BUTTERFLY FACTORIZATION[*]

YINGZHOU LI[†], HAIZHAO YANG[‡], EILEEN R. MARTIN[†], KENNETH L. HO[‡], AND LEXING YING[§]

Abstract. The paper introduces the butterfly factorization as a data-sparse approximation for the matrices that satisfy a complementary low-rank property. The factorization can be constructed efficiently if either fast algorithms for applying the matrix and its adjoint are available or the entries of the matrix can be sampled individually. For an $N \times N$ matrix, the resulting factorization is a product of $O(\log N)$ sparse matrices, each with $O(N)$ nonzero entries. Hence, it can be applied rapidly in $O(N \log N)$ operations. Numerical results are provided to demonstrate the effectiveness of the butterfly factorization and its construction algorithms.

Key words. data-sparse matrix, butterfly algorithm, randomized algorithm, matrix factorization, operator compression, Fourier integral operators, special functions

AMS subject classifications. 44A55, 65R10, 65T50

DOI. 10.1137/15M1007173

1. Introduction. One of the key problems in scientific computing is the rapid evaluation of dense matrix-vector multiplication. Given a matrix $K \in \mathbb{C}^{N \times N}$ and a vector $g \in \mathbb{C}^N$, the direct computation of the vector $u = Kg \in \mathbb{C}^N$ takes $O(N^2)$ operations since each entry of K contributes to the result. This type of dense multiplication problem appears widely in special function transforms, integral transforms and equations, data fitting and smoothing, etc. Since the number of unknowns N is typically quite large in these applications, a lot of work has been devoted to performing this computation more efficiently without sacrificing the accuracy. Such a reduction in computational complexity depends highly on the algebraic and numerical properties of the matrix K. For certain types of matrices K, such as the Fourier matrix, numerically low-rank matrices, hierarchically semiseparable (HSS) matrices [20], and hierarchical matrices [5, 4], there exist fast algorithms for computing Kg accurately in $O(N \log N)$ or even $O(N)$ operations.

1.1. Complementary low-rank matrices and butterfly algorithm. Recent work in this area has identified yet another class of matrices for which fast $O(N \log N)$ application algorithms are available. These matrices satisfy a special kind of complementary low-rank property. For such a matrix, the rows are typically indexed by a set of points, say X, and the columns by another set of points, say Ω. Both X and Ω are often point sets in $\mathbb{R}^d$ for some dimension d. Associated with X and Ω are two

[†]Institute for Computational & Mathematical Engineering, Stanford University, Stanford, CA 94305 (ryanli@stanford.edu, ermartin@stanford.edu). The third author was supported in part by DOE grant DE-FG02-97ER25308.

[‡]Department of Mathematics, Stanford University, Stanford, CA 94305 (haizhao@stanford.edu, klho@stanford.edu). The fourth author was supported by the National Science Foundation under award DMS-1203554.

[§]Department of Mathematics and Institute for Computational & Mathematical Engineering, Stanford University, Stanford, CA 94305 (lexing@stanford.edu).

trees T_X and T_Ω, respectively, and both trees are assumed to have the same depth $L = O(\log N)$, with the top level being level 0 and the bottom one being 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, i.e., for a given precision ϵ there exists a low-rank approximation of $K_{A,B}$ with the 2-norm error bounded by ϵ and the rank bounded polynomially in $\log N$ and $\log(1/\epsilon)$. In many applications, one can even show that the rank is only bounded polynomially in $\log(1/\epsilon)$ and is independent of N. While it is straightforward to generalize the concept of the complementary low-rank property to a matrix with different row and column dimensions, the following discussion is restricted to the square matrices for simplicity.

A simple yet important example is the Fourier matrix K of size $N \times N$, where

$$X = \Omega = \{0, \dots, N - 1\},$$
$$K = (\exp(2\pi \imath jk/N))_{0 \le j,k < N}.$$

Here the trees T_X and T_Ω are generated by bisecting the sets X and Ω recursively. Both trees have the same depth $L = \log_2 N$. For each pair of nodes $A \in T_X$ and $B \in T_\Omega$ with A at level ℓ and B at level $L - \ell$, the numerical rank of the submatrix $K_{A,B}$ for a fixed precision ϵ is bounded by a number that is independent of N and scales linearly with respect to $\log(1/\epsilon)$ [14].

