A Fast Algorithm for Multidimensional Jacobi Polynomial Transform

Abstract

Based on a well-known observation that the solution of Jacobi's differential equation can be accurately represented via a nonoscillatory phase function, we develop a fast algorithm for computing Jacobi polynomial transform. We observe that transformation matrix corresponding to a Jacobi polynomial transform has a special form, that is, a Hadamard product between a low-rank matrix and a DFT matrix. The low-rank matrix can be approximated by randomized methods, thus the application of the Hadamard product can be carried out via $O(1)$ FFTs, resulting in an $O(N \log N)$ algorithm to compute Jacobi polynomial transform where N is the problem size. Numerical results will be shown to demonstrate the efficiency of our algorithm for one and higher dimensional cases.

Keywords. Multidimensional Jacobi Transform, nonoscillatory phase function, randomized low rank approximation, FFT

1 Introduction

Given locations $x_1, \cdots, x_n$, $x_i \in (-1, 1)$ for each $i = 1, \cdots, n$, discrete Jacobi polynomial transform is defined via

$$f(x_k) = \sum_{j=1}^{n} \hat{f}_j P_{j-1}^{(a,b)}(x_k), k = 1, \cdots, m, \tag{1}$$

where $P_\nu^{(a,b)}$ denotes order-ν Jacobi polynomial corresponding to parameters a and b, and $\hat{f}_1, \cdots, \hat{f}_n$ denote real coefficiences. In this paper, for sake of simplicity, "discrete" will be dropped when we mention discrete Jacobi polynomial transform. If the coefficients $\hat{f}_1, \cdots, \hat{f}_n$ are known, the evaluation of f at each location is called forward Jacobi polynomial transform, the inverse process is called inverse Jacobi polynomial transform. Sometimes, "forward" will be dropped when we mention forward Jacobi polynomial transform, but readers could disguise this from context. The discrete Jacobi polynomial transform can be divided into two classes in terms of the locations. When the locations $x_1, \cdots, x_n$ are the nodes of n-point Gauss-Jacobi quadrature rule corresponding to the parameters a and b,

$$\int_{-1}^{1} f(x)(1+x)^a(1+x)^b dx \approx \sum_{j=1}^{n} f(x_j)\omega_j \tag{2}$$

where $\omega_1, \cdots, \omega_n$ are the weights, the sum (1) is called uniform discrete Jacobi polynomial transform. When the locations $x_1, \cdots, x_n$ are not exactly the nodes, $x_i \in (-1, 1)$ for each i, we refer to sum (1) as nonuniform discrete Jacobi polynomial transform. For sake of simplicity, we assume $n = m$ in this paper and often drop the term "discrete" when mention discrete Jacobi polynomial transform.

In recent decades, some fast polynomial transform have been proposed for discrete Jacobi polynomial transform, so we first review them before introducing a new one. For some special cases of Jacobi polynomial transform there are some fast algorithms available, such as [1, 2, 3] for Chebyshev and[4, 5, 6] for Legendre polynomial transform. For general classes of Jacobi polynomial transform, there exist mainly two classes of algorithms as follows. One class of such algorithms provide a way to achieve acceptable cost by allowing one to efficiently replace the sum (1) with an equivalent one

$$f(x_k) = \sum_{j=1}^{n} \hat{c}_j e^{\mathrm{i}j \arccos(x_k)}, k = 1, \cdots, n, \tag{3}$$

with new coefficients $\hat{c}_j$. The sum above can be evaluated, up to a fixated accuracy ϵ, using either the fast Fourier transform (FFT) [7] or its non-equispaced variant (NFFT) (for instance, [8] and [9]) with no more than $O(n \log n + n \log(1/\epsilon))$ arithmetic operations. What remaining to make the total cost acceptable is that replacing the coefficients $\hat{f}_j$ with $\hat{c}_j$ must be efficient enough. Note that a method for making such replacement for Jacobi polynomial transform is actually a part of a more general algorithm for applying connection matrices between sequences of classical orthogonal polynomials. For connection matrices, reading [10, 11] is recommended, other references include [12, 13, 14]. The algorithms for applying connection matrices can be divided into two types in terms of on what techniques they depend [15]. One type of these algorithms are based on the observation that the corresponding connection matrix can be represented as the properly scaled eigenvector matrix of a known (triangular) generator representable semiseparable matrix(see [16, 17] for a comprehensive introduction to semiseparable matrix), whose eigendecomposition can be constructed by employing divide-and-conquer methods [18, 15]. This $O(n \log n)$ algorithm has first been published in [18] , and then modified in [15]. The algorithms of the other type are based on fast multipole method (FMM) which was originated in [19]. More specifically, [6] devised a variant of FFM to compute the connection between Legendre and Chebyshev polynomials of first kind, and then such technique was extended by the author in [20] to Gegenbauer polynomials, and a further modification can be found in [15].

