Journal of Computational Physics 388 (2019) 103–122

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A unified framework for oscillatory integral transforms: When to use NUFFT or butterfly factorization?

Haizhao Yang

Department of Mathematics, National University of Singapore, Singapore

ARTICLE INFO

Article history:
Received 9 October 2018
Received in revised form 25 February 2019
Accepted 26 February 2019
Available online 14 March 2019

Keywords:
Non-uniform fast Fourier transform
Butterfly factorization
Randomized algorithm
Matrix completion
Fourier integral operator
Special function transform

ABSTRACT

This paper concerns the fast evaluation of the matvec $g = Kf$ for $K \in \mathbb{C}^{N\times N}$, which is the discretization of an oscillatory integral transform $g(x) = \int K(x,\xi) f(\xi) d\xi$ with a kernel function $K(x,\xi) = \alpha(x,\xi) e^{2\pi i \Phi(x,\xi)}$, where $\alpha(x,\xi)$ is a smooth amplitude function, and $\Phi(x,\xi)$ is a piecewise smooth phase function with $O(1)$ discontinuous points in x and ξ. A unified framework is proposed to compute Kf with $O(N\log N)$ time and memory complexity via the non-uniform fast Fourier transform (NUFFT) or the butterfly factorization (BF), together with an $O(N)$ fast algorithm to determine whether NUFFT or BF is more suitable. This framework works for two cases: 1) explicit formulas for the amplitude and phase functions are known; 2) only indirect access of the amplitude and phase functions are available. Especially in the case of indirect access, our main contributions are: 1) an $O(N\log N)$ algorithm for recovering the amplitude and phase functions is proposed based on a new low-rank matrix recovery algorithm; 2) a new stable and nearly optimal BF with amplitude and phase functions in a form of a low-rank factorization (IBF-MAT[1]) is proposed to evaluate the matvec Kf. Numerical results are provided to demonstrate the effectiveness of the proposed framework.

1. Introduction

Oscillatory integral transforms have been an important topic for scientific computing. After discretization with N grid points in each variable, the integral transform is reduced to a dense matrix-vector multiplication (matvec) $g = Kf$. The direct computation of the matvec takes $O(N^2)$ operations and is prohibitive in large-scale computation, which has motivated an active research line in developing nearly linear matvec. The most famous example is the fast Fourier transform (FFT) [38] that evaluate the integral:

$$\widehat{f}(\xi) = \int_0^1 K(x,\xi) f(x) dx = \int_0^1 e^{-2\pi i x\xi} f(x) dx \tag{1}$$

via the discretization

E-mail addresses: matyh@nus.edu.sg, haizhaoyang@yahoo.com.

[1] IBF-MAT means interpolative butterfly factorization based on low-rank matrix information instead of explicit formulas used in the IBF in [22].

https://doi.org/10.1016/j.jcp.2019.02.044
0021-9991/

Table 1
Summary of existing algorithms and the proposed algorithms (in bold) for the evaluation of Kf when amplitude and phase have explicit formulas. Although the BF in [9] requires no precomputation and $O(N)$ memory, it is a few times slower than the BF in [22] regarding the application time. Hence, we adopt the scaling of [22] in this paper.

Kernels $K(x,\xi)$	Algorithms	Precomputation time	Application time	memory
$\alpha(x,\xi)e^{2\pi \mathrm{i} p(x)q(\xi)}$	NUFFT [1,34]	$O(N)$	$O(N\log N)$	$O(N)$
$\alpha(x,\xi)e^{2\pi \mathrm{i}\Phi(x,\xi)}$	**NUFFT**	$O(N)$	$O(N\log N)$	$O(N)$
$\alpha(x,\xi)e^{2\pi \mathrm{i}\Phi(x,\xi)}$	BF [28,9,22]	$O(N\log N)$	$O(N\log N)$	$O(N\log N)$

Table 2
Summary of existing algorithms and the proposed algorithms (in bold) for the evaluation of Kf for a general kernel $\alpha(x,\xi)e^{2\pi \mathrm{i}\Phi(x,\xi)}$ when only the indirect access of amplitude and phase is available according to different scenarios listed in Table 3. A remaining open problem was in Scenario 2.

Scenarios	Algorithms	Precomputation time	Application time	memory
Scenario 1	BF [28]	$O(N\log N)$	$O(N\log N)$	$O(N\log N)$
Scenario 2	BF [23,26]	$O(N^{1.5}\log N)$	$O(N\log N)$	$O(N\log N)$
Scenario 3	BF [9,22]	$O(N\log N)$	$O(N\log N)$	$O(N\log N)$
All scenarios	**NUFFT** or **IBF-MAT**	$O(N\log N)$	$O(N\log N)$	$O(N\log N)$

$$\widehat{f}(\xi_i) = \frac{1}{N}\sum_{x_j} e^{-2\pi i x_j \xi_i} f(x_j), \quad i,j = 1,2,\ldots,N, \tag{2}$$

with $\{x_i\}$ and $\{\xi_j\}$ as uniformly distributed points in $[0,1)$ and $[-N/2, N/2)$ following

$$x_i = (i-1)/N \text{ and } \xi_j = j - 1 - N/2.$$

The matvec in (2) has a dense matrix $K \in \mathbb{C}^{N\times N}$ with the (i,j)-th entry as $\frac{1}{N}e^{-2\pi i x_j \xi_i}$ and can be evaluated with $O(N\log(N))$ operations using the FFT algorithm [38]. In the case of non-uniform distributed points $\{x_i\}$ and $\{\xi_j\}$, the non-uniform FFT (NUFFT) algorithms in [15,34] are able to evaluate (2) with $O(N\log(N))$ operations based on FFT. For a kernel function $K(x,\xi) = e^{2\pi \mathrm{i} p(x)q(\xi)}$ with either uniform or nonuniform $\{x_i\}$ and $\{\xi_j\}$, the transformation in (1) is an NUFFT. Nearly linear scaling matvec for more general kernel functions have been proposed either based on the similarity of K to the Fourier matrix in (2) [1,34], i.e., $K(x,\xi) = \alpha(x,\xi)e^{2\pi \mathrm{i} p(x)q(\xi)}$, or based on the complementary low-rank structure of K [12,20,22,23,25,28,30,35,37] when the phase function is not in a form of separation of variables.

The main ideas of existing algorithms are as follows. After computing the low-rank approximation of $\alpha(x,\xi) \approx \sum_{k=1}^{r} a_k(x)b_k(\xi)$, we have

$$g(x) \approx \sum_{k=1}^{r} a_k(x) \int e^{2\pi \mathrm{i}\Phi(x,\xi)} (b_k(\xi) f(\xi))\, d\xi. \tag{3}$$

If $K(x,\xi) = \alpha(x,\xi)e^{2\pi \mathrm{i} p(x)q(\xi)}$, then

$$g(x) \approx \sum_{k=1}^{r} a_k(x) \int e^{2\pi \mathrm{i} p(x)q(\xi)} (b_k(\xi) f(\xi))\, d\xi$$

can be evaluated through r NUFFT's. If the phase function $\Phi(x,\xi)$ is not of the form $p(x)q(\xi)$, then the butterfly factorization (BF) [23,28,30] of $e^{2\pi \mathrm{i}\Phi(x,\xi)}$ is computed and the dense matrix $e^{2\pi \mathrm{i}\Phi(x,\xi)}$ can be factorized as a product of $O(\log(N))$ sparse matrices, each of which has only $O(N)$ non-zero entries. Hence, storing and applying $e^{2\pi \mathrm{i}\Phi(x,\xi)}$ via the BF and (3) take only $O(rN\log(N))$ complexity. In sum, after precomputation (low-rank factorization in (3), and BF,[2] if needed), both kinds of algorithms admit $O(N\log N)$ computational complexity for applying K to a vector f. However, existing algorithms are efficient only when the explicit formulas of the kernel is known (see Table 1 and 2 for a detailed summary). The computational challenge in the case of indirect access of the kernel function (see Table 3 for a list of different scenarios) motivates a series of new algorithms in this paper.

This paper proposes an $O(N\log N)$ unified framework for evaluating Kf either based on NUFFT or BF (see Fig. 1 for the main computational flowchart of the unified framework). This framework considers possibly most application scenarios of oscillatory integral transforms. We also briefly discuss how to choose NUFFT or BF to maximize the computational efficiency according to several factors (e.g., accuracy and rank parameters in low-rank factorization, the number of vectors in the matvec) in a serial computational environment. The unified framework works in two cases: 1) explicit formulas for the amplitude and phase functions are known; 2) only indirect access of the amplitude and phase functions are available. When

[2] In most applications, K is applied to multiple vectors f's. Hence, it is preferable to save the results of expensive computational routines that are independent of the input vectors f's for later applications.

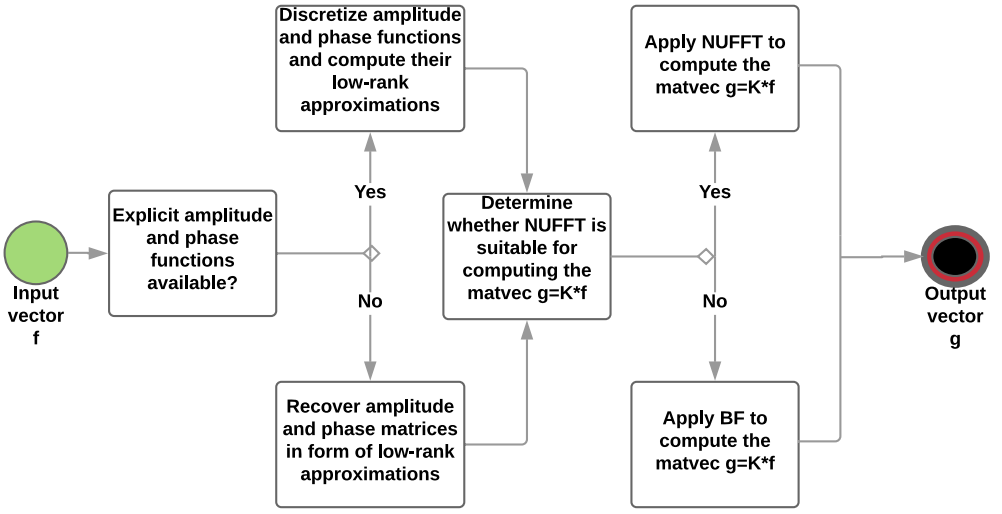

Fig. 1. The computational flowchart of the unified framework using NUFFT or BF. The framework consists of three main steps: 1) construct the low-rank approximations of the amplitude and phase matrices; 2) determine whether NUFFT is applicable; 3) apply NUFFT or BF. When the numerical rank of the phase function r_ϵ is only larger than the dimension of the problem by one or two, NUFFT is usually faster than BF and hence it will be applied to compute Kf.

Table 3
Three scenarios of the indirect access of the amplitude and phase functions.

Scenario 1:	There exists an algorithm for evaluating an arbitrary entry of the kernel matrix in $O(1)$ operations [4,3,23,30].
Scenario 2:	There exist an $O(N\log N)$ algorithm for applying K and its transpose to a vector [17,23,26,33].
Scenario 3:	The amplitude and the phase functions are solutions of partial differential equations (PDE's) [12]. $O(1)$ columns and rows of the amplitude and phase matrices are available by solving PDE's.

the explicit formulas are given, computing Kf is relatively simple. Hence, we only focus on the case of indirect access. To the best of our knowledge, the most common indirect access can be summarized into three scenarios in Table 3.

As the first main contribution of this paper, in the case of indirect access, a nearly linear scaling algorithm is proposed to recover the amplitude and phase matrices in a form of low-rank matrix factorization. In scientific computing, there are several important problems requiring an approximate or precise construction of the low-rank amplitude and phase matrices, e.g., special function transforms [4,3], the compositions of Fourier integral operators (FIO's) as a preconditioner for certain classes of parabolic and hyperbolic equations [21,32,33], etc. In signal and image processing, amplitude and phase matrix recovery is also called the phase unwrapping problem [10,29,36]. As far as we know, this paper is the first to study the low-rank matrix recovery problem aiming at a nearly linear scaling algorithm. Previous algorithms [10,29,36] require computation with all the matrix entries and hence take at least $O(N^2)$ operations.

As the second main contribution, when the low-rank amplitude and phase matrices have been recovered, a new BF (named as IBF-MAT for short) is proposed for the matvec Kf. IBF-MAT is the first BF for the matvec Kf with $O(N\log N)$ complexity for both precomputation and application in the case of indirect access (see Table 2 for the comparison with existing algorithms).