For complementary low-rank matrices, the matrix-vector multiplication can be carried out efficiently via the *butterfly algorithm*, which was initially proposed in [13] and later extended in [14]. For a general matrix K of this type, the butterfly algorithm consists of two stages: the off-line stage and the on-line stage. In the off-line stage, it conducts simultaneously a top down traversal of T_X and a bottom up traversal of T_Ω (see Figure 1 for an interpretation of data flows) to recursively compress all complementary low-rank submatrices (see Figure 2 for an example of necessary submatrices). This typically takes $O(N^2)$ operations [14, 16] for a general complementary low-rank matrix K. In the on-line stage, the butterfly algorithm then evaluates $u = Kg$ for a given input vector $g \in \mathbb{C}^N$ in $O(N \log N)$ operations. While the on-line application cost is essentially linear, the $O(N^2)$ off-line precomputation cost appears to be a major bottleneck for many calculations. For certain complementary low-rank matrices, such as those obtained from the Fourier integral operators (FIOs) [1, 8, 15], the sparse Fourier transforms [23], and the numerical solutions of acoustic wave equations [2], the off-line precomputation cost can be reduced to nearly linear or even totally eliminated. However, in all these cases, the reduction relies heavily on strong assumptions on the analytic properties of the kernel function of K. When such detailed information is not available, we are then forced to fall back on the $O(N^2)$ off-line precomputation algorithm.

1.2. Motivations and significance. A natural question is whether it is still possible to reduce the cost of the precomputation stage if the analytic properties of the kernel are not accessible. The following two cases are quite common in applications:

1. Only black-box routines for computing Kg and K^*g in $O(N \log N)$ operations are given.
2. Only a black-box routine for evaluating any entry of the matrix K in $O(1)$ operations is given.
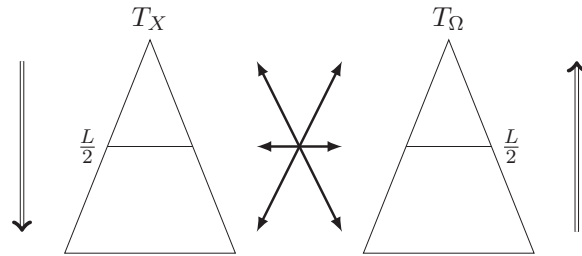
FIG. 1. *Trees of the row and column indices. Left: T_X for the row indices X. Right: T_Ω for the column indices Ω. The interaction between $A \in T_X$ and $B \in T_\Omega$ starts at the root of T_X and the leaves of T_Ω.*
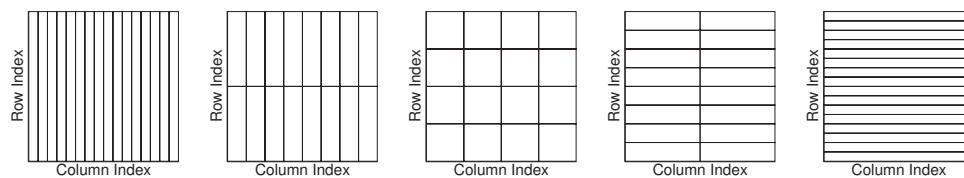

FIG. 2. *Hierarchical decomposition of the row and column indices of a 16×16 matrix. The trees T_X and T_Ω have roots containing 16 column and row indices and leaves containing a single column and row index. The rectangles above indicate the submatrices satisfying the complementary low-rank property.*

To answer this question, this paper proposes the *butterfly factorization*, which represents K as a product of $L + 3$ sparse matrices:

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

where the depth $L = O(\log N)$ of T_X and T_Ω is assumed to be even, $h = L/2$ is a middle level index, and all factors are sparse matrices with $O(N)$ nonzero entries.