Recently, instead of using (3), the other class of algorithms for computing discrete Jacobi polynomial transform has been proposed, which is based on a well-known observation that the solution of Jacobi's differential equation can be accurately represented via a nonoscillatory phase function. Though phase functions for certain second-order differential equations were first introduced in [21] long ago, the fact that this phase function can be nonoscillatory in some cases has been overlooked until [22] describes some numerical experiments which illustrate several implications of it. Inspired by the nonoscillatory phase function and low-rank matrix techniques in [23, 24], [25] recently develops an algorithm to compute discrete Jacobi polynomial transform. Through exploiting the nonoscillatory phase function, [25] expresses Jacobi polynomial transformation matrix as a Hadamard product between a NUDFT matrix and a low-rank matrix. Once a low-rank approximation of the low-rank matrix obtained, the (uniform) discrete Jacobi polynomial transform can be achieved by carrying out $r = O(1)$ NUFFTs, where r is a numerical rank of the low-rank matrix up to a fixated accuracy ϵ. This nonoscillatory phase function method will be described more elaborately in Section 2.1.

Inspired by the nonoscillatory phase function method, here we observate that the same transformation matrixcan can be expressed as another Hadamard product between a permutated DFT matrix and a low-rank matrix. By applying SVD via randomized sampling, we can construct an approximation of the low-rank matrix with an optimal rank, so that we can evaluate the discrete Jacobi polynomial transform by carrying out an optimal number of FFTs. Since the number of terms needed to be summed in our method would be smaller and each term costs fewer operations

than that in [25], that is because in each term applying an FFT is cheaper than a NUFFT, our method would run faster than the original one, as numerical experiments in Section 3 show. What's more, our method also works for nonuniform Jacobi polynomial transform when the locations are far away from the endpoints of $(-1, 1)$. We acknowledge that the cost of a pre-computation stage to construct an approximation of the low-rank matrix would be more expensive than that in [25], it would be at least compensated by repeatedly applying the same transformation matrix to varied coefficients vectors. We also extend our method for higher dimensional cases, in terms of tensor product. Numerical results indicate that our algorithm could be faster, especially in the cases of high dimensions.

The remainder of the paper is organized as follows. In section 2 we will first briefly introduce the nonoscillatory phase function method and randomized sampling, then describe our one-dimensional algorithm in detail, and at last, extend it to higher dimension version. In section 3, some numerical results would be shown to demonstrate the efficiency of our algorithm. Conclusion and some discussions will be made in Section 4.

2 Compute Jacobi Polynomial Transform

In this section, we will first briefly describe nonoscillatory phase function method in [25] in subsection 2.1, then develop our algorithm for one, two and three dimensional Jacobi Polynomial transform in other subsections. One randomized technique for constructing low-rank approximation–SVD via randomized sampling, which is employed in our algorithm, will also be included as a subsection following subsection 2.1.

2.1 Nonoscillatory phase function method

As we mentioned in Section 1, the phase function for Jacobi's differential equation is nonoscillatory, which indicates a new method for computing Jacobi polynomial transform [22, 25]. More specifically, this well-known observation is that Jacobi polynomial $P_\nu^{(a,b)}$ can be represented in terms of nonoscillatory amplitude and phase functions, that is,

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

where $M^{(a,b)}(t,\nu)$ and $\psi^{(a,b)}(t,\nu)$ are nonoscillatory amplitude and phase functions respectively, and $\tilde{P}_\nu^{(a,b)}$ is defined via

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

with

$$C_\nu^{(a,b)} = \sqrt{(2\nu+a+b+1)\frac{\Gamma(1+\nu)\Gamma(1+\nu+a+b)}{\Gamma(1+\nu+a)\Gamma(1+\nu+b)}} \tag{6}$$

which ensures that 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 introduction of change of variables $x = \cos(t)$ makes the singularities in phase and amplitude functions for Jacobi's differential equation more tractable. We refer to $\tilde{P}_\nu^{(a,b)}$ as modified Jacobi polynomial.