Finally, this paper shows that if the numerical rank of $\Phi(x,\xi)$ is r_ϵ (depending on an ϵ accuracy parameter), an r_ϵ-dimensional NUFFT can be applied to evaluate (3) in $O(N\log N)$ operations. The dimension of the NUFFT, r_ϵ, could be larger than the dimension of the variables x and ξ, and hence we consider it as a dimension lifting technique. This new method significantly extends the application range of the NUFFT approach for computing Kf. Applications of this kind include many special function transforms, e.g. the Jacobi polynomial [6] and the Bessel function [4] permit phase functions with one dominant term in the form of separation of variables asymptotically. It is also worth mentioning that the prefactor implicit in the NUFFT grows roughly exponentially in the dimension r_ϵ. In the case when r_ϵ is large, the dimension lifting technique would be less attractive than other methods.

The rest of the paper is organized as follows. In Section 2, we revisit existing low-rank factorization techniques and propose our new low-rank matrix factorization in the case of indirect access. In Section 3, we introduce the new NUFFT approach by dimension lifting. In Section 4, we introduce the IBF-MAT. Finally, we provide several numerical examples to demonstrate the efficiency of the proposed unified framework in Section 5. Throughout this paper, we adopt MATLAB notations for the algorithm description for simplicity: given row and column index sets I and J, $K_{I,J} = K(I,J)$ 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 matrix factorization

This section is for the first main step in the unified framework as shown in Fig. 1: low-rank matrix factorizations of the amplitude and phase matrices.

2.1. Existing low-rank matrix factorization

Low-rank approximation by randomized sampling

For $K \in \mathbb{C}^{m\times n}$, we define a rank-$r$ approximate singular value decomposition (SVD) of K as

$$K \approx U_0 \Sigma_0 V_0^*, \tag{4}$$

where $U_0 \in \mathbb{C}^{m\times r}$ is orthogonal, $\Sigma_0 \in \mathbb{R}^{r\times r}$ is diagonal, and $V_0 \in \mathbb{C}^{n\times r}$ is orthogonal, and $r = O(1)$. Efficient randomized tools have been proposed to compute approximate SVDs for numerically low-rank matrices [14,18]. The one in [14] is more attractive because it only requires $O(m+n)$ operations and memory: $O(1)$ randomly sampled rows and columns of K are sufficient for construction (4).

Interpolative low-rank approximation

Randomized SVD is sufficiently efficient if we allow a linear complexity to construct the low-rank approximation. However, to construct the BF in nearly linear operations, we cannot even afford linear scaling low-rank approximations; we can only afford an algorithm that provides the low-rank factors with explicit formulas. This motivates the interpolative low-rank approximation below.

Let us focus on the case of a kernel function $K(x,\xi) = e^{2\pi \mathrm{i}\Phi(x,\xi)}$ and its discretization $K = e^{2\pi \mathrm{i}\Phi} \in \mathbb{C}^{N_A\times N_B}$ to introduce the interpolative low-rank approximation. We assume that x and ξ are one-dimensional variables and the algorithm below can be easily generalized to higher dimensional cases by tensor products. Note that if the phase function is given in a form of separation of variables, i.e., $\Phi(x,\xi) = \sum_{k=1}^{r} u_k(x)v_k(\xi)$, the following interpolative factorization will also work with a minor modification.

Let A and B denote the sets of contiguous row and column indices of K. If $A\times B$ corresponds to a small two-dimensional interval in the variables $x\times\xi$, then a low-rank approximation

$$K(A,B) = e^{2\pi \mathrm{i}\Phi(A,B)} \approx U_0 V_0^*$$

exists and can be constructed via Lagrange interpolation following a discrete version of the algorithm in [9] as follows.

Suppose the numbers of elements in A and B are N_A and N_B, respectively. Let

$$R(A,B) := \Phi(A,B) - ones(N_A,1) * \Phi(c_A,B) - \Phi(A,c_B) * ones(1,N_B) + \Phi(c_A,c_B), \tag{5}$$

where c_A and c_B are the indices of A and B closest to the mean of all indices in A and B, respectively, then K can be written as

$$K(A,B) = e^{-2\pi \mathrm{i}\Phi(c_A,c_B)} * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(A,c_B)}\right) * e^{2\pi \mathrm{i}R(A,B)} * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(c_A,B)}\right). \tag{6}$$

Hence, the low-rank approximation of $e^{2\pi \mathrm{i}R(A,B)}$ immediately gives the low-rank approximation of $K(A,B)$. A Lagrange interpolation can be applied to construct the low-rank approximation of $e^{2\pi \mathrm{i}R(A,B)}$.

Recall the challenge that we may not have explicit formulas for the amplitude or phase functions. Hence, we cannot use Chebyshev grid points in the Lagrange interpolation to maintain a small uniform error as the previous BF in [9,22] does. Therefore, we choose indices in A or B in a similar manner like Mock-Chebyshev points,[3] because both the Chebyshev grid points and the Mock-Chebyshev grid points have almost the same numerical performance when A and B correspond to equispaced grid points in the discretization of the kernel function K according to [2,19]. For the interest of the reader, we refer to Fig. 7 in [2] for the numerical comparison between the original Chebyshev interpolation and the Mock-Chebyshev interpolation on equispaced grid points. In the case of highly non-equispaced grid points, a stable Mock-Chebyshev interpolation algorithm is still under development.

Let us assume $A = \{1,\dots,N_A\}$ and $B = \{1,\dots,N_B\}$. If an index set doesn't start with the index 1, we can simply shift the grid points accordingly. For a fixed integer r, the Chebyshev grid of order r on $[-\frac{1}{2},\frac{1}{2}]$ is defined by

$$\left\{ z_t = \frac{1}{2}\cos\left(\frac{(t-1)\pi}{r-1}\right)\right\}_{1\le t\le r}.$$

[3] Though it was shown in [31] that no fast stable approximation of analytic functions from equispaced samples in a bounded interval in the sense of L^∞-norm with an exponential convergence rate is available, the Mock-Chebyshev points admit polynomial interpolation with a root-exponential convergence rate. In this paper, we care more about the approximation error at the equispaced sampling locations, in which case it is still unknown whether the Mock-Chebyshev points admit an exponential convergence rate.

A grid adapted to the index set A is then defined via shifting, scaling, and rounding as

$$\{x_t\}_{t=1,\dots,r} = \left\{ \text{Round}\left(t + (N_A - r)(z_t + \frac{1}{2}) \right) \right\}_{t=1,\dots,r}. \tag{7}$$

Note that the rounding operator may result in repeated grid points. Only one grid point will be kept if repeated. Similarly, a grid adapted to the index set B is defined as

$$\{\xi_t\}_{t=1,\dots,r} = \left\{ \text{Round}\left(t + (N_B - r)(z_t + \frac{1}{2}) \right) \right\}_{t=1,\dots,r}. \tag{8}$$

Given a set of indices $\{x_t\}_{t=1,\dots,r}$ in A, define Lagrange interpolation polynomials $M_t^A(x)$ by

$$M_t^A(x) = \prod_{1\le j\le r, j\neq t} \frac{x - x_j}{x_t - x_j}.$$

Similarly, M_t^B is denoted as the Lagrange interpolation polynomials for B.

Now we are ready to construct the low-rank approximation of $e^{2\pi \mathrm{i} R(A,B)}$ by interpolation:

- when we interpolate in ξ, the low-rank approximation of $e^{2\pi \mathrm{i} R(A,B)}$ is given by

$$e^{2\pi \mathrm{i} R(A,B)} \approx U_0 V_0^*, \tag{9}$$

where

$$U_0 = \left(e^{2\pi \mathrm{i} R(A,\xi_1)}, \dots, e^{2\pi \mathrm{i} R(A,\xi_r)} \right) \in \mathbb{C}^{N_A \times r},$$
$$V_0 = \left((M_1^B(B))^*, \dots, (M_r^B(B))^* \right) \in \mathbb{C}^{N_B \times r},$$

and each $M_t^B(B)$ denotes a row vector of length N_B such that the k-th entry is

$$M_t^B(\xi_k) = \prod_{1\le j\le r, j\neq t} \frac{\xi_k - \xi_j}{\xi_t - \xi_j}$$

for $\xi_k \in B$, $k = 1, \dots, N_B$, given by (8).

- when we interpolate in x, the low-rank approximation of $e^{2\pi \mathrm{i} R(A,B)}$ is

$$e^{2\pi \mathrm{i} R(A,B)} \approx U_0 V_0^*, \tag{10}$$

where

$$U_0 = \left((M_1^A(A))^*, \dots, (M_r^A(A))^* \right) \in \mathbb{C}^{N_A \times r},$$
$$V_0 = \left(\left(e^{2\pi \mathrm{i} R(x_1,B)} \right)^*, \dots, \left(e^{2\pi \mathrm{i} R(x_r,B)} \right)^* \right) \in \mathbb{C}^{N_B \times r},$$

and each $M_t^A(A)$ denotes a row vector of length N_A such that the k-th entry is

$$M_t^A(x_k) = \prod_{1\le j\le r, j\neq t} \frac{x_k - x_j}{x_t - x_j}$$

for $x_k \in A$, $k = 1, \dots, N_A$, given by (7).

Finally, we are ready to construct the low-rank approximation for the matrix $e^{2\pi \mathrm{i} \Phi(A,B)}$ when we have $\Phi(A, B)$ or equivalently a low-rank factorization of $\Phi(A, B)$ as in Algorithm 1.

2.2. New low-rank matrix factorization with indirect access

This section introduces a nearly linear scaling algorithm for constructing the low-rank factorization of the phase matrix $\Phi \in \mathbb{R}^{N\times N}$ when we only know the kernel matrix $K = e^{2\pi \mathrm{i}\Phi}$ through Scenarios 1 and 2 in Table 3. The main idea is to recover $O(1)$ randomly selected columns and rows of Φ from the corresponding columns and rows of $K = e^{2\pi \mathrm{i}\Phi}$. Then by the randomized SVD in Section 2.1, we can construct the low-rank factorization of Φ.

Obtaining $O(1)$ randomly selected columns and rows of K is simple in Scenarios 1 and 2: we can directly evaluate them in Scenario 1; we apply the kernel matrix K and its transpose to $O(1)$ randomly selected natural basis vectors in $\mathbb{R}^N$ to obtain the columns and rows.

1 Input: The phase matrix $\Phi \in \mathbb{C}^{N\times N}$ or its low-rank factorization $\Phi = \bar{U}\bar{V}^*$. Contiguous index sets A and B of the row and column indices of Φ, respectively. A rank parameter r.
2 Output: The low-rank factorization UV^* such that $UV^* \approx e^{2\pi \mathrm{i}\Phi(A,B)}$, where $U \in \mathbb{C}^{N_A\times r}$, and $V \in \mathbb{C}^{N_B\times r}$, where N_A is the number of elements in A and N_B is for B.
3 **if** *the input contains low-rank factors* $\bar{U}$ *and* $\bar{V}$ *of* Φ **then**
4 define a function to evaluate an arbitrary entry of Φ at the position (m,n) in $O(1)$ operations as follows

$$\Phi(m,n) = \bar{U}(m,:)\bar{V}(n,:)^*.$$

5 **if** *interpolation in the variable* ξ *in* B **then**
6 by (6) and (9), we have

$$U := e^{-2\pi \mathrm{i}\Phi(c_A,c_B)} * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(A,c_B)}\right) * U_0, \quad V^* := V_0^* * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(c_A,B)}\right), \tag{11}$$

where U_0 and V_0 are given just below (9).
7 **if** *interpolation in the variable* x *in* A **then**
8 by (6) and (10), we have

$$U := e^{-2\pi \mathrm{i}\Phi(c_A,c_B)} * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(A,c_B)}\right) * U_0, \quad V^* := V_0^* * \operatorname{diag}\left(e^{2\pi \mathrm{i}\Phi(c_A,B)}\right), \tag{12}$$

where U_0 and V_0 are given just below (10).

Algorithm 1: Interpolative low-rank approximation for one-dimensional kernel $e^{2\pi \mathrm{i}\Phi(x,\xi)}$. Factorization in higher dimensions can be constructed similarly via tensor products.

However, reconstructing the corresponding columns and rows of Φ from those of $K = e^{2\pi \mathrm{i}\Phi}$ is more challenging. The difficulty comes from the fact that

$$\frac{1}{2\pi}\Im\left(\log\left(K(i,j)\right)\right) = \frac{1}{2\pi}\Im\left(\log\left(e^{2\pi \mathrm{i}\Phi(i,j)}\right)\right) = \frac{1}{2\pi}\arg\left(e^{2\pi \mathrm{i}\Phi(i,j)}\right) = \mod(\Phi(i,j),1),$$

where $\Im(\cdot)$ returns the imaginary part of the complex number, and $\arg(\cdot)$ returns the argument of a complex number. Hence, Φ is only known up to modular 1.