The construction of the butterfly factorization proceeds as follows in two stages. The first stage is to construct a preliminary middle level factorization that is associated with the middle level of T_X and T_Ω,

$$(2) \qquad K \approx U^h M^h (V^h)^*,$$

where U^h and V^h are block diagonal matrices and M^h is a weighted permutation matrix. In the first case, this is achieved by applying K to a set of $O(N^{1/2})$ structured random vectors and then applying the randomized singular value decomposition (SVD) to the result. This typically costs $O(N^{3/2} \log N)$ operations. In the second case, (2) is built via the randomized sampling method proposed in [3, 21] for computing approximate SVDs. This randomized sampling needs to make the assumption that the columns and rows of middle level blocks of K are incoherent with respect to the delta functions, and it typically takes only $O(N^{3/2})$ operations in practice.

Once the middle level factorization (2) is available, the second stage is a sequence of truncated SVDs that further factorize each of U^h and V^h into a sequence of sparse matrices, resulting in the final factorization (1). The operation count of this stage is $O(N^{3/2})$, and the total memory complexity for constructing butterfly factorization is $O(N^{3/2})$.

When the butterfly factorization (1) is constructed, the cost of applying K to a given vector $g \in \mathbb{C}^N$ is $O(N \log N)$ because (1) is a sequence of $O(\log N)$ sparse matrices, each with $O(N)$ nonzero entries. Although we shall limit our discussion to one-dimensional problems in this paper, the proposed butterfly factorization, along with its construction algorithm, can be easily generalized to higher dimensions.

This work is motivated by problems that require repeated applications of a butterfly algorithm. In several applications, such as inverse scattering [17, 22] and fast spherical harmonic transform (SHT) [18], the butterfly algorithm is called repeatedly either in an iterative process of minimizing some regularized objective function or to a large set of different input vectors. Therefore, it becomes important to reduce the constant prefactor of the butterfly algorithm to save actual running time. For example, in [1], Chebyshev interpolation is applied to recover low-rank structures of submatrices with a sufficiently large number of interpolation points. The recovered rank is far from the optimum. Hence, the prefactor of the corresponding butterfly algorithm in [1] is large. The butterfly factorization can further compress this butterfly algorithm to obtain nearly optimal low-rank approximations resulting in a much smaller prefactor, as will be shown in the numerical results. Therefore, it is more efficient to construct the butterfly factorization using this butterfly algorithm and then apply the butterfly factorization repeatedly. In this sense, the butterfly factorization can be viewed as a compression of certain butterfly algorithms.

Another important application is the computation of a composition of several FIOs. A direct method to construct the composition takes $O(N^3)$ operations, while the butterfly factorization provides a data-sparse representation of this composition in $O(N^{3/2} \log N)$ operations, once the fast algorithm for applying each FIO is available. After the construction, the application of the butterfly factorization is independent of the number of FIOs in the composition, which is significant when the number of FIOs is large.

Recently, there has also been a sequence of papers on recovering a structured matrix via applying it to (structured) random vectors. For example, the randomized SVD algorithms [6, 9, 19] recover a low-rank approximation to an unknown matrix when it is numerically low-rank. The work in [12] constructs a sparse representation for an unknown HSS matrix. More recently, [10] considers the more general problem of constructing a sparse representation of an unknown $\mathcal{H}$-matrix. To the best of our knowledge, the present paper is the first to address such a matrix recovery problem if the unknown matrix satisfies the complementary low-rank property.

1.3. Content. The rest of this paper is organized as follows. Section 2 briefly reviews some basic tools that shall be used repeatedly in section 3. Section 3 describes in detail the butterfly factorization and its construction algorithm. In section 4, numerical examples are provided to demonstrate the efficiency of the proposed algorithms. Finally, section 5 lists several directions for future work.