According to (5), to compute discrete Jacobi polynomial transform (1), one just need to apply a necessary but cheap multiplication to the following sum,

$$f(t_i) = \sum_{j=1}^{n} \hat{f}_j \tilde{P}_{j-1}^{(a,b)}(t_i), i = 1, \cdots, n, \tag{7}$$

where $t_1, \cdots, t_n$ are locations coming from the change of viables. The expansion (7) is called modified discrete Jacobi polynomial transform. Analogously, we can define forward and inverse modified discrete Jacobi polynomial transform just like the original transform, and "forward" is sometimes dropped for simplicity. Uniform modified discrete Jacobi polynomial transform are the expansion (7) with uniform locations $t_1, \cdots, t_n$ exactly equal to the nodes of the modified Gauss-Jacobi quadrature rule,

$$\int_0^\pi f(cos(t)) \cos^{2a+1}(\frac{t}{2}) \sin^{2b+1}(\frac{t}{2}) dt \approx \sum_{j=1}^{n} f(\cos(t_j)) \cos^{2a+1}(\frac{t_j}{2}) \sin^{2b+1}(\frac{t_j}{2}) \omega_j, \tag{8}$$

where $w_1, \cdots, w_n$ are weights. When the locations $t_1, \cdots, t_n$ are not exactly the nodes of (8), the expansion (7) is called nonuniform modified discrete Jacobi polynomial transform and the corresponding locations are called nonuniform locations. An algorithm for computing nonuniform locations with an asymptotic running time of $O(n)$ is accessible in [25].

For sake of simplicity, here we just describe how to apply the nonoscillatory phase function method for computing the modified uniform Jacobi polynomial transform (7). Before it, one must evaluate values of the modified Jacobi polynomial $\tilde{P}_\nu^{(a,b)}$ at uniform locations for each integer order ν. Note that the phase and amplitude functions are bivariate functions of t and ν. Owing to the smoothness of the phase and amplitude functions when a and b are in $(-1/2, 1/2)$, there are some fast algorithms available for calculating representations of the nonoscillatory amplitude and phase functions in (4), such as bivariate piecewise Chebyshev discretization scheme [25] which costs only $O(\log^2(n))$ operations. The bivariate piecewise Chebyshev discretization scheme first evaluates the nonoscillatory amplitude and phase functions at some piecewise Chebyshev girds, then evaluates the two functions at any t and ν by repeatedly applying the barycentric Chebyshev interpolation formula in a number of operations which is independent of ν and t. Once the values of nonoscillatory amplitude and phase at uniform locations and integral orders have been obtained, the values of $\tilde{P}_\nu^{(a,b)}(t)$ at uniform locations and integral orders can be evaluated easily.

In terms of (4), and define a diagonal matrix $W = diag(1/\sqrt{w_1}, \cdots, 1/\sqrt{w_n})$, thus uniform Jacobi polynomial transform is equal to a matvec

$$f = W \mathcal{J}_n^{(a,b)} \hat{f}, \tag{9}$$

where the entries of $\mathcal{J}_n^{(a,b)}$ is defined by

$$\mathcal{J}_n^{(a,b)}(j,k) = \tilde{P}_{k-1}^{(a,b)}(t_j)\sqrt{w_j} = M^{(a,b)}(t_j, k-1) \cos(\psi^{(a,b)}(t_j, k-1))\sqrt{w_j}, \tag{10}$$

and $\hat{f} = (\hat{f}_1, \cdots, \hat{f}_n)^T$, $f = (f(t_1), \cdots, f(t_n))^T$. Let $\mathcal{A}_n^{(a,b)}$ be a matrix whose (j,k) entry is defined via

$$M^{(a,b)}(t_j, k-1) \exp(\mathfrak{i}(\psi^{(a,b)}(t_j, k-1) - t_j(k-1)))\sqrt{w_j}, \tag{11}$$

and $\mathcal{N}_n$ be a matrix whose (j,k) entry is

$$\exp(\mathfrak{i} t_j (k-1)). \tag{12}$$

Note that $\mathcal{N}_n$ is actually a NUDFT matrix. Hence $\mathcal{J}_n^{(a,b)}$ can be represented as the real part of a Hadamard product between $\mathcal{A}_n^{(a,b)}$ and $\mathcal{N}_n$, that is