Fortunately, our main purpose is not to recover the exact Φ that generates K; instead, we are interested in a low-rank matrix Ψ such that

$$\mod(\Psi,1) = \frac{1}{2\pi}\Im\left(\log(K)\right). \tag{13}$$

Based on the smoothness of the phase function, a TV^3-norm[4] minimization technique is proposed to recover the columns and rows of Φ up to an additive error matrix E that is numerically low-rank, i.e., the TV^3-norm minimization technique returns a matrix $\Psi = \Phi + E$ such that $e^{2\pi \mathrm{i}\Psi} = e^{2\pi \mathrm{i}\Phi}$ and E is numerically low-rank.

To be more rigorous, we look for the solution of the following combinatorial constrained TV^3-norm minimization problem:

$$\begin{aligned} &\min_{\Phi\in\mathbb{R}^{N\times N}} \sum_{i\in\mathcal{R}} \|\Phi(i,:)\|_{TV^3} + \sum_{j\in\mathcal{C}} \|\Phi(:,j)\|_{TV^3} \\ &\text{subject to} \quad \mod(\Phi(i,j),1) = \frac{1}{2\pi}\Im\left(\log\left(K(i,j)\right)\right) \\ &\qquad\qquad \text{for } i\in\mathcal{R} \text{ or } j\in\mathcal{C}, \end{aligned} \tag{14}$$

where $\mathcal{C}$ and $\mathcal{R}$ are column and row index sets with $O(1)$ randomly selected indices, respectively.

The problem addressed here is similar to phase retrieval problems, but has a different setting to existing phase retrieval applications and different aims in numerical computation. Phase retrieval problems usually have sparsity assumptions on the signals (or after an appropriate transformation) that lose phases. In the problem considered in this paper, $e^{2\pi \mathrm{i}\Phi}$ is dense and might not be sparse after a transformation (e.g., the Fourier transform or wavelet transform). Furthermore, there are only $O(N)$ samples of the target matrix of size $N\times N$ to be recovered and the hard constrain (14) is preferred instead of treating it as a soft constraint. TV^1-norm is a useful tool for regularization in phase retrieval problems; however, TV^3-norm is preferred in this paper since, for example, $\{\Phi(x,y)+ax+by\}_{a,b\in\mathbb{Z}}$ are good solutions to obtain the low-rank factorization of the phase function, and it is not necessary to pick up one function among $\{\Phi(x,y)+ax+by\}_{a,b\in\mathbb{Z}}$ with the minimum

[4] The TV^3-norm of a vector $v\in\mathbb{R}^N$ is defined as $\|v\|_{TV^3} := \sum_{i=2}^{N-2} |v_{i+1}+v_{i-1}-2v_i-(v_{i+2}+v_i-2v_{i+1})|$ in this paper. Similarly, The TV^1-norm of a vector $v\in\mathbb{R}^N$ is defined as $\|v\|_{TV^1} := \sum_{i=2}^{N} |v_i - v_{i-1}|$. The TV^2-norm of a vector $v\in\mathbb{R}^N$ is defined as $\|v\|_{TV^2} := \sum_{i=2}^{N-1} |v_{i+1}+v_{i-1}-2v_i|$.

TV^1-norm using much extra effort. TV^3-norm minimization leave us much more flexibility to obtain an approximately good solution to (13) quickly.

Our goal here is an $O(N)$ algorithm for solving the matrix recovery problem in (14). Though there have been many efficient algorithms for phase retrieval problems, they usually require computational cost at least $O(nN^2)$, where N^2 is the size of the target and n is the number of iterations. n and N^2 are both too large to be applied to our problem. Hence, instead of solving (14) exactly using advanced optimization techniques, we propose a heuristic fast algorithm to identify a reasonably good approximate solution to (14). As we can see in numerical examples, the proposed heuristic algorithm works well in most applications.

A heuristic solution of the TV^3-norm minimization is to use the columns and rows of $\frac{1}{2\pi}\Im(\log(K))$ to identify smooth columns and rows of Ψ agreeing with (13) and satisfying the following conditions:

1. the variation of these columns and rows of Ψ is small;
2. recovered columns and rows share the same value at the intersection.

Let us start with an example of vector recovery with TV^3-norm minimization to motivate the algorithm for matrix recovery:

$$\begin{aligned} &\min_{v\in\mathbb{R}^N} \quad \|v\|_{TV^3} \\ &\text{subject to} \quad \mod(v,1) = \frac{1}{2\pi}\Im(\log(k)), \end{aligned} \tag{15}$$

where $k\in\mathbb{R}^N$ is a given vector. The discussion below will be summarized in Algorithm 2 and visualized in Fig. 2.

First, we assume that k is a vector from the discretization of $e^{2\pi i\phi(\xi)}$ with a smooth function $\phi(\xi)$. To simplify notations, let $u=\frac{1}{2\pi}\Im(\log(k))$, then we would like to identify a smooth v from a given vector u. It is easy to check that the solution to (15) is not unique. Fortunately, we can empirically identify v using the following steps:

1. Set $v(1)=u(1)+m$ for an arbitrary integer m.
2. Identify $v(2)$ via minimizing $|v(2)-v(1)|$ such that $\mod(v(2),1)=u(2)$ (corresponding to Line 6-9 in Algorithm 2).
3. Identify $v(3)$ via minimizing $|(v(2)-v(1))-(v(3)-v(2))|$ such that $\mod(v(3),1)=u(3)$ (corresponding to Line 10-13 in Algorithm 2).
4. For each $i=4,5,\ldots,N$, set $v(i)$ via minimizing

$$|(v(i-1)+v(i-3)-2v(i-2))-(v(i)+v(i-2)-2v(i-1))|$$

such that $\mod(v(i),1)=u(i)$ (corresponding to Line 16 in Algorithm 2).

Second, in a more general case when k is a vector from the discretization of $e^{2\pi i\phi(\xi)}$ with a piecewise smooth function $\phi(\xi)$. Suppose

$$\mathcal{S}=\{c_1,c_2,\ldots,c_n\}$$

is an index set indicating the discontinuity locations of $\phi(\xi)$ with $c_1=1<c_2<\cdots<c_n<N$. Since we assume that $\phi(\xi)$ is discontinuous at the end points of its domain, we let $c_1=1$. By applying the procedures just above to each piece $v(c_i:c_{i+1})$ for $i=1,\ldots,n$, we can empirically identify v. Since there is no prior information about $\mathcal{S}$ except that we know $c_1=1\in\mathcal{S}$, Algorithm 2 automatically determines the discontinuous locations in Line 17-19 according to a threshold τ: when the second derivative of v at a certain location is larger than τ, we consider v is discontinuous at this location.

Recall the goal of matrix recovery in (13). It is not necessary to tune the parameter τ such that the discontinuous locations are exactly identified. If Algorithm 2 misses some discontinuous locations, Algorithm 3 will provide a smoother estimation of the phase matrix; if Algorithm 2 artificially detects $O(1)$ fake discontinuous locations, Algorithm 3 will provide an estimation of the phase matrix with more pieces of smooth domains. As long as (13) is satisfied, all these estimations are satisfactory. The correct τ depends on the phase function and is not known a priori. In practice, one can specify a small τ and it takes $O(N)$ operations to obtain the corresponding discontinuous points; if the number of discontinuous points is too large, then $\frac{1}{2}\tau$ is used to identify discontinuous points; this procedure is repeated until desired discontinuous points have been detected. The total cost to obtain a reasonably good τ is at most $O(N\log(N))$. In our numerical tests, τ is set to be $\frac{\pi}{2}$ for all numerical examples. Other values of τ result in similar numerical results.

When the vector recovery algorithm in Algorithm 2 is ready, we apply it to design a matrix recovery algorithm in Algorithm 3. Recall that recovered columns and rows by Algorithm 2 should share the same value at the intersection. To guarantee this, we carefully choose the recovery order of the rows and columns, and the initial values of vector recovery, to avoid assignment conflicts at the intersection. For simplicity, we only introduce Algorithm 3 for a phase function defined on $\mathbb{R}\times\mathbb{R}$. We will leave the extension to high-dimensional cases as future work.

In the case of higher dimensions, the discretization of the oscillatory integral transform and the arrangement of grid points will lead to fake discontinuity along the column and row indices. For example, a column or a row as a one-dimensional function in the index is discontinuous at a certain point, while we look back to the original high dimensional

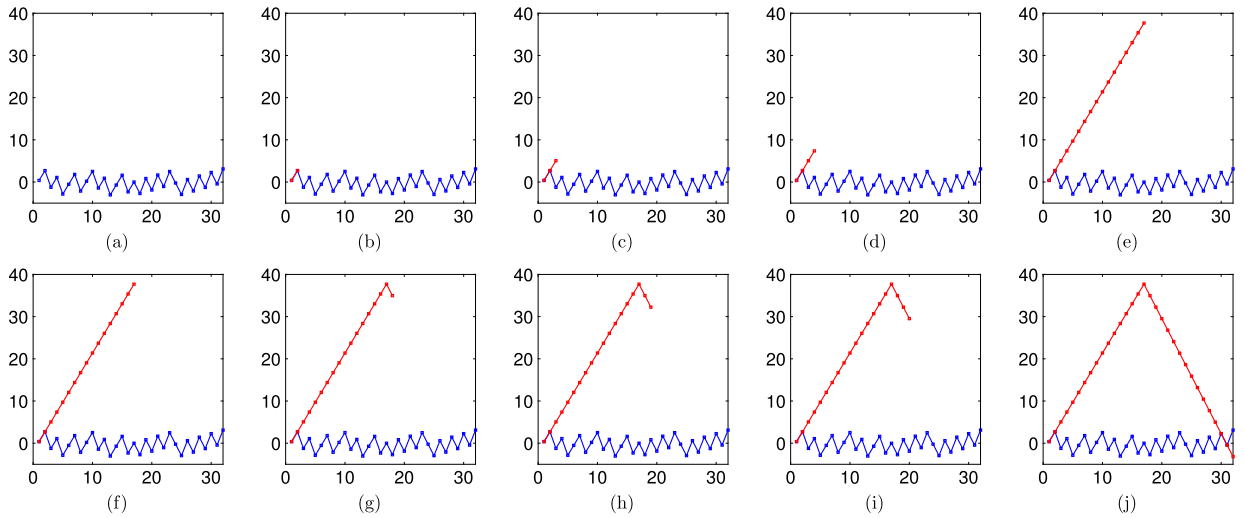

Fig. 2. Illustration of the recovery of one row of the phase function $\Phi(x,\xi) = x\cdot\xi + c(x)|\xi|$, where $c(x) = (2+\sin(2\pi x))/2$, by Algorithm 2. This row is a function in ξ denoted as v of length N, and v has two discontinuous point: one at the beginning and one in the middle. Suppose $u = \mod(v,1)$, we only know u (in blue) and would like to recover v (in red) from u. Top panel: (a) u. (b) Line 6-9 in Algorithm 2 assign the first two entries of v right after the first discontinuous point such that they have the minimum distance while maintaining $\mod(v,1) = \mod(u,1)$. (c) Line 10-13 in Algorithm 2 assign the third entry of v such that $v(2)-v(1)$ and $v(3)-v(2)$ have the minimum distance while maintaining $\mod(v,1) = \mod(u,1)$. (d) Line 16 in Algorithm 2 assigns the fourth entry of v such that $v(3)+v(1)-2v(2)$ and $v(4)+v(2)-2v(3)$ have the minimum distance while maintaining $\mod(v,1) = \mod(u,1)$. (e) Similarly, for all other i's before the second discontinuous point, assign the i-th entry of v by minimizing the distance between $v(i-1)+v(i-3)-2v(i-2)$ and $v(i)+v(i-2)-2v(i-1)$ while maintaining $\mod(v,1) = \mod(u,1)$. Bottom panel: the second discontinuous point is detected by Line 17 in Algorithm 2; apply the same procedure as for (a)-(e) to recover the second part of v after the second discontinuous point. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

1 Input: a vector u of length N, a discontinuity detection parameter τ.
2 Output: a vector v satisfying $\mod(v,1) = \mod(u,1)$, and a vector of indices $\mathcal{S}$ for discontinuity locations.
3 Initialize: $\mathcal{S} = [1]$; let n be the number of elements in $\mathcal{S}$; and let $c = 1$.
4 **while** $c \le n$ **do**
5 If $c < n$, let $st = \mathcal{S}(c)$ and $ed = \mathcal{S}(c+1) - 1$; otherwise, let $st = \mathcal{S}(c)$ and $ed = N$.
6 **if** $c = 1$ **then**
7 Set $v(st : st+1)$ such that these two values have the minimum distance while maintaining $\mod(u(st : st+1),1) = \mod(v(st : st+1),1)$.
8 **else**
9 Set $v(st)$ such that two values in $v(st-1 : st)$ have the minimum distance while maintaining $\mod(u(st-1 : st),1) = \mod(v(st-1 : st),1)$.
10 **if** $c = 1$ **then**
11 Set $v(st+2)$ such that $v(st+2) - v(st+1)$ and $v(st+1) - v(st)$ have the minimum distance while maintaining $\mod(u(st+2),1) = \mod(v(st+2),1)$.
12 **else**
13 Set $v(st+1)$ such that $v(st+1) - v(st)$ and $v(st) - v(st-1)$ have the minimum distance while maintaining $\mod(u(st+1),1) = \mod(v(st+1),1)$.
14 If $c = 1$, let $bg = st+3$; otherwise, let $bg = st+2$.
15 **for** *all indices a from bg to ed* **do**
16 Assign the value of $v(a)$ such that $v(a-1)+v(a-3)-2v(a-2)$ and $v(a)+v(a-2)-2v(a-1)$ have the minimum distance while maintaining $\mod(v(a),1) = \mod(u(a),1)$.
17 **if** $|v(a)+v(a-2)-2v(a-1)| > \tau$ **then**
18 Consider a as a new location at which v is discontinuous, add a to $\mathcal{S}$, and let $n \leftarrow n+1$.
19 Break the for-loop.
20 $c \leftarrow c+1$.