2. Preliminaries. For a matrix $Z \in \mathbb{C}^{m \times n}$, we define a rank-$r$ approximate SVD of Z as

$$Z \approx U_0 \Sigma_0 V_0^*,$$

where $U_0 \in \mathbb{C}^{m \times r}$ is unitary, $\Sigma_0 \in \mathbb{R}^{r \times r}$ is diagonal, and $V_0 \in \mathbb{C}^{n \times r}$ is unitary. A straightforward method to obtain the optimal rank-r approximation of Z is to compute its truncated SVD, where U_0 is the matrix with the first r left singular vectors, Σ_0 is a diagonal matrix with the first r singular values in decreasing order, and V_0 is the matrix with the first r right singular vectors.

A typical computation of the truncated SVD of Z takes $O(mn\min(m,n))$ operations, which can be quite expensive when m and n are large. Therefore, a lot of research has been devoted to faster algorithms for computing approximate SVDs, especially for matrices with rapidly decaying singular values. In sections 2.1 and 2.2, we will introduce two randomized algorithms for computing approximate SVDs for numerically low-rank matrices Z: the first one [6] is based on applying the matrix to random vectors, while the second [3, 21] relies on sampling the matrix entries randomly.

Once an approximate SVD $Z \approx U_0\Sigma_0 V_0^*$ is computed, it can be written in several equivalent ways, each of which is convenient for certain purposes. First, one can write

$$Z \approx USV^*,$$

where

$$(3) \qquad U = U_0\Sigma_0, \quad S = \Sigma_0^{-1}, \quad \text{and} \quad V^* = \Sigma_0 V_0^*.$$

This construction is analogous to the well-known CUR decomposition [11] in the sense that the left and right factors in both factorization methods inherit similar singular values of the original numerical low-rank matrix. Here, the middle matrix S in (3) can be carefully constructed to ensure numerical stability, since the singular values in Σ_0 can be computed to nearly full relative precision.

As we shall see, sometimes it is also convenient to write the approximation as

$$Z \approx UV^*,$$

where

$$(4) \qquad U = U_0 \quad \text{and} \quad V^* = \Sigma_0 V_0^*,$$

or

$$(5) \qquad U = U_0\Sigma_0 \quad \text{and} \quad V^* = V_0^*.$$

One of the factors U and V^* absorbs the singular values of Z.

2.1. SVD via random matrix-vector multiplication. One popular approach is the randomized algorithm in [6] that reduces the cubic complexity to $O(rmn)$ complexity. We briefly review this following [6] for constructing a rank-r approximation SVD $Z \approx U_0\Sigma_0 V_0^*$ below.

ALGORITHM 2.1 (randomized SVD).
1. Generate two tall, skinny random Gaussian matrices $R_{col} \in \mathbb{C}^{n\times(r+p)}$ and $R_{row} \in \mathbb{C}^{m\times(r+p)}$, where $p = O(1)$ is an additive oversampling parameter that increases the approximation accuracy.
2. Apply the pivoted QR factorization to ZR_{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^*R_{row}, and let Q_{row} be the matrix of the first r columns of the Q-matrix.
3. Generate a tiny middle matrix $M = Q_{col}^*ZQ_{row}$, and compute its rank-r truncated SVD: $M \approx U_M\Sigma_M V_M^*$.
4. Let $U_0 = Q_{col}U_M$, $\Sigma_0 = \Sigma_M$, and $V_0^* = V_M^*Q_{row}^*$. Then $Z \approx U_0\Sigma_0 V_0^*$.

The dominant complexity comes from the application of Z to $O(r)$ random vectors. If fast algorithms for applying Z are available, the quadratic complexity can be further reduced.

Once the approximate SVD of Z is ready, the equivalent forms in (3), (4), and (5) can be constructed easily. Under the condition that the singular values of Z decay sufficiently rapidly, the approximation error of the resulting rank-r is nearly optimal with an overwhelming probability. Typically, the additive over-sampling parameter $p = 5$ is sufficient to obtain an accurate rank-r approximation of Z.

For most applications, the goal is to construct a low-rank approximation up to a fixed relative precision ϵ, rather than a fixed rank r. The above procedure can then be embedded into an iterative process that starts with a relatively small r, computes a rank-r approximation, estimates the error probabilistically, and repeats the steps with doubled rank $2r$ if the error is above the threshold ϵ [6].