$$\mathcal{J}_n^{(a,b)} = \Re(\mathcal{A}_n^{(a,b)} \otimes \mathcal{N}_n), \tag{13}$$

where "$\otimes$" denotes Hadamard product. The advantage of this expression is that $\mathcal{A}_n^{(a,b)}$ is a low-rank matrix when a and b are in $(-1/2, 1/2)$. More specifically, [25] gives an asymptotic approximation of the nonoscillatory phase function:

$$\psi^{(a,b)}(t,\nu) = (\nu + \frac{a+b+1}{2})t + c + O(\frac{1}{\nu^2}) \ as \ \nu \to \infty \tag{14}$$

with c a constant in the interval $(0, 2\pi)$. Accordingly,

$$\psi^{(a,b)}(t,\nu) - \nu t \tag{15}$$

is of relatively small magnitude, hence $\mathcal{A}_n^{(a,b)}$ is low-rank.

Once a low-rank approximation of $\mathcal{A}_n^{(a,b)}$ up to a fixated accuracy ϵ obtained; that is,

$$\mathcal{A}_n^{(a,b)} \approx \sum_{j=1}^{r} u_j v_j^T \tag{16}$$

where u_j and v_j are column vectors for each j, (9) can be approximated by following sum,

$$f = W\mathcal{J}_n^{(a,b)}\hat{f} \approx W\Re(\sum_{j=1}^{r} D_{u_j}\mathcal{N}_n D_{v_j}\hat{f}), \tag{17}$$

where D_u denotes a diagonal matrix with a column vector u on its diagonal. The formula (17) indicates that (9) can be approximated by carrying out r NUFFTs, resulting in $O(rn\log n)$ arithmetic operations.

2.2 One dimensional transform and its inverse

Note that in nonoscillatory phase function method, there are two main aspects worthy attention. One is that the rank r is not preassigned but up to a fixated accuracy ϵ or to the method we choose while constructing a low-rank approximation of the low-rank matrix. The other is the application of NUFFT in each term in sum (17). Note that NUDFT is actually a generalization of DFT, a natural consideration is that whether the transformation matrix can be represented as the real part of a Hadamard product between a low-rank matrix and a DFT matrix. This is what we develop as follows.

Like $\mathcal{A}_n^{(a,b)}$, for modified uniform Jacobi polynomial transform, consider another matrix $\mathcal{B}_n^{(a,b)}$ whose (j,k) entry is defined via

$$M^{(a,b)}(t_j, k-1)\exp(i(\psi^{(a,b)}(t_j,k-1) - 2\pi\frac{[t_j n/2\pi]}{n}(k-1)))\sqrt{w_j}, \tag{18}$$

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

$$\mathcal{J}_n^{(a,b)} = \Re(\mathcal{B}_n^{(a,b)} \otimes \mathcal{F}_n) \tag{19}$$

where $\mathcal{F}_n$ is a matrix whose (j,k) entry is

$$\exp(i(2\pi\frac{[t_j n/2\pi]}{n}(k-1))). \tag{20}$$

From (20), it's obivious that $\mathcal{F}_n$ is a row permutation of inverse DFT matrix. Since the structure between $\mathcal{A}_n^{(a,b)}$ and $\mathcal{B}_n^{(a,b)}$ is similar, $\mathcal{B}_n^{(a,b)}$ is also a low-rank matrix. Since the entries of $\mathcal{B}_n^{(a,b)}$ are

accessible, we can apply SVD via randomized sampling to construct a low-rank approximation of $\mathcal{B}_n^{(a,b)}$. Of course, other decomposition methods are available, but we can obtain an optimal rank of the approximation via SVD, even SVD is sometimes a little more expensive than others.

By applying SVD via randomized sampling, we have a SVD of $\mathcal{B}_n^{(a,b)}$ up to a fixed accuracy ε; that is,

$$\mathcal{B}_n^{(a,b)} \approx USV, \tag{21}$$

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

$$\mathcal{B}_n^{(a,b)} \approx (US^{\frac{1}{2}})(S^{\frac{1}{2}}V) = u_1 v_1^T + \cdots + u_r v_r^T, \tag{22}$$

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 matrix transpose. This randomized method only visits $O(pr)$ columns and rows of $\mathcal{B}_n^{(a,b)}$ and hence only requires $O(rn)$ operations and memory [26], where p is a adaptive parameter. Once (22) obtained, then we have

$$f = W\mathcal{J}_n^{(a,b)}\hat{f} \approx W\Re(((\sum_{j=1}^{r} u_j v_j^T)\otimes \mathcal{F}_n)\hat{f}) = W\Re(\sum_{j=1}^{r} D_{u_j}\mathcal{F}_n D_{v_j}\hat{f}), \tag{23}$$

where D_u denotes a diagonal matrix with u on its diagonal. Formula (23) indicates that (9) can be computed by carrying out an optimal number of inverse FFTs, resulting in $O(rn\log n)$ arithmetic operations. Numerical experiments show that the rank in (23) is always smaller than that in (17). And in each term in the sum above, we just need to apply one inverse FFT, rather than one NUFFT, which is usually expensive than one inverse FFT.

Now we turn to inverse modified Jacobi polynomial transform. Note that the set $\{\tilde{P}_j^{(a,b)}\}_{j=0}^{\infty}$ is an orthonormal basis for $L^2(0,\pi)$. Therefore, the inverse transform can be computed via

$$\hat{f} = (W\mathcal{J}_n^{(a,b)})^T f \approx \Re(((\sum_{j=1}^{r} u_j v_j^T)\otimes \mathcal{F}_n)^T W^T f) = \Re(\sum_{j=1}^{r} D_{v_j}\mathcal{F}_n^T D_{u_j} W^T f), \tag{24}$$

where $\mathcal{F}_n^T$ is a permutation of DFT matrix. This formula indicates that the inverse transform can be computed by carrying out a optimal number of FFTs.

The algorithm here also works for modified nonuniform Jacobi polynomial transform, when the nonuniform locations are away from the endpoints of $(0,\pi)$. Note that the values of $\tilde{P}_\nu^{(a,b)}(t)$ at nonuniform locations and integral orders can be obtained through the method we mentioned in Section 2.1. In this case, the corresponding matrix $\mathcal{B}_n^{(a,b)}$ is also low rank. Here we don't prove a rigorous bound on the rank, but instead offer some numerical results in Section 3. Note that this algorithm doesn't work when some locations are close to the endpoints, since in this case, the corresponding matrix $\mathcal{B}_n^{(a,b)}$ might not be low-rank.

2.3 Multidimensional Jacobi Polynomial Transform

In this section, we extend our algorithm for two-dimensional and three-dimensional Jacobi polynomial transform. For sake of simplicity, we mainly consider two-dimensional Jacobi polynomial transform, the algorithm for the three-dimensional case would be described laconically. To evaluate two-dimensional Jacobi polynomial transform, we first express it as a Kronecker product of two one-dimensional Jacobi polynomial transforms, then in terms of the obtained approximations of these two one-dimensional Jacobi polynomial transform, we approximate two-dimensional Jacobi

polynomial transform by applying a sum of diagonally scaled (2D) DFT matrices. Note that the task of computing Jacobi polynomial transform is nearly equivalent to computing modified Jacobi polynomial, this fact clearly holds higher dimensional transform. For this, in the rest of the paper, we just take modified Jacobi polynomial transform into consideration.

Given a $n \times n$ tensor of coefficients $\hat{f}$, parameters $a, b \in (-\frac{1}{2}, \frac{1}{2})$, and locations $(t_i, s_j) \in (0, \pi) \times (0, \pi)$ for $i, j = 1, \cdots, n$, our aim is to efficiently evaluate the two dimensional modifed Jacobi polynomial transformation, i.e., an full tensor product as follows,

$$f(i,j) = \sum_{k=1}^{n} \sum_{\ell=1}^{n} \hat{f}(k,\ell) \tilde{P}_{k-1}^{(a,b)}(t_i) \tilde{P}_{\ell-1}^{(a,b)}(s_j), i, j = 1, \cdots, n, \tag{25}$$

where $\tilde{P}_{\nu}^{(a,b)}(t)$ is the modified order-ν Jocobi polynomial defined on $(0, \pi)$. The directly computation for (25) costs $O(n^4)$ operations, obviously.

Let $\mathcal{J}_n^{(a,b)}(s)$ denotes a one-dimensional transformation matrix like (10) with locations $s_1, \cdots, s_n$ instead, and so does $\mathcal{J}_n^{(a,b)}(t)$. There is a well-known fact that (25) is equal to a matvec,

$$vec(f) = (\mathcal{J}_n^{(a,b)}(t) \odot \mathcal{J}_n^{(a,b)}(s)) vec(\hat{f}), \tag{26}$$

where "$\odot$" denotes Kronecker product and $vec(\hat{f})$ is the vectorization of $\hat{f}$ such that $vec(\hat{f})((i-1)n + j) = \hat{f}(j, i)$ for all $i, j = 1, \cdots, n$. In terms of one dimensional approximation and two dimensional FFT, we can approximate (26) by a sum like (23). Let $\mathcal{B}_n^{(a,b)}(s)$ denotes a matrix like (18) in section 2.2 with locations $s_1, \cdots, s_n$ instead, and $\mathcal{F}_n(s)$ denotes a matrix like (20). After applying SVD via randomized sampling to $\mathcal{B}_n^{(a,b)}(s)$ to get a rank-r_s approximation, we have

$$\mathcal{J}_n^{(a,b)}(s) = \mathcal{B}_n^{(a,b)}(s) \odot \mathcal{F}_n(s) \approx \Re((\sum_{i=1}^{r_s} u_i(s) v_i^T(s)) \odot \mathcal{F}_n(s)) = \Re(\sum_{i=1}^{r_s} D_{u_i(s)} \mathcal{F}_n(s) D_{v_i(s)}), \tag{27}$$

where r_s is the numerical rank of $\mathcal{B}_n^{(a,b)}(s)$ and $u_i(s)$ and $v_i(s)$ are both column vectors in the SVD of $\mathcal{B}_n^{(a,b)}(s)$. So does $\mathcal{J}_n^{(a,b)}(t)$. Once the these approximations of $\mathcal{J}_n^{(a,b)}(s)$ and $\mathcal{J}_n^{(a,b)}(t)$ have been obtained, we insert them into (26) and have

$$\begin{aligned} vec(f) &\approx \Re\{[(\sum_{i=1}^{r_t} D_{u_i(t)} \mathcal{F}_n(t) D_{v_i(t)}) \odot (\sum_{j=1}^{r_s} D_{u_j(s)} \mathcal{F}_n(s) D_{v_j(s)})] vec(\hat{f})\} \\ &= \Re\{\sum_{i=1}^{r_t} \sum_{j=1}^{r_s} [((u_i(t) \odot u_j(s))(v_i(t) \odot v_j(s))^T) \odot (\mathcal{F}_n(t) \odot \mathcal{F}_n(s))] vec(\hat{f})\} \\ &= \Re\{\sum_{i=1}^{r_t} \sum_{j=1}^{r_s} D_{u_i(t) \odot u_j(s)} [\mathcal{F}_n(t) \odot \mathcal{F}_n(s)] D_{v_i(s) \odot v_j(t)} vec(\hat{f})\}, \end{aligned} \tag{28}$$

where the second and third equality signs come from the basic properties of Kronecker product. Note that since both $\mathcal{F}_n(s)$ and $\mathcal{F}_n(t)$ are row permutations of inverse DFT matrix, apply $\mathcal{F}_n(t) \odot \mathcal{F}_n(s)$ to a $n^2 \times 1$ vector is equivalent to first apply 2D inverse FFT to the vector and then permutate the transformed vector, which totally costs $O(n^2 \log n)$ operations. Since $D_{u_i(t) \odot u_j(s)}$ and $D_{v_i(s) \odot v_j(t)}$ are $n^2 \times n^2$ diagonal matrices, the total computational complexity is of $O(r_t r_s n^2 \log n)$ if we use formula (28) to approximate the two dimensional modifed Jacobi polynomial transform (25). When the locations are away from the edges in $(0, \pi) \times (0, \pi)$, numerical results in Section 3 shows that

the ranks r_t and r_s are quite small even n is very large, thus the totally complexity is nearly of $O(n^2 \log n)$.

Analogically, the three dimensional modifed Jacobi polynomial transformation

$$f(i,j,m) = \sum_{g=1}^{n}\sum_{k=1}^{n}\sum_{\ell=1}^{n} C(g,k,\ell)\tilde{P}_{g-1}^{(a,b)}(x_i)\tilde{P}_{k-1}^{(a,b)}(y_j)\tilde{P}_{\ell-1}^{(a,b)}(z_m), i,j,m = 1,\cdots,n, \tag{29}$$

where $C \in \mathbb{R}^{n\times n\times n}$ and x, y, z are locations in $(0,\pi)$, can be approximated through applying a sum of diagonally scaled (3D) DFT matrices like formula (28), which takes $O(n^3 \log n)$ operations totally. The details in this case is left to the reader.

Again, it's worthy to emphasize that the reduce of the rank of the low-rank matrix, i.e., the number of higher dimensional FFTs needed to be applied, compared to the counterpart in [25], is much more prominent than that of the one-dimensional case. Analogously, applying higher dimensional FFT is much cheaper than higher dimensional NUFFT, the reduce of total computational complexity is more remarkable. For numerical results, please see Section 3.

$\log N$	r_A	r_B	A_err	B_err
7	17	15	6.31E-12	6.87E-13
8	20	17	6.06E-12	5.63E-13
9	21	18	5.16E-12	5.31E-13
10	24	20	3.86E-12	4.80E-13
11	25	21	4.94E-12	1.01E-12
12	30	23	6.69E-12	1.15E-12
13	31	26	1.59E-11	3.09E-12
14	33	28	2.60E-11	5.91E-12
15	34	30	1.04E-10	1.24E-11
16	36	31	1.19E-10	7.05E-11
17	37	31	2.82E-10	4.29E-11
18	39	34	6.31E-10	1.77E-10
19	41	34	1.11E-09	3.90E-10

Table 1: In the experiment, the parameter a = 0, b = 0; r_A and r_B are the ranks of $\mathcal{A}_n^{(a,b)}$ and $\mathcal{B}_n^{(a,b)}$ respectively. And A_err is the relative error between (**??**) and the direct computation of Jacobi transform (9), B_err is the relative error between (23) and the direct method.

3 Numerical results

up to continue...

3.1 SVD via Randomized Sampling

For a general matrix Z whose individual entries are accessible, the following randomized sampling method for low-rank approximations introduced in [26, 27] can be efficient. This method only visits $O(r)$ columns and rows of Z and hence only requires $O(r(m+n))$ operations and memory.

Here, we adopt the standard notation for a submatrix: 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

alsouse ":" 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 sampling algorithm to construct a rank-r approximation of $Z \approx U_0 \Sigma_0 V_0^*$.

Algorithm 3.1. *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*

$$Z_{I,:}P = QR, \tag{30}$$

   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*

$$PZ_{:,J} = LQ, \tag{31}$$

   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*

$$\min_M \parallel Z_{I;J}(Q_{col})_{I;:} M (Q^*_{row})_{:;J} \parallel, \tag{32}$$

   gives $M = (Q_{col})^\dagger_{I;:} Z_{I;J} (Q^*_{row})^\dagger_{:;J}$, 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*

$$U_0 = Q_{col} U_M; \Sigma_0 = \Sigma_M; V_0 = V_M^* Q_{row}^*. \tag{33}$$

In the typical implementation, the multiplicative oversampling parameter q is equal to 3 and Steps 2 and 3 are iterated no more than three times. Though we have not been able to quantify the error and success probability rigorously for this procedure at this point, [27] shows that these parameters are empirically sufficient to achieve accurate low-rank approximations. Note that our goal is to construct a low-rank approximation of the low-rank matrix in our algorithm up to a fixed relative error ε, rather than a fixed rank. This process can also be embedded into an iterative process to achieve the desired accuracy.

References

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

[2] A. Townsend and L. Trefethen. An extension of chebfun to two dimensions. *SIAM Journal on Scientific Computing*, 35(6):C495–C518, 2013.

[3] B. Hashemi and L. Trefethen. Chebfun in three dimensions. *SIAM Journal on Scientific Computing*, 39(5):C341–C363, 2017.

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

[5] 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.

[6] 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.

[7] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

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

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

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

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

[12] 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.

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

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

[15] Jens Keiner. *Fast Polynomial Transforms*. Logos Verlag Berlin GmbH, 2011.

[16] Raf Vandebril, Marc Van Barel, and Nicola Mastronardi. Matrix computations and semiseparable matrices, volume ii: Eigenvalue and singular value methods, 2008.

[17] L Gemignani. Matrix computations and semiseparable matrices. volume i: Linear systems, r. vandebril, m. van barel, n. mastronardi, 2010.

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

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

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

[21] Ernst Eduard Kummer. De generali quadam aequatione differentiali tertii ordinis. *Journal für die reine und angewandte Mathematik*, 100:1–9, 1887.

[22] 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.

[23] E. Candès, L. Demanet, and L. 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 transform: When to use nufft or butterfly factorization? *arXiv preprint arXiv:1803.04128*, 2018.

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

[26] 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.

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