Algorithm 2: An $O(N)$ algorithm for recovering a vector v from the observation $u = \mod(v,1)$. The discontinuous locations of v is automatically detected. See Fig. 2 for an illustration with a simple example.

domain, the original kernel function is continuous at the corresponding point. Hence, once the discretization and arrangement of grid points have been fixed, we can remove the fake discontinuity and apply the same ideas as in Algorithm 3 to recover high dimensional phase functions.

With Algorithm 3 ready, we are able to introduce the nearly linear scaling algorithm for constructing a low-rank factorization UV^*, where $U \in \mathbb{C}^{N\times r}$ and $V \in \mathbb{C}^{N\times r}$, such that $e^{2\pi \mathrm{i} UV^*} = e^{2\pi \mathrm{i}\Phi}$ when we only know the kernel matrix $K = e^{2\pi \mathrm{i}\Phi}$ through Scenarios 1 and 2 in Table 3. This method is summarized in Algorithm 4.

1 Input: a vector $\mathcal{C}$ and a vector $\mathcal{R}$ as the column and row index sets indicating $O(1)$ randomly selected columns and rows of Φ, columns $U = \text{mod}(\Phi(:,\mathcal{C}),1)$, rows $V = \text{mod}(\Phi(\mathcal{R},:),1)$, a discontinuity detection parameter τ.
2 Output: columns $\tilde{U}$ and rows $\tilde{V}$ satisfying $\text{mod}(\tilde{U},1) = \text{mod}(U,1)$, and $\text{mod}(\tilde{V},1) = \text{mod}(V,1)$.
3 Apply Algorithm 2 to $U(:,\mathcal{C}(1))$ to detect a discontinuous point set $\mathcal{S}_r$; add $\mathcal{S}_r$ to $\mathcal{R}$ and update row samples V accordingly.
4 Apply Algorithm 2 to $V(\mathcal{R}(1),:)$ to detect a discontinuous point set $\mathcal{S}_c$; add $\mathcal{S}_c$ to $\mathcal{C}$ and update row samples U accordingly.
5 Let n_r be the number of elements in $\mathcal{S}_r$ and n_c be the number of elements in $\mathcal{S}_c$. The discontinuous point sets naturally partition the phase matrix into $n_r \times n_c$ blocks (see Fig. 3 for an example).
6 **for** *Each block partitioned by discontinuous point sets* **do**
7 Set $\tau = 2\pi$, since it is not necessary to detect discontinuity here.
8 Apply Algorithm 2 to recover the first row and the first column of each block.
9 Apply Algorithm 2 to recover the second and the third columns of each block. Make sure that the recovery shares the same entries when they intersect with the first row, and there is no discontinuity along rows inside the first three columns.
10 Apply Algorithm 2 to recover $O(1)$ rows of each block such that the first three entries of these rows have the same entries as in the first three columns.
11 Apply Algorithm 2 to recover $O(1)$ columns of each block such that these columns have the same entries as in the recovered rows when a column and a row intersect.

Algorithm 3: An $O(N)$ algorithm for the approximate solution of the TV^3-norm minimization when the phase function $\Phi(x,\xi)$ is defined on $\mathbb{R}\times\mathbb{R}$.

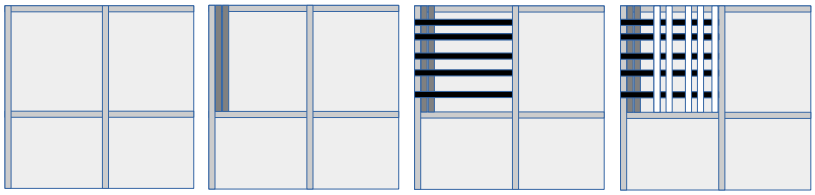

Fig. 3. Illustration of the low-rank matrix recovery in Algorithm 3. (a) The matrix is partitioned into submatrices such that there is no discontinuity along columns and rows in each submatrix. Line 8 in Algorithm 3 recovers the first column and row of each submatrix. (b) Next, Line 9 in Algorithm 3 recovers the second and the third columns for each submatrix. (c) Next, Line 10 in Algorithm 3 recovers $O(1)$ rows of each submatrix such that the first three entries of these rows have the same entries as in the first three columns. (d) Finally, Line 11 in Algorithm 3 recovers $O(1)$ columns of each submatrix such that these columns have the same entries as in the recovered rows when a column and a row intersects.

2.3. Summary for the low-rank matrix factorization

Before moving to the algorithms for other main steps of the unified framework as shown in Fig. 1, let us summarize how those algorithms in Section 2.1 and Section 2.2 can be applied to construct the low-rank matrix factorization of the amplitude and phase functions with nearly linear computational complexity.

For a general kernel $K(x,\xi) = \alpha(x,\xi)e^{2\pi i\Phi(x,\xi)}$, suppose we discretize $\alpha(x,\xi)$ and $\Phi(x,\xi)$ with N grid points in each variable to obtain the amplitude matrix $\mathcal{A}$ and the phase matrix Φ. When the explicit formulas of $\alpha(x,\xi)$ and $\Phi(x,\xi)$ are known, it takes $O(N)$ operations to evaluate one column or one row of $\mathcal{A}$ and Φ. Hence, the randomized SVD in Section 2.1 is able to construct the low-rank matrix factorization of $\mathcal{A}$ and Φ in $O(N)$ operations.

When the explicit formulas are unknown but they are solutions of certain PDE's as in Scenario 3 in Table 3. In this paper, we simply assume that $O(1)$ columns and rows of the amplitude and phase functions are available and the randomized SVD in Section 2.1 can be applied to construct the low-rank factorization in $O(N)$ operations. In practical applications like solving wave equations [12], this assumption for the phase function is reasonable since it can be obtained via interpolating the solution of the PDE's on a coarse grid of size independent of N. However, obtaining the amplitude function might take expensive computation for solving PDE's on a grid depending on N. Optimizing this complexity will be left as interesting future work.

In the case of indirect access in Scenario 1 and 2 in Table 3, it takes $O(N)$ or $O(N\log N)$ operations to evaluate one column or one row of the kernel matrix K. By taking the absolute value of K, we obtain one column or one row of $\mathcal{A}$. Hence, the low-rank factorization of $\mathcal{A}$ can be constructed via the randomized SVD in Section 2.1 in $O(N\log N)$ operations. Dividing the amplitude from the kernel, we have the access of the phase in the form of $e^{2\pi i\Phi(x,\xi)}$. Hence, the low-rank factorization of Φ can be constructed by Algorithm 4 in Section 2.2 in $O(N\log N)$ operations.

3. NUFFT and dimension lifting

This section introduces a new NUFFT approach by dimension lifting to evaluate the oscillatory integral transform

$$g(x) = \int \alpha(x,\xi)e^{2\pi i\Phi(x,\xi)} f(\xi)d\xi. \tag{16}$$

1 Input: Scenario 1: an algorithm for evaluating an arbitrary entry of the kernel matrix K in $O(1)$ operations; Scenario 2: an $O(N\log N)$ algorithm for applying K and its transpose to a vector. A rank parameter r, an over-sampling parameter q, and the matrix size N.
2 Output: $U\in\mathbb{C}^{N\times r}$ and $V\in\mathbb{C}^{N\times r}$ such that $e^{2\pi \mathrm{i}UV^*}=e^{2\pi \mathrm{i}\Phi}$.
3 **if** *Scenario* 1 **then**
4 | Evaluate rq randomly selected columns and rows of K.
5 **else if** *Scenario* 2 **then**
6 | Apply the kernel matrix K and its transpose to rq randomly selected natural basis vectors in $\mathbb{R}^N$ to obtain the columns and rows of K.
7 Apply Algorithm 3 with the columns and rows of K to obtain rq columns and rows of a matrix Ψ such that $e^{2\pi \mathrm{i}\Psi}=e^{2\pi \mathrm{i}\Phi}$.
8 Apply the randomized SVD with the columns and rows of Ψ to obtain the low-rank factorization of $\Psi\approx UV^*$ such that $e^{2\pi \mathrm{i}UV^*}=e^{2\pi \mathrm{i}\Phi}$, $U\in\mathbb{C}^{N\times r}$, and $V\in\mathbb{C}^{N\times r}$.

Algorithm 4: Low-rank matrix factorization for indirect access. The computational complexity in Scenario 1 is $O(N)$ and that in Scenario 2 is $O(N\log N)$.

If we could find $\{p_j(x)\}_{1\le j\le r}$ and $\{q_j(\xi)\}_{1\le j\le r}$ such that $e^{2\pi \mathrm{i}(\Phi(x,\xi)-\sum_{j=1}^r p_j(x)q_j(\xi))}$ is numerically low-rank, then (16) is reduced to $O(1)$ r-dimensional NUFFT's:

$$g(x)\approx\sum_{k=1}^{r_\epsilon}a_k(x)\int e^{2\pi \mathrm{i}\sum_{j=1}^r p_j(x)q_j(\xi)}\,(b_k(\xi)f(\xi))\,d\xi, \tag{17}$$

where $a_k(x)$ and $b_k(\xi)$ are the low-rank approximation of

$$\alpha(x,\xi)e^{2\pi \mathrm{i}(\Phi(x,\xi)-\sum_{j=1}^r p_j(x)q_j(\xi))}\approx\sum_{k=1}^{r_\epsilon}a_k(x)b_k(\xi).$$

Note that the prefactor of an r-dimensional NUFFT increases as r increases. Hence, the key condition for deciding whether NUFFT is suitable for evaluating (16) is the existence of $\{p_j(x)\}_{1\le j\le r}$ and $\{q_j(\xi)\}_{1\le j\le r}$ to ensure a small r and r_ϵ.

The choice of $\{p_j(x)\}_{1\le j\le r}$ and $\{q_j(\xi)\}_{1\le j\le r}$ is related to but different from classical low-rank approximation problems that can be solved by the SVD. In fact, we have a new low-rank approximation problem for fixed rank parameters r and r_ϵ as follows:

$$\min_{P,Q\in\mathbb{R}^{N\times r},U,V\in\mathbb{R}^{N\times r_\epsilon}}\|\mathcal{A}e^{2\pi \mathrm{i}(\Phi-PQ^*)}-UV^*\|_2, \tag{18}$$

where $\mathcal{A}$ represents the amplitude matrix for $\alpha(x,\xi)$, and Φ is the phase matrix for $\Phi(x,\xi)$. An immediate idea is to set reasonable r and r_ϵ, and solve the minimization problem in (18). If the minimum value of the objective function is sufficiently small, then we can use the NUFFT to evaluate (16) via (17). However, solving the optimization problem in (18) could be much more expensive than $O(N)$. This motivates Algorithm 5 below for deciding whether we could use NUFFT in $O(N)$ operations.

1 Input: the low-rank factorization of the phase matrix $\Phi\approx U_1V_1^*$, where $U_1\in\mathbb{C}^{N\times r_1}$ and $V_1\in\mathbb{C}^{N\times r_1}$; the low-rank factorization of the amplitude matrix $\mathcal{A}\approx U_2V_2^*$, where $U_2\in\mathbb{C}^{N\times r_2}$ and $V_2\in\mathbb{C}^{N\times r_2}$; rank parameters $r<r_1$ and r_ϵ, an over-sampling parameter $q>1$, an accuracy parameter $\epsilon\approx 0$, and the matrix size N.