2.2. SVD via random sampling. The above algorithm relies only on the product of the matrix $Z \in \mathbb{C}^{m \times n}$ or its transpose with given random vectors. If one is allowed to access the individual entries of Z, the following randomized sampling method for low-rank approximations introduced in [3, 21] can be more efficient. This method visits only $O(r)$ columns and rows of Z and hence requires only $O(r(m+n))$ 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 also use ":" to denote the entire columns or rows of the matrix, i.e., $Z_{I,:} = Z(I,:)$ and $Z_{:,J} = Z(:,J)$. With this handy notation, we briefly introduce the randomized sampling algorithm to construct a rank-r approximation of $Z \approx U_0 \Sigma_0 V_0^*$.

ALGORITHM 2.2 (randomized sampling for low-rank approximation).
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

$$Z_{I,:}P = QR,$$

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

$$PZ_{:,J} = LQ,$$

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

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

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

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

$$(6) \qquad U_0 = Q_{col} U_M, \quad \Sigma_0 = \Sigma_M, \quad V_0^* = V_M^* Q_{row}^*.$$

We have not been able to quantify the error and success probability rigorously for this procedure at this point. On the other hand, when the columns and rows of K are incoherent with respect to "delta functions" (i.e., vectors that have only one significantly larger entry), this procedure works well in our numerical experiments. Here, a vector u is said to be incoherent with respect to a vector v if $\mu = |u^T v|/(\|u\|_2 \|v\|_2)$ is small. 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. These parameters are empirically sufficient to achieve accurate low-rank approximations and are used throughout numerical examples in section 4.

As we mentioned above, for most applications the goal is to construct a low-rank approximation 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.

3. Butterfly factorization. This section presents the butterfly factorization algorithm for a matrix $K \in \mathbb{C}^{N \times N}$. For simplicity let $X = \Omega = \{1, \ldots, N\}$. The trees T_X and T_Ω are complete binary trees with $L = \log_2 N - O(1)$ levels. We assume that L is an even integer and the number of points in each leaf node of T_X and T_Ω is bounded by a uniform constant.

At each level ℓ, $\ell = 0, \ldots, L$, we denote the ith node at level ℓ in T_X as A_i^ℓ for $i = 0, 1, \ldots, 2^\ell - 1$ and the jth node at level $L - \ell$ in T_Ω as $B_j^{L-\ell}$ for $j = 0, 1, \ldots, 2^{L-\ell} - 1$. These nodes naturally partition K into $O(N)$ submatrices $K_{A_i^\ell, B_j^{L-\ell}}$. For simplicity, we write $K_{i,j}^\ell := K_{A_i^\ell, B_j^{L-\ell}}$, where the superscript is used to indicate the level (in T_X). The butterfly factorization utilizes rank-r approximations of all submatrices $K_{i,j}^\ell$ with $r = O(1)$.

The butterfly factorization of K is built in two stages. In the first stage, we compute a rank-r approximation of each submatrix $K_{i,j}^h$ at the level $\ell = h = L/2$ and then organize them into an initial factorization:

$$K \approx U^h M^h (V^h)^*,$$

where U^h and V^h are block diagonal matrices and M^h is a weighted permutation matrix. This is referred to as the *middle level factorization* and is described in detail in section 3.1.

In the second stage, we recursively factorize $U^\ell \approx U^{\ell+1} G^\ell$ and $(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^*$ for $\ell = h, h+1, \ldots, L-1$, since U^ℓ and $(V^\ell)^*$ inherit the complementary low-rank

property from K; i.e., the low-rank property of U^ℓ comes from the low-rank property of $K_{i,j}^\ell$, and the low-rank property of V^ℓ results from that of $K_{i,j}^{L-\ell}$. After this recursive factorization, one reaches the butterfly factorization of K:

$$(7) \qquad K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*,$$

where all factors are sparse matrices with $O(N)$ nonzero entries. We refer to this stage as the *recursive factorