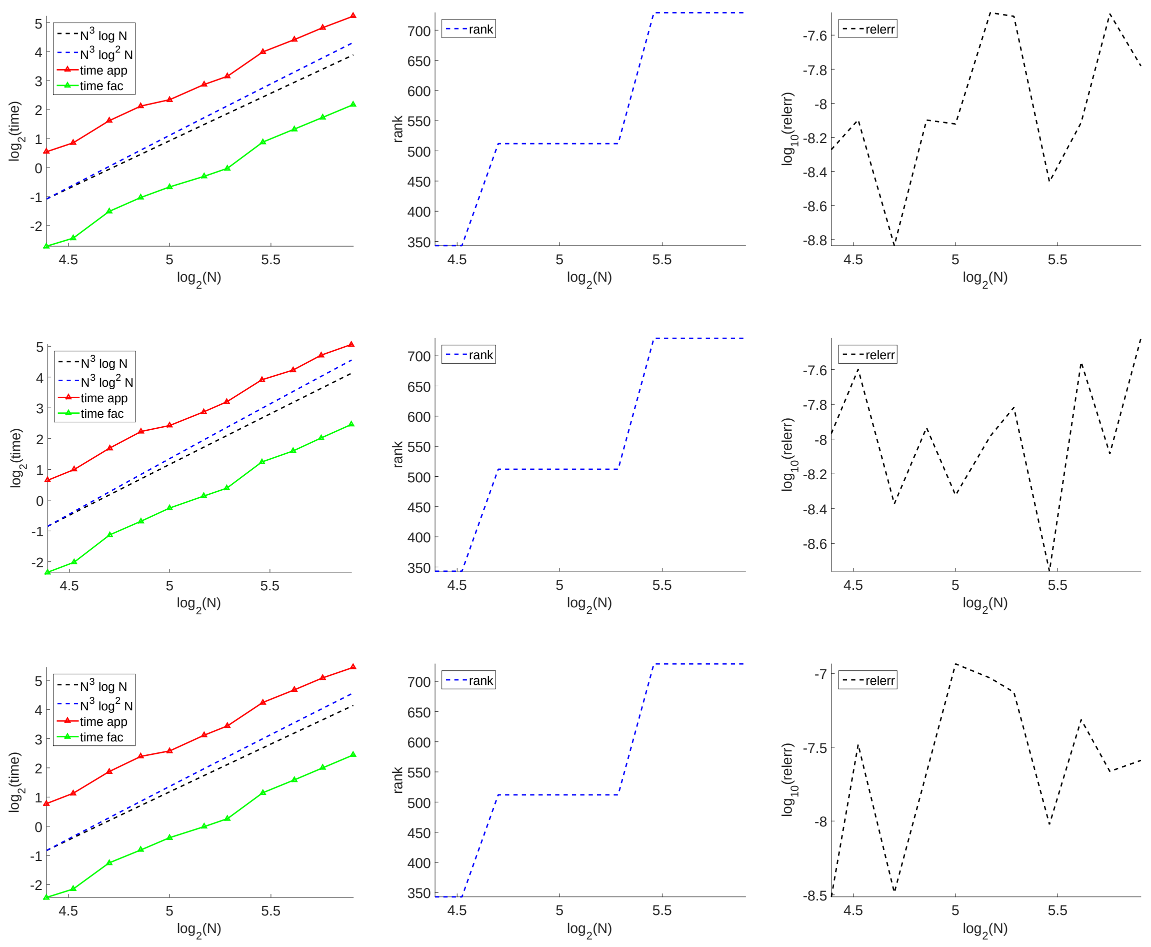

Figure 4: Numerical results for the 3D uniform forward transform (the first row), the 3D uniform inverse transform (the second row), and the 3D nonuniform forward transform. The running time in second, the numerical rank of the low-rank matrix in (57), and the relative error of the fast algorithm compared to the exact summation are visualized from left to right columns.

4.1 Comparison of CHEB and RS methods

4.1.1 Comparisons in the three-dimensional case for a and b in $(-\frac{1}{2}, \frac{1}{2})$

First, we provide performance comparisons of the CHEB algorithm and the RS algorithm in the three-dimensional case with $a = b = 0.25$ to demonstrate that the RS is more efficient than the CHEB algorithm. In the one and two-dimensional cases, RS is also faster than CHEB but the speed-up is less obvious.

As we can see from the results summarized in Figure 1, the RS method provides a more compact matrix compression while keeping the compression accuracy competitive to that of the CHEB algorithm. The more compact compression results in faster set-up and application time for the Jacobi transform for all problem sizes.

4.1.2 Comparisons in the one-dimensional case for a and b in $(-1,1) \setminus (-\frac{1}{2}, \frac{1}{2})$

Second, we provide comparisons of the CHEB and RS algorithms when a and b are assigned as 0.6, 0.7, 0.8, and 0.9 in the range $(-1,1) \setminus (-\frac{1}{2}, \frac{1}{2})$, which is the range of a and b that has not been explored in [19]. Since the case when a and b are negative is similar to the case when a and b are positive, we only show the results for positive a and b. The numerical results are summarized in Figure 2 and they show that both CHEB and RS achieve desired accuracy when a and b are not close to 1; RS works but CHEB fails when a and b are close to 1. Hence, the newly proposed RS algorithm works in a larger range of a and b than the CHEB method in [19].

4.2 Performance of fast multi-dimensional transforms

Finally, we will present examples for two and three-dimensional Jacobi polynomial transforms using the RS algorithm. In either the two or three-dimensional case, there are numerical results for one uniform forward transform, one uniform inverse transform, and one nonuniform forward transform. a and b are all set to 0.25 in these examples.

Figure 3 and 4 summarize the numerical results in 2D and 3D, respectively. In cases, no matter forward or inverse transform, uniform or nonuniform transform, 2D or 3D, the factorization and application time of our algorithm scale like $O(N^d \log N)$ or $O(N^d \log^2 N)$, where d is the dimension. The numerical rank gradually increases like $O(\log N)$ as the problem size increases. The relative error of the fast algorithm is in line with the desired accuracy in the low-rank factorization.

5 Conclusion

This paper proposed a fast algorithm for multi-dimensional Jacobi polynomial transforms based on the observation that the solution of Jacobi's differential equation can be represented via non-oscillatory phase and amplitude functions. In particular, the transformation matrix corresponding to the discrete transform is a Hadamard product of a numerically low-rank matrix and a multi-dimensional discrete Fourier transform matrix. After constructing a low-rank approximation to the numerical low-rank matrix, the application of the Hadamard product can be carried out via $O(1)$ fast Fourier transforms, resulting in a nearly optimal algorithm to compute the multi-dimensional Jacobi polynomial transform.

We proposed to apply the randomized SVD to construct low-rank factorizations, resulting in a faster Jacobi transform in high-dimensional spaces. Moreover, numerical experiments show that the new fast transform works for a larger class of Jacobi polynomials with parameter a and b in the interval $(-1,1)$ than the one-dimensional algorithm in [19].

For other values of a and b outside $(-1,1)$, the Jacobi transformation matrix is no longer purely oscillatory and hence the proposed method is not applicable. We will tackle this issue in the future via hierarchically dividing the transformation matrix into sub-matrices that are either purely non-oscillatory, which can be handled by fast low-rank factorization, or purely oscillatory, which can be applied via the fast algorithm in this paper.

The fast inverse in the nonuniform case is still an open problem. It involves a highly ill-conditioned linear system of equations. We are working on an efficient preconditioner for this linear system and will summarize our work in a separate paper.

Acknowledgments. H. Yang thanks the support of the start-up grant by the Department of Mathematics at the National University of Singapore.

References

[1] Arieh Iserles. A fast and simple algorithm for the computation of Legendre coefficients. *Numerische Mathematik*, 117(3):529–553, Mar 2011.

[2] Nicholas Hale and Alex Townsend. A fast, simple, and stable Chebyshev–Legendre transform using an asymptotic formula. *SIAM Journal on Scientific Computing*, 36(1):A148–A167, 2014.

[3] Bradley K Alpert and Vladimir Rokhlin. A fast algorithm for the evaluation of Legendre expansions. *SIAM Journal on Scientific and Statistical Computing*, 12(1):158–179, 1991.

[4] V. Rokhlin. A fast algorithm for the evaluation of Legendre expansions. *SIAM Journal on Sci. and Stat. Computing*, 12:158–179, 1991.

[5] Jens Keiner. *Fast Polynomial Transforms.* Logos Verlag, 2011.

[6] Zachary Battles and Lloyd N. Trefethen. An extension of matlab to continuous functions and operators. *SIAM Journal on Scientific Computing*, 25(5):1743–1770, 2004.

[7] Richard Askey. *Orthogonal polynomials and special functions*, volume 21. Siam, 1975.

[8] George E Andrews, Richard Askey, and Ranjan Roy. Special functions, volume 71 of encyclopedia of mathematics and its applications, 1999.

[9] P. Maroni and Z. da Rocha. Connection coefficients between orthogonal polynomials and the canonical sequence: an approach based on symbolic computation. *Numerical Algorithms*, 47(3):291–314, Mar 2008.

[10] Luogeng Hua. *Harmonic analysis of functions of several complex variables in the classical domains.* Number 6. American Mathematical Soc., 1963.

[11] Gabor Szeg. *Orthogonal polynomials*, volume 23. American Mathematical Soc., 1939.

[12] L. Greengard and J. Lee. Accelerating the nonuniform fast Fourier transform. *SIAM Review*, 46(3):443–454, 2004.

[13] Diego Ruiz-Antolín and Alex Townsend. A nonuniform fast Fourier transform based on low rank approximation. *SIAM Journal on Scientific Computing*, 40(1):A529–A547, 2018.

[14] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[15] Jens Keiner. Computing with expansions in Gegenbauer polynomials. *SIAM Journal on Scientific Computing*, 31(3):2151–2171, 2009.

[16] Raf Vandebril, Marc Van Barel, and Nicola Mastronardi. *Matrix Computations and Semiseparable Matrices.* The Johns Hopkins University Press, Baltimore, MD, 2008.

[17] Jens Keiner. Gegenbauer polynomials and semiseparable matrices. *Electronic Transactions on Numerical Analysis*, 30:26–53, 2008.

[18] Michael ONeil, Franco Woolfe, and Vladimir Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Applied and Computational Harmonic Analysis*, 28:203–226, 2010.

[19] James Bremer and Haizhao Yang. Fast algorithms for Jacobi expansions via nonoscillatory phase functions. *arXiv:1803.03889 [math.NA]*, 2018.

[20] *NIST Digital Library of Mathematical Functions.* http://dlmf.nist.gov/, Release 1.0.13 of 2016-09-16. 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.

[21] Zhu Heitman, James Bremer, and Vladimir Rokhlin. On the existence of nonoscillatory phase functions for second order ordinary differential equations in the high-frequency regime. *Journal of Computational Physics*, 290:1 – 27, 2015.

[22] James Bremer. On the numerical solution of second order differential equations in the high-frequency regime. *Applied and Computational Harmonic Analysis*, 44:312–349.

[23] Emmanuel Candès, Laurent Demanet, and Lexing Ying. Fast computation of Fourier integral operators. *SIAM Journal on Scientific Computing*, 29(6):2464–2493, 2007.

[24] Haizhao Yang. A unified framework for oscillatory integral transforms: When to use nufft or butterfly factorization? *arXiv preprint arXiv:1803.04128*, 2018.

[25] Björn Engquist and Lexing Ying. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun. Math. Sci.*, 7(2):327–345, 06 2009.

[26] Jiawei Chiu and Laurent Demanet. Sublinear randomized algorithms for skeleton decompositions. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1361–1383, 2013.

[27] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice.* Society for Industrial and Applied Mathematics, 2013.