2 Output: $y\in\{0,1\}$; if $y=1$, return $P,Q\in\mathbb{R}^{N\times r},U,V\in\mathbb{R}^{N\times r_\epsilon}$ satisfying the low-rank factorization

$$(U_2V_2^*)\odot e^{2\pi \mathrm{i}(U_1V_1^*-PQ^*)}\approx UV^*,$$

where $\odot$ means the entry-wise dot product of two matrices.
3 Compute the approximate r-leading SVD of the rank-r_1 matrix $U_1V_1^*$ using the randomized truncated SVD algorithm in [16,18][a] and denote it as $P\Sigma Q^*$. Update $P\Sigma\to P$.
4 Let M be a matrix consisting of rq randomly selected columns of $(U_2V_2^*)\odot e^{2\pi \mathrm{i}(U_1V_1^*-PQ^*)}$. Perform a pivoted QR decomposition of M and let R be the resulting $rq\times rq$ upper triangular matrix.
5 Let n be the number of diagonal entries of R that are larger than $R(1,1)\epsilon$. If $n<r$, let $y=1$; otherwise, let $y=0$.
6 **if** $y=1$ **then**
7 | Apply the randomized SVD to compute the low-rank factorization UV^* of $(U_2V_2^*)\odot e^{2\pi \mathrm{i}(U_1V_1^*-PQ^*)}$ with the rank parameter r_ϵ and the over-sampling parameter q.

Algorithm 5: An $O(N)$ algorithm for deciding whether NUFFT is applicable; if NUFFT is applicable, returns the low-rank factorization for the evaluation in (17).

[a] In the computation of the leading singular pair, since we have the rank-r_1 factorization, the computational cost is $O(N)$, the convergence to the ground true singular pair is very fast if a test matrix with a number of columns larger than r_1 is applied [16], and the probability to obtain high accuracy is very close to 1.

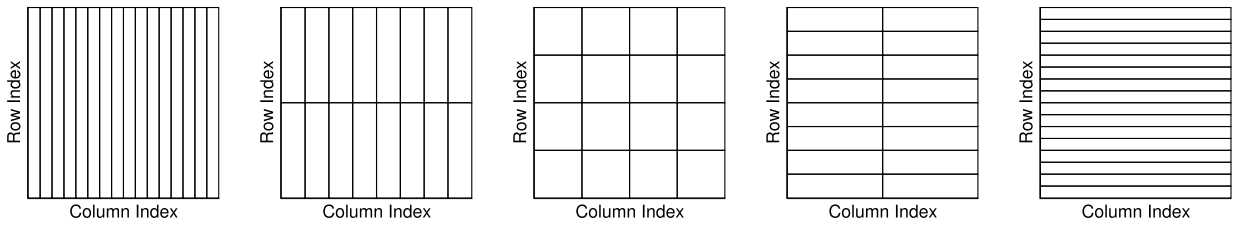

Fig. 4. Hierarchical decomposition of the row and column indices of a one-dimensional complementary low-rank matrix of size 16×16. The trees T_X (T_Ω) has a root containing 16 column (row) indices and leaves containing a single column (row) index. The rectangles above indicate some of the low-rank submatrices.

The stability and probability analysis of the main components of this algorithm can be found in [16,18,27]. If the output of Algorithm 5 is $y = 1$, then the low-rank factorization of $\mathcal{A}e^{2\pi \mathrm{i}(\Phi - PQ^*)}$ is incorporated into (17) to evaluate (16) with r_ϵ r-dimensional NUFFT's. Note that r is a parameter less than or equal to 3 according to the current development of NUFFT, and r_ϵ usually can be as large as $O(100)$ since N is usually very large. If the output of Algorithm 5 is $y = 0$, then the NUFFT approach is not applicable and we use the IBF-MAT introduced below to evaluate (16).

4. IBF-MAT

This section introduces the IBF-MAT for evaluating the oscillatory integral transform if NUFFT is not applicable. Recall that after computing the low-rank factorization of the amplitude function, our target is to evaluate (3). If NUFFT is not applicable, we compute the IBF-MAT of $e^{2\pi \mathrm{i}\Phi(x,\xi)}$, where the phase function is given in a form of low-rank matrix factorization. Then the evaluation of (3) is reduced to the application of IBF-MAT to $O(1)$ vectors. Hence, we only focus on the IBF-MAT of $e^{2\pi \mathrm{i}UV^*}$, where U and $V \in \mathbb{R}^{N\times r}$. To simplify the discussion, we also assume that x and ξ are one-dimensional variables. It is easy to extend the IBF-MAT to multi-dimensional cases following the ideas in [9,22,24,25].

$K := e^{2\pi \mathrm{i}UV^*}$ is a complementary low-rank matrix that has been widely studied in [13,14,23,25,28,30,39]. Let X and Ω be the row and column index sets of $e^{2\pi \mathrm{i}UV^*}$. Two trees T_X and T_Ω of the same depth $L = O(\log N)$, associated with X and Ω respectively, are constructed by dyadic partitioning. Denote the root level of the tree as level 0 and the leaf one as level L. Such a matrix K of size $N \times N$ is said to satisfy the **complementary low-rank property** if for any level ℓ, any node A in T_X at level ℓ, and any node B in T_Ω 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. See Fig. 4 for an illustration of low-rank submatrices in a complementary low-rank matrix of size 16×16.

In a special case when K has an explicit formula, [9] proposed an $O(N \log N)$ butterfly algorithm to construct a data-sparse representation of K using the low-rank factorizations of low-rank submatrices in the complementary low-rank structure. [22] further optimized this algorithm and formulated it into the form of BF:

$$K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \ldots (H^1)^* (V^0)^*, \tag{19}$$

where the depth $L = O(\log N)$ is assumed to be even, $h = L/2$ is a middle-level index, and all factors are sparse matrices with $O(N)$ nonzero entries. Storing and applying the BF requires only $O(N \log N)$ complexity. However, in a general case when only the low-rank factorization of the phase matrix $\Phi \approx UV^*$ is available, the state-of-the-art purely algebraic approach to constructing the BF requires at least $O(N^{1.5})$ computational complexity [23]. Though the application of the BF is highly efficient, the precomputation of the factorization is still not practical when N is large.

The IBF-MAT in this paper admits $O(N \log N)$ construction and application complexity, which would be a useful tool in developing nearly linear scaling algorithms to solve a wide class of differential and integral equations when incorporated into the schemes in [17,21,26,32,33]. The main difference between IBF-MAT and the BF in [9,22] is that we apply Algorithm 1 in Section 2.1 to construct the low-rank factorization of low-rank submatrices, instead of the interpolative low-rank approximation in Section 2.1 in [22]. Hence, to reduce the length of this paper, we only illustrate how Algorithm 1 in this paper is applied to design an $O(N \log N)$ butterfly algorithm. The reader is referred to [22] for the routines that construct the data-sparse representation in the form of (19) using the new butterfly algorithm.

With no loss of generality, we assume that $K = e^{2\pi \mathrm{i}UV^*}$ coming from the discretization of $K(x,\xi) = e^{2\pi \mathrm{i}\Phi(x,\xi)}$ with a uniform grid. Given an input vector $\{f(\xi), \xi \in \Omega\}$, the goal is to compute the *potential vector* $\{g(x), x \in X\}$ defined by

$$g(x) = \sum_{\xi\in\Omega} K(x,\xi) f(\xi), \quad x \in X.$$

The main data structure of the butterfly algorithm is a pair of dyadic trees T_X and T_Ω. Recall that for any pair of intervals $A \times B \in T_X \times T_\Omega$ obeying the condition $\ell_A + \ell_B = L$, the submatrix $\{K(x,\xi)\}_{x\in A,\xi\in B}$ is approximately of a constant rank. An explicit method to construct its low-rank approximation is given by Algorithm 1. More precisely, for any $\epsilon > 0$, there exists a constant r_ϵ independent of N and two sets of functions $\{\alpha_t^{AB}(x)\}_{1\le t\le r_\epsilon}$ and $\{\beta_t^{AB}(\xi)\}_{1\le t\le r_\epsilon}$ given in (11) or (12) such that

$$\left| K(x,\xi) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\beta_t^{AB}(\xi) \right| \le \epsilon, \quad \forall x \in A, \forall \xi \in B. \tag{20}$$

For a given interval B in Ω, define $u^B(x)$ to be the *restricted potential* over the sources $\xi \in B$

$$u^B(x) = \sum_{\xi \in B} K(x,\xi) g(\xi).$$

The low-rank property gives a compact expansion for $\{u^B(x)\}_{x\in A}$. Summing (20) over $\xi \in B$ with coefficients $g(\xi)$ gives

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x) \left(\sum_{\xi\in B} \beta_t^{AB}(\xi) g(\xi) \right) \right| \le \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

Therefore, if one can find coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ obeying

$$\lambda_t^{AB} \approx \sum_{\xi\in B} \beta_t^{AB}(\xi) g(\xi), \quad 1 \le t \le r_\epsilon, \tag{21}$$

then the restricted potential $\{u^B(x)\}_{x\in A}$ admits a compact expansion

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x) \lambda_t^{AB} \right| \le \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

The butterfly algorithm below provides an efficient way for computing $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ recursively. The general structure of the algorithm consists of a top-down traversal of T_X and a bottom-up traversal of T_Ω, carried out simultaneously. A schematic illustration of the data flow in this algorithm is provided in Fig. 5.

Algorithm 4.1. Butterfly algorithm

1. *Preliminaries.* Construct the trees T_X and T_Ω.
2. *Initialization.* Let A be the root of T_X. For each leaf interval B of T_Ω, construct the expansion coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ for the potential $\{u^B(x)\}_{x\in A}$ by simply setting

$$\lambda_t^{AB} = \sum_{\xi\in B} \beta_t^{AB}(\xi) g(\xi), \quad 1 \le t \le r_\epsilon. \tag{22}$$

   By the interpolative low-rank approximation in Algorithm 1 applied to $e^{2\pi i \Phi(A,B)}$ in the variable ξ in B, we can define the expansion coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ by

$$\lambda_t^{AB} := e^{-2\pi i \Phi(c_A, \xi_t^B)} \sum_{\xi\in B} \left(M_t^B(\xi) e^{2\pi i \Phi(c_A,\xi)} g(\xi) \right), \tag{23}$$

   where $\{\xi_t^B\}_{1\le t\le r_\epsilon}$ is the set of grid points adapted to B by (8).
3. *Recursion.* For $\ell = 1, 2, \ldots, L/2$, visit level ℓ in T_X and level $L-\ell$ in T_Ω. For each pair (A,B) with $\ell_A = \ell$ and $\ell_B = L-\ell$, construct the expansion coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ for the potential $\{u^B(x)\}_{x\in A}$ using the low-rank representation constructed at the previous level. Let P be A's parent and C be a child of B. Throughout, we shall use the notation $C \succ B$ when C is a child of B. At level $\ell - 1$, the expansion coefficients $\{\lambda_s^{PC}\}_{1\le s\le r_\epsilon}$ of $\{u^C(x)\}_{x\in P}$ are readily available and we have

$$\left| u^C(x) - \sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x) \lambda_s^{PC} \right| \le \left(\sum_{\xi\in C} |g(\xi)| \right) \epsilon, \quad \forall x \in P.$$

   Since $u^B(x) = \sum_{C\succ B} u^C(x)$, the previous inequality implies that

$$\left| u^B(x) - \sum_{C\succ B} \sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x) \lambda_s^{PC} \right| \le \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in P.$$

   Since $A \subset P$, the above approximation is of course true for any $x \in A$. However, since $\ell_A + \ell_B = L$, the sequence of restricted potentials $\{u^B(x)\}_{x\in A}$ also has a low-rank approximation of size r_ϵ, namely,

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\lambda_t^{AB} \right| \le \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

Combining the last two approximations, we obtain that $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ should obey

$$\sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\lambda_t^{AB} \approx \sum_{C\succ B}\sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x)\lambda_s^{PC}, \quad \forall x \in A. \tag{24}$$

This is an over-determined linear system for $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ when $\{\lambda_s^{PC}\}_{1\le s\le r_\epsilon, C\succ B}$ are available. The butterfly algorithm uses an efficient linear transformation approximately mapping $\{\lambda_s^{PC}\}_{1\le s\le r_\epsilon, C\succ B}$ into $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ as follows

$$\lambda_t^{AB} := e^{-2\pi \mathrm{i}\Phi(c_A,\xi_t^B)} \sum_{C\succ B}\sum_{s=1}^{r_\epsilon} M_t^B(\xi_s^C) e^{2\pi \mathrm{i}\Phi(c_A,\xi_s^C)} \lambda_s^{PC}, \tag{25}$$

where $\{\xi_t^B\}_{1\le t\le r_\epsilon}$ (and $\{\xi_t^C\}_{1\le t\le r_\epsilon}$) is the set of grid points adapted to B (and C) by (8).

4. *Switch.* For the levels visited, interpolation is applied in variable ξ, while the interpolation is applied in variable x for levels $\ell > L/2$. Hence, we are switching the interpolation variable in Algorithm 4 at this step. Now we are still working on level $\ell = L/2$ and the same domain pairs (A, B) in the last step. Let λ_s^{AB} denote the expansion coefficients obtained by interpolative low-rank factorization using Algorithm 1 applied to $e^{2\pi \mathrm{i}\Phi(A,B)}$ in the variable ξ in B in the last step. Correspondingly, $\{\xi_s^B\}_s$ are the interpolation grid points in B in the last step. We take advantage of the interpolation in variable x in A using Algorithm 1 applied to $e^{2\pi \mathrm{i}\Phi(A,B)}$ and generate grid points $\{x_t^A\}_{1\le t\le r_\epsilon}$ in A by (7). Then we can define new expansion coefficients

$$\lambda_t^{AB} := \sum_{s=1}^{r_\epsilon} e^{2\pi \mathrm{i}\Phi(x_t^A,\xi_s^B)} \lambda_s^{AB}.$$

5. *Recursion.* Similar to the discussion in Step 3, we go up in tree T_Ω and down in tree T_X at the same time until we reach the level $\ell = L$. We construct the low-rank approximation functions by interpolation in variable x using Algorithm 1 as follows:

$$\alpha_t^{AB}(x) = e^{2\pi \mathrm{i}\Phi(x,c_B)} M_t^A(x) e^{-2\pi \mathrm{i}\Phi(x_t^A,c_B)},\ \beta_t^{AB}(\xi) = e^{2\pi \mathrm{i}\Phi(x_t^A,\xi)}, \tag{26}$$

where $\{x_t^A\}_{1\le t\le r_\epsilon}$ is the set of grid points adapted to A by (7).
Hence, the new expansion coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ can be defined as

$$\lambda_t^{AB} := \sum_{C\succ B} e^{2\pi \mathrm{i}\Phi(x_t^A,c_C)} \sum_{s=1}^{r_\epsilon} \left(M_s^P(x_t^A) e^{-2\pi \mathrm{i}\Phi(x_s^P,c_C)} \lambda_s^{PC} \right), \tag{27}$$

where again P is A's parent and C is a child interval of B.

6. *Termination.* Finally, $\ell = L$ and set B to be the root node of T_Ω. For each leaf interval $A \in T_X$, use the constructed expansion coefficients $\{\lambda_t^{AB}\}_{1\le t\le r_\epsilon}$ in (27) to evaluate $u^B(x)$ for each $x \in A$,

$$\begin{aligned} u(x) = u^B(x) &= \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\lambda_t^{AB} \\ &= e^{2\pi \mathrm{i}\Phi(x,c_B)} \sum_{t=1}^{r_\epsilon} \left(M_t^A(x) e^{-2\pi \mathrm{i}\Phi(x_t^A,c_B)} \lambda_t^{AB} \right), \end{aligned} \tag{28}$$

where $\{x_t^A\}_{1\le t\le r_\epsilon}$ is the set of grid points adapted to A by (7).

Like the butterfly algorithm in [9], Algorithm 4.1 is a direct approach that uses the low-rank matrix factorization by Algorithm 1 on-the-fly to evaluate the oscillatory integral transform

$$g(x) = \int e^{2\pi \mathrm{i}\Phi(x,\xi)} f(\xi) d\xi$$

in $O(N \log N)$ operations without precomputation. If repeated applications of the integral transform to multiple functions f's are required, it is more efficient to keep the low-rank matrix factorizations and arrange them into the form of BF in (19). Besides, the rank provided by interpolative factorization is far from optimal, which motivates the structure-preserving sweeping matrix compression technique in [22] to further compress the preliminary BF by interpolative factorization to obtain a sparser BF, which is the IBF-MAT of the kernel $e^{2\pi \mathrm{i}\Phi(x,\xi)}$ in this paper. The reader is referred to [22] for a complete re-compression algorithm.

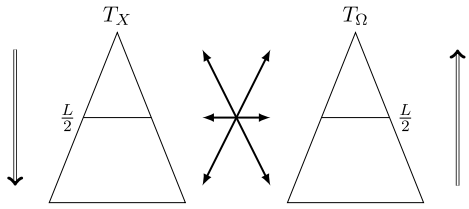

Fig. 5. Trees of the row and column indices. Left: T_X for the row indices X. Right: T_Ω for the column indices Ω. The interaction between $A \in T_X$ and $B \in T_\Omega$ starts at the root of T_X and the leaves of T_Ω.

5. Numerical results

This section presents several numerical examples to demonstrate the efficiency of the proposed unified framework. The numerical results were obtained on a computer with Intel® Xeon® CPU X5690 @ 3.47GHz (6 core/socket) and 128 GB RAM. All implementations are in MATLAB® and are available in the second version of ButterflyLab (https://github.com/ButterflyLab/ButterflyLab).

Let $\{g^d(x), x \in X\}$, $\{g^b(x), x \in X\}$, and $\{g^n(x), x \in X\}$ denote the results given by the direct matrix-vector multiplication, IBF-MAT, and NUFFT, respectively. The accuracy of applying fast algorithms is estimated by the relative error defined as follows,

$$\epsilon^b = \sqrt{\frac{\sum_{x\in S} |g^b(x) - g^d(x)|^2}{\sum_{x\in S} |g^d(x)|^2}} \quad \text{and} \quad \epsilon^n = \sqrt{\frac{\sum_{x\in S} |g^n(x) - g^d(x)|^2}{\sum_{x\in S} |g^d(x)|^2}}, \tag{29}$$

where S is an index set containing 256 randomly sampled row indices of the kernel matrix K. The error for recovering the amplitude function is defined as

$$\epsilon^{amp} = \frac{\|\mathcal{A}(S,S) - U(S,:)V(:,S)^*\|_2}{\|\mathcal{A}(S,S)\|_2}, \tag{30}$$

where $\mathcal{A}$ is the amplitude matrix and UV^* is its low-rank recovery. The error for recovering the phase and the kernel functions are defined similarly and denoted as ϵ^{pha} and ϵ^K, respectively.

5.1. Accuracy and scaling of low-rank matrix recovery and IBF-MAT

In the first part of the numerical section, we present numerical results of several examples to demonstrate the accuracy and asymptotic scaling of the proposed low-rank matrix recovery for amplitude and phase functions, and IBF-MAT. With no loss of generality, we only focus on Scenarios 1 and 2 of indirect access. For the first scenario, we apply the proposed algorithms to evaluate an FIO in 1D and a Hankel matrix transform. For the second scenario, we compute the IBF-MAT of the composition of two FIO's when we only have the BF representing each FIO.

One-dimensional FIO

Our first example is to evaluate a one-dimensional FIO [23] of the following form:

$$g(x) = \int_{\mathbb{R}} \alpha(x,\xi) e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi) d\xi, \tag{31}$$

where $\widehat{f}$ is the Fourier transform of f, $\alpha(x,\xi) = 1$, and $\Phi(x,\xi)$ is a phase function given by

$$\Phi(x,\xi) = x\cdot\xi + c(x)|\xi|, \quad c(x) = (2 + 0.2\sin(2\pi x))/16. \tag{32}$$

The discretization of (31) is

$$g(x_i) = \sum_{\xi_j} \alpha(x_i,\xi_j) e^{2\pi\imath\Phi(x_i,\xi_j)} \widehat{f}(\xi_j), \quad i,j = 1,2,\ldots,N, \tag{33}$$

where $\{x_i\}$ and $\{\xi_j\}$ are points uniformly distributed in $[0,1)$ and $[-N/2, N/2)$ following

$$x_i = (i-1)/N \text{ and } \xi_j = j - 1 - N/2. \tag{34}$$

This example is for Scenario 1 in Table 3. The unified framework is applied to recover the amplitude and phase functions in a form of low-rank matrix factorization, compute the IBF-MAT of the kernel function, and apply the IBF-MAT as in (3) to a randomly generated f in (31) to obtain g. Table 4 summarizes the results of this example for different grid sizes N and

Table 4
Numerical results for the one-dimensional FIO given in (33). T_{rec} is the time for recovering the amplitude and phase functions, T_{fac} is the time for computing the IBF-MAT, T_{app} is the time for applying the IBF-MAT, and T_d is the time for a direct summation in (33).

N, r_ϵ	ϵ^b	ϵ^K	ϵ^{pha}	ϵ^{amp}	T_{rec} (min)	T_{fac} (min)	T_{app} (sec)	T_d/T_{app}
4096, 8	3.16e-06	3.15e-11	3.15e-11	1.20e-15	4.21e-02	2.26e-01	1.62e-02	4.52e+01
4096,12	7.87e-11	2.23e-11	2.23e-11	1.31e-15	4.06e-02	2.11e-01	2.34e-02	3.53e+01
16384, 8	3.98e-06	4.77e-11	4.77e-11	1.22e-15	1.54e-01	1.13e+00	7.39e-02	1.08e+02
16384,12	1.87e-10	2.12e-10	2.12e-10	1.23e-15	1.47e-01	1.13e+00	1.14e-01	6.67e+01
65536, 8	5.35e-06	2.30e-09	2.30e-09	1.22e-15	5.77e-01	5.41e+00	3.07e-01	3.32e+02
65536,12	2.01e-09	3.47e-09	3.47e-09	1.22e-15	5.96e-01	5.92e+00	5.40e-01	2.63e+02
262144, 8	4.51e-06	7.31e-09	7.31e-09	1.14e-15	2.32e+00	2.73e+01	1.48e+00	1.12e+03
262144,12	7.70e-09	9.88e-09	9.88e-09	1.25e-15	2.33e+00	2.94e+01	2.55e+00	7.82e+02

Table 5
Numerical results for the Hankel function transform given in (35). T_{rec} is the time for recovering the amplitude and phase functions, T_{fac} is the time for computing the IBF-MAT, T_{app} is the time for applying the IBF-MAT, and T_d is the time for a direct summation in (35).

N, r_ϵ	ϵ^b	ϵ^K	ϵ^{pha}	ϵ^{amp}	T_{rec} (min)	T_{fac} (min)	T_{app} (sec)	T_d/T_{app}
4096, 8	9.03e-06	2.52e-09	1.42e-09	1.16e-10	6.68e-02	1.98e-01	8.72e-02	9.98e+01
4096,12	4.93e-07	9.66e-08	8.34e-08	3.05e-11	6.66e-02	1.64e-01	9.15e-02	1.01e+02
16384, 8	1.66e-04	7.16e-07	6.00e-07	9.17e-10	2.42e-01	1.02e+00	5.03e-01	2.04e+02
16384,12	2.43e-07	3.61e-08	2.16e-08	1.87e-10	2.49e-01	8.88e-01	5.01e-01	2.08e+02
65536, 8	7.15e-06	2.98e-06	1.24e-06	1.11e-07	9.61e-01	4.96e+00	3.57e+00	4.74e+02
65536,12	4.10e-05	1.99e-04	2.89e-06	3.12e-05	9.56e-01	4.67e+00	3.97e+00	4.69e+02
262144, 8	5.41e-06	9.77e-06	4.93e-06	1.26e-06	3.89e+00	2.42e+01	1.79e+01	1.38e+03
262144,12	5.94e-05	1.58e-04	4.27e-06	2.17e-05	3.89e+00	2.44e+01	2.08e+01	1.17e+03

numbers of interpolation points r_ϵ. In the low-rank approximations of amplitude and phase functions, the rank parameter is 20 and the over-sampling parameter is 5.

Table 4 shows that for a fixed number of interpolation points r_ϵ, and a rank parameter for the amplitude and phase functions, the accuracy of the low-rank matrix recovery and the IBF-MAT stay in almost the same order. As for the computational complexity, both the factorization time and the application time of the IBF-MAT, and the reconstruction time of the amplitude and phase functions scales like $N \log N$.

Special function transform

Next, we provide an example of a special function transform. Following the standard notation, we denote the Hankel function of the first kind of order m by $H_m^{(1)}$. When m is an integer, $H_m^{(1)}$ has a singularity at the origin and a branch cut along the negative real axis. We are interested in evaluating the sum of Hankel functions over different orders,

$$g(x_i) = \sum_{j=1}^{N} H_{j-1}^{(1)}(x_i) f_j, \quad i = 1, 2, \ldots, N, \tag{35}$$

which is analogous to expansion in orthogonal polynomials. The points x_i are defined via the formula

$$x_i = N + \frac{2\pi}{3}(i-1),$$

which are bounded away from zero. The entries of the matrix in the above matvec can be explicitly calculated on-the-fly in $O(1)$ operations per entry using asymptotic formulas. The unified framework will work for many other orthogonal transforms in the oscillatory regime that admit smooth amplitude and phase functions. For more examples see [4,3].

This example is also for Scenario 1 in Table 3. The unified framework is applied to recover the amplitude and phase functions in a form of low-rank matrix factorization, compute the IBF-MAT of the kernel function, and apply the IBF-MAT as in (3) to a randomly generated f to obtain g. Table 5 summarizes the results of this example for different grid sizes N and numbers of interpolation points r_ϵ. In the low-rank approximations of amplitude and phase functions, the rank parameter is 20 and the over-sampling parameter is 5.

The results in Table 5 agree with the $O(N \log N)$ complexity analysis and the speed-up factor over a direct summation is significant. The accuracy of the IBF-MAT becomes better if r_ϵ is larger and is almost independent of the problem size. Note that the recovery accuracy of the amplitude and phase functions becomes worse as N increases. This is due to the fact that there is a singularity point in the corner of the amplitude matrix (see Fig. 6), leading to an increasing rank of the amplitude matrix as the problem size increases. Besides, the randomized sampling algorithm in the randomized SVD is not good in the presence of singularity, unless we know this singularity a prior so that we sample more at the corner. Hence, when

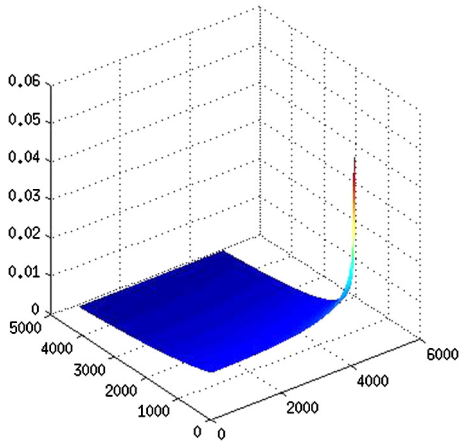

Fig. 6. The exact amplitude function of the example in (35). There is a singularity point in the corner of the amplitude matrix, leading to an increasing rank of the amplitude matrix as the problem size increases.

$N > 16384$ the accuracy of the low-rank amplitude and phase recovery is not very good and this influences the accuracy of the IBF-MAT, since the accuracy of the IBF-MAT is bounded below by the recovery accuracy. It is easy to fix this issue. After reconstructing the amplitude and phase, we can check singularity and reconstruct these functions again with adjusted sampling strategies to improve the accuracy. This works well in practice and we don't show the numerical results to save the space of the paper.

Composition of two FIO's in 1D

The third example is to evaluate a composition of two FIO's of the following form:

$$g(x) = \mathcal{L} \circ \mathcal{L}(f), \tag{36}$$

where $\mathcal{L}$ is an FIO of the form

$$g(x) = \int_{\mathbb{R}} e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi) d\xi, \tag{37}$$

where $\Phi(x,\xi)$ is a phase function given by

$$\Phi(x,\xi) = x \cdot \xi + c(x)\xi, \quad c(x) = (2 + 0.2\sin(2\pi x))/16. \tag{38}$$

The discretization of (37) is similar to (33).

This example is for Scenario 2 in Table 3. The unified framework is applied to recover the amplitude and phase functions in a form of low-rank matrix factorization, compute the IBF-MAT of the kernel function, and apply the IBF-MAT as in (3) to a randomly generated f in (36) to obtain g. Table 6 summarizes the results of this example for different grid sizes N and numbers of interpolation points r_ϵ. In the low-rank approximations of amplitude and phase functions, the rank parameter is 20 and the over-sampling parameter is 5.

We would like to emphasize that the composition of two FIO's results in an FIO with a phase function that is singular at the point $\xi = 0$. This leads to large-rank submatrices in the kernel matrix. In this case, we can adopt the multiscale butterfly algorithm/factorization in [24,25] to deal with this singularity. We have implemented the multiscale version of the IBF-MAT and present its numerical performance in Table 6. For the purpose of reducing the length of this paper, we don't introduce the multiscale IBF-MAT. The reader is referred to [24,25] for a detailed description of the multiscale idea.

Table 6 shows that for a fixed number of interpolation points r_ϵ, and a rank parameter for the amplitude and phase functions, the accuracy of the low-rank matrix recovery and the multiscale IBF-MAT stay in almost the same order, though the accuracy becomes slightly worse as the problem size increases. The slightly increasing error is due to the randomness of the proposed algorithm as explained previously. There is no explicit formula for the amplitude and phase functions in this example. Hence, we cannot estimate the accuracy of the recovery algorithm. Since the accuracy of the multiscale IBF-MAT is bounded below by the accuracy of amplitude and phase recovery. We see that the recovery accuracy should be very good.

As for the computational complexity, both the factorization time and the application time of the IBF-MAT, and the reconstruction time of the amplitude and phase functions scales like $N \log N$.

5.2. Comparison of NUFFT and BF

In the second part of the numerical section, we illustrate the $O(N)$ strategy in Algorithm 5 for deciding whether we can use NUFFT in the oscillatory integral transform. We will show that once the NUFFT is applicable, it is more efficient than

Table 6
Numerical results for the composition of two FIO's given in (36). T_{rec} is the time for recovering the amplitude and phase functions, T_{fac} is the time for computing the multiscale IBF-MAT, and T_{app} is the time for applying the multiscale IBF-MAT.

N, r_ϵ	ϵ^b	T_{rec} (min)	T_{fac} (min)	T_{app} (sec)
4096, 8	4.59e-06	2.41e-01	1.81e-01	1.79e-01
4096,12	9.24e-10	2.45e-01	1.71e-01	3.14e-01
16384, 8	5.42e-06	1.80e+00	1.02e+00	1.69e+00
16384,12	1.69e-09	1.80e+00	1.08e+00	1.86e+00
65536, 8	6.29e-06	1.01e+01	5.36e+00	1.42e+01
65536,12	9.25e-09	1.01e+01	5.72e+00	2.68e+01
262144, 8	7.16e-06	3.29e+01	2.51e+01	8.35e+01
262144,12	2.45e-08	3.25e+01	3.00e+01	1.43e+02

the BF considering that the prefactor of the factorization and application time of the BF is larger than that of the NUFFT approach, when we require an approximate matvec with high accuracy, no matter how many vectors in the matvec. To this end, we will provide an example of FIO's in solving wave equations. In the case of low accuracy requirement, according to the comparison of BF and NUFFT in Table 1 and 2 in [22], our conclusion just above still valid.

Fast algorithms for solving wave equations with variable coefficients based on FIO's have been studied based on either the BF in [12] or the wave packet representation of the FIO's in [7,11]. [12] also proposed an approach to solve wave equations based on a carefully designed NUFFT according to the explicit formulas of FIO's inspired by the work in [8].

We propose to apply the new NUFFT approach with dimension lifting for the evaluation of FIO's used in [12] in the one-dimensional case. This new method does not rely on the explicit formula of an FIO and can be applied to more general scenarios. Besides, the dimension lifting idea could lead to fewer applications of the NUFFT, since the rank r_ϵ in (17) could be smaller compared to the NUFFT approach in [12]. The application of the new NUFFT approach in higher dimensional spaces will be left as future work.

In more particular, the FIO used in [12] arises from the solution of the one-dimensional wave equation as follows:

$$\begin{cases} \partial_{tt}u(x,t) - \partial_x(c^2(x)\partial_x u(x,t)) = 0 & t>0, x\in[0,1) \\ u(x,0) = u_0(x) & x\in[0,1) \\ \partial_t u(x,0) = u_1(x) & x\in[0,1), \end{cases} \tag{39}$$

where the boundary conditions are taken to be periodic. The theory of FIO's states that for a given smooth and positive $c(x)$ there exists a time t^* that depends only on $c(x)$ such that for any $t<t^*$, the general solution of (39) is given by a summation of two FIO's [12]:

$$u(x,t) = \sum_{\xi\in\mathbb{Z}} e^{2\pi \mathrm{i}\Phi_+(x,\xi,t)}\alpha_+(x,\xi,t)\widehat{f_+}(\xi) + \sum_{\xi\in\mathbb{Z}} e^{2\pi \mathrm{i}\Phi_-(x,\xi,t)}\alpha_-(x,\xi,t)\widehat{f_-}(\xi),$$

where f_+ and f_- are two functions depending on the initial conditions.

In this example, we assume that $c(x) = 2+\sin(2\pi x)$ and follow the ideas in [12] to construct the FIO's in the solution operator of (39). Without loss of generality, we focus on the evaluation of the FIO

$$\sum_{\xi\in\mathbb{Z}} e^{2\pi \mathrm{i}\Phi_+(x,\xi,t)}\widehat{f_+}(\xi). \tag{40}$$

The phase function $\Phi_+(x,\xi,t)$ satisfies the Hamilton-Jacobi equation

$$\begin{cases} \partial_t\Phi_+(x,\xi,t) - c(x)|\partial_x\Phi_+(x,\xi,t)| = 0 \\ \Phi_+(x,\xi,0) = x\cdot\xi. \end{cases} \tag{41}$$

Note that $\Phi_+(x,\xi,t)$ is homogeneous of degree 1 in ξ, i.e., $\Phi_+(x,\lambda\xi,t) = \lambda\Phi_+(x,\xi,t)$ for $\lambda>0$. Therefore, we only need to evaluate $\Phi_+(x,\xi,t)$ at $\xi=\pm1$. From the algebraic point of view, the phase matrix is piecewise rank-1, i.e.,

$$\Phi_+(x,\xi,t) = \begin{cases} \Phi_+(x,1,t)\xi, & \forall\xi\geq 0, \\ -\Phi_+(x,-1,t)\xi, & \forall\xi<0. \end{cases} \tag{42}$$

In fact, to make the boundary condition periodic in x, $\Psi_+(x,\xi,t) := \Phi_+(x,\xi,t) - x\xi$ is introduced for $\xi=\pm1$. Then we have

$$\begin{cases} \partial_t\Psi_+(x,\xi,t) - c(x)|\partial_x\Psi_+(x,\xi,t) + \xi| = 0, \\ \Psi_+(x,\xi,0) = 0. \end{cases} \tag{43}$$

Table 7
Numerical results for the evaluation of (44) for different problem sizes N at different time t. T_{FFT} is the runtime of a FFT on a vector of length N as comparison. T^b_{fac}, T^b_{app}, T^n_{fac}, and T^n_{app} are the factorization time and the application time for the IBF-MAT and the one-dimensional NUFFT, respectively. ϵ^b and ϵ^n are the relative evaluation error by the IBF-MAT and the NUFFT approaches, respectively.

N	t	$T_{FFT}(sec)$	$T^b_{fac}(sec)$	$T^b_{app}(sec)$	ϵ^b	$T^n_{fac}(sec)$	$T^n_{app}(sec)$	ϵ^n
4096	2.441e-04	2.07e-04	1.08e+01	1.14e-02	1.33e-12	9.26e-04	3.44e-03	4.22e-13
4096	1.562e-02	2.07e-04	1.03e+01	1.14e-02	1.52e-12	7.78e-04	3.36e-03	1.93e-13
16384	2.441e-04	3.21e-04	5.56e+01	5.63e-02	6.46e-12	8.95e-04	1.26e-02	1.02e-12
16384	1.562e-02	3.21e-04	5.61e+01	5.65e-02	5.29e-12	1.02e-03	1.48e-02	1.06e-12
65536	2.441e-04	2.91e-03	2.92e+02	2.70e-01	3.06e-12	9.89e-04	5.33e-02	7.89e-12
65536	1.562e-02	2.91e-03	2.93e+02	3.14e-01	3.78e-12	1.12e-03	6.05e-02	4.30e-12
262144	2.441e-04	5.41e-03	1.46e+03	1.23e+00	3.61e-12	8.45e-04	2.04e-01	6.33e-12
262144	1.562e-02	5.41e-03	1.56e+03	1.31e+00	1.04e-11	1.20e-03	2.00e-01	4.04e-11

When $c(x)$ is a band-limited function, $\Psi_+(x,\xi,t)$ is a smooth function in x when t is sufficiently smaller than t^*. Hence, a small grid in x is enough to discretize (43). The value of Ψ_+ on a finer grid in x can evaluated by spectral interpolation using FFT.

In the numerical examples here, we adopt a uniform grid with 512 grid points for x in $[0,1)$, and a time step size $\frac{1}{4096}$ to solve (43). The standard local Lax-Friedrichs Hamiltonian method is applied for x and the third order TVD Runge-Kutta method is used for t to solve (43). We vary the problem size N of the evaluation in (40) and discretize $\Phi_+(x,\xi,t)$ with a uniform spacial grid with a step size $\frac{1}{N}$ for $x\in[0,1)$ and a uniform frequency grid with a step size 1 for $\xi\in[-\frac{N}{2},\frac{N}{2})$.

By (42), we solve (43) and obtain a low-rank factorization of the phase matrix and apply IBF-MAT to evaluate (40). Note that the phase matrix is piecewise rank-1, we can split the summation in (40) into two parts:

$$\sum_{\xi\in\{-\frac{N}{2},\ldots,-1\}} e^{2\pi \mathrm{i}\Phi_+(x,\xi,t)}\widehat{f_+}(\xi) + \sum_{\xi\in\{0,1,\cdots,\frac{N}{2}-1\}} e^{2\pi \mathrm{i}\Phi_+(x,\xi,t)}\widehat{f_+}(\xi), \tag{44}$$

and apply the one-dimensional NUFFT approach to evaluate the two summations in (44). Or we can also apply the two-dimensional NUFFT approach to compute the summation in (40). The numerical results are summarized in Table 7 and Table 8. To make the accuracy of the BF and the NUFFT approaches comparable, we choose the rank parameter r_ϵ in the IBF-MAT as 12, the accuracy tolerance ϵ in the IBF-MAT and the NUFFT as $1e-12$.

Numerical results in Table 7 show that both the IBF-MAT and the one-dimensional NUFFT approach without dimension lifting admit $O(N\log N)$ factorization and application time. For almost the same evaluation accuracy, the one-dimensional NUFFT approach has a much smaller prefactor (about $O(1000)$ times smaller considering the total cost) making it more preferable.

Numerical results in Table 8 show that the two-dimensional NUFFT approach with dimension lifting also admits $O(N\log N)$ factorization and application time. Though the BF might be a few times more efficient in some cases in terms of the application time, the NUFFT approach is still more preferable considering the expensive factorization time of the BF. Although the two-dimensional NUFFT approach is more expensive than the one-dimensional NUFFT method, the two-dimensional NUFFT approach doesn't rely on the piecewise rank-1 property of the phase function and therefore is applicable in more general situations.

Although we know that the NUFFT approach is applicable to (40) and (44), we still apply Algorithm 5 to test its time scaling. The results of T_{DEC} in Table 8 also verify that Algorithm 5 for deciding whether we can apply the NUFFT approach has a linear scaling.

6. Conclusion and discussion

This paper introduced a unified framework for $O(N\log N)$ evaluation of the oscillatory integral transform $g(x)=\int\alpha(x,\xi)e^{2\pi \mathrm{i}\Phi(x,\xi)}f(\xi)d\xi$. This framework works for two cases: 1) explicit formulas for the amplitude and phase functions are known; 2) only indirect access of the amplitude and phase functions are available. In the case of indirect access, this paper proposed a novel fast algorithms for recovering the amplitude and phase functions in $O(N\log N)$ operations. Second, a new algorithm for the oscillatory integral transform based on the NUFFT and a dimension lifting technique is proposed. Finally, a new BF, the IBF-MAT, for amplitude and phase matrices in a form of a low-rank factorization is proposed. These two algorithms both require only $O(N\log N)$ operations to evaluate the oscillatory integral transform.

This unified framework would be very useful in developing efficient tools for fast special function transforms, solving wave equations, and solving electromagnetic (EM) scattering problems. We have provided several examples to support these applications. For example, the state-of-the-art fast algorithm for computing the compositions of FIO's, which could be applied as a preconditioner for certain classes of parabolic and hyperbolic equations [21,32,33]; a fast algorithm for solving wave equation via FIO's. We have explored the potential applications of the proposed framework to: 1) fast evaluation of

Table 8
Numerical results for the evaluation of (40) for different problem sizes N at different time t. T_{dec} is the runtime of Algorithm 5. T^b_{fac}, T^b_{app}, T^n_{fac}, and T^n_{app} are the factorization time and the application time for the IBF-MAT and the two-dimensional NUFFT approach by dimension lifting, respectively. ϵ^b and ϵ^n are the relative evaluation error by the IBF-MAT and the NUFFT approaches, respectively.

N	t	$T_{dec}(sec)$	$T^b_{fac}(sec)$	$T^b_{app}(sec)$	ϵ^b	$T^n_{fac}(sec)$	$T^n_{app}(sec)$	ϵ^n
4096	2.441e-04	4.09e-02	1.08e+01	1.14e-02	1.33e-12	7.83e-04	1.96e-03	4.66e-13
4096	1.562e-02	3.87e-02	1.03e+01	1.14e-02	1.52e-12	1.66e-03	2.21e-02	8.89e-13
16384	2.441e-04	1.40e-01	5.56e+01	5.63e-02	6.46e-12	6.30e-04	7.41e-03	1.07e-12
16384	1.562e-02	1.47e-01	5.61e+01	5.65e-02	5.29e-12	4.16e-04	1.54e-01	1.62e-12
65536	2.441e-04	6.60e-01	2.92e+02	2.70e-01	3.06e-12	7.04e-04	3.25e-02	4.69e-12
65536	1.562e-02	6.76e-01	2.93e+02	3.14e-01	3.78e-12	5.01e-04	9.34e-01	3.92e-11
262144	2.441e-04	2.94e+00	1.46e+03	1.23e+00	3.61e-12	2.34e-03	1.18e-01	2.34e-11
262144	1.562e-02	3.15e+00	1.56e+03	1.31e+00	1.04e-11	4.51e-04	8.13e+00	3.02e-10

other special functions [5,6] to develop nearly linear scaling polynomial transforms; 2) fast solvers developed in [17,26] for nearly linear algorithms for solving high-frequency EM equations. Numerical results will be reported in forthcoming papers.

Acknowledgements

The author thanks the fruitful discussion with Lexing Ying, the support of the start-up grant R-146-000-251-133 by the Department of Mathematics at the National University of Singapore, and the support of the Ministry of Education in Singapore under the grant MOE2018-T2-2-147.

References

[1] G. Bao, W.W. Symes, Computation of pseudo-differential operators, SIAM J. Sci. Comput. 17 (2) (1996) 416–429.
[2] J.P. Boyd, F. Xu, Divergence (Runge Phenomenon) for least-squares polynomial approximation on an equispaced grid and Mock Chebyshev subset interpolation, Appl. Comput. Math. 210 (1) (2009) 158–168.
[3] J. Bremer, An algorithm for the numerical evaluation of the associated Legendre functions that runs in time independent of degree and order, J. Comput. Phys. 360 (2018) 15–38.
[4] J. Bremer, An algorithm for the rapid numerical evaluation of Bessel functions of real orders and arguments, Adv. Comput. Math. 45 (1) (Feb. 2019) 173–211.
[5] J. Bremer, Q. Pang, H. Yang, Fast algorithms for the multi-dimensional Jacobi polynomial transform, arXiv:1901.07275 [math.NA], 2019.
[6] J. Bremer, H. Yang, Fast algorithms for Jacobi expansions via nonoscillatory phase functions, IMA J. Numer. Anal. (2019), in press.
[7] P. Caday, Computing Fourier integral operators with caustics, Inverse Probl. 32 (12) (2016) 125001.
[8] E. Candès, L. Demanet, L. Ying, Fast computation of Fourier integral operators, SIAM J. Sci. Comput. 29 (6) (2007) 2464–2493.
[9] E.J. Candès, L. Demanet, L. Ying, A fast butterfly algorithm for the computation of Fourier integral operators, Multiscale Model. Simul. 7 (4) (2009) 1727–1750.
[10] M. Costantin, A. Farina, F. Zirilli, A fast phase unwrapping algorithm for SAR interferometry, IEEE Trans. Geosci. Remote Sens. 37 (1) (1999).
[11] M.V. de Hoop, G. Uhlmann, A. Vasy, H. Wendt, Multiscale discrete approximations of Fourier integral operators associated with canonical transformations and caustics, Multiscale Model. Simul. 11 (2) (2013) 566–585.
[12] L. Demanet, L. Ying, Fast wave computation via Fourier integral operators, Math. Comput. 81 (279) (2012).
[13] B. Engquist, L. Ying, Fast directional multilevel algorithms for oscillatory kernels, SIAM J. Sci. Comput. 29 (4) (2007) 1710–1737.
[14] B. Engquist, L. Ying, A fast directional algorithm for high frequency acoustic scattering in two dimensions, Commun. Math. Sci. 7 (2) (06 2009) 327–345.
[15] L. Greengard, J.-Y. Lee, Accelerating the nonuniform fast Fourier transform, SIAM Rev. 46 (3) (2004) 443–454.
[16] M. Gu, Subspace iteration randomization and singular value problems, SIAM J. Sci. Comput. 37 (3) (2015) A1139–A1173.
[17] H. Guo, Y. Liu, J. Hu, E. Michielssen, A butterfly-based direct integral-equation solver using hierarchical Lu factorization for analyzing scattering from electrically large conducting objects, IEEE Trans. Antennas Propag. 65 (9) (Sep. 2017) 4742–4750.
[18] N. Halko, P.-G. Martinsson, J.A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.
[19] P. Hoffman, K. Reddy, Numerical differentiation by high order interpolation, SIAM J. Sci. Stat. Comput. 8 (6) (1987) 979–987.
[20] J. Hu, S. Fomel, L. Demanet, L. Ying, A fast butterfly algorithm for generalized Radon transforms, Geophysics 78 (4) (June 2013) U41–U51.
[21] H. Isozaki, J.L. Rousseau, Pseudodifferential multi-product representation of the solution operator of a parabolic equation, Commun. Partial Differ. Equ. 34 (7) (2009) 625–655.
[22] Y. Li, H. Yang, Interpolative butterfly factorization, SIAM J. Sci. Comput. 39 (2) (2017) A503–A531.
[23] Y. Li, H. Yang, E.R. Martin, K.L. Ho, L. Ying, Butterfly factorization, Multiscale Model. Simul. 13 (2) (2015) 714–732.
[24] Y. Li, H. Yang, L. Ying, A multiscale butterfly algorithm for Fourier integral operators, Multiscale Model. Simul. 13 (2) (2015) 614–631.
[25] Y. Li, H. Yang, L. Ying, Multidimensional butterfly factorization, Appl. Comput. Harmon. Anal. (2017).
[26] Y. Liu, H. Guo, E. Michielssen, An HSS matrix-inspired butterfly-based direct solver for analyzing scattering from two-dimensional objects, IEEE Antennas Wirel. Propag. Lett. 16 (2017) 1179–1183.
[27] M.W. Mahoney, Lecture notes on randomized linear algebra, arXiv:1608.04481 [cs.DS], 2016.
[28] E. Michielssen, A. Boag, A multilevel matrix decomposition algorithm for analyzing scattering from large structures, IEEE Trans. Antennas Propag. 44 (8) (Aug. 1996) 1086–1093.
[29] G. Nico, G. Palubinskas, M. Datcu, Bayesian approaches to phase unwrapping: theoretical study, IEEE Trans. Signal Process. 48 (9) (2000).
[30] M. O'Neil, F. Woolfe, V. Rokhlin, An algorithm for the rapid evaluation of special function transforms, Appl. Comput. Harmon. Anal. 28 (2) (2010) 203–226.
[31] R. Platte, L. Trefethen, A. Kuijlaars, Impossibility of fast stable approximation of analytic functions from equispaced samples, SIAM Rev. 53 (2) (2011) 308–318.

[32] J.L. Rousseau, Fourier-integral-operator approximation of solutions to first-order hyperbolic pseudodifferential equations I: convergence in Sobolev spaces, Commun. Partial Differ. Equ. 31 (6) (2006) 867–906.
[33] J.L. Rousseau, G. Hörmann, Fourier-integral-operator approximation of solutions to first-order hyperbolic pseudodifferential equations II: microlocal analysis, J. Math. Pures Appl. 86 (5) (2006) 403–426.
[34] D. Ruiz-Antolín, A. Townsend, A nonuniform fast Fourier transform based on low rank approximation, SIAM J. Sci. Comput. 40 (1) (2018) A529–A547.
[35] D.O. Trad, T.J. Ulrych, M.D. Sacchi, Accurate interpolation with high-resolution time-variant Radon transforms, Geophysics 67 (2) (2002) 644–656.
[36] E. Trouvé, J.-M. Nicolas, H. Maître, Improving phase unwrapping techniques by the use of local frequency estimates, IEEE Trans. Geosci. Remote Sens. 36 (6) (1998).
[37] M. Tygert, Fast algorithms for spherical harmonic expansions, {III}, J. Comput. Phys. 229 (18) (2010) 6181–6192.
[38] C. Van Loan, Computational Frameworks for the Fast Fourier Transform, Society for Industrial and Applied Mathematics, 1992.
[39] L. Ying, Sparse Fourier transform via butterfly algorithm, SIAM J. Sci. Comput. 31 (3) (Feb. 2009) 1678–1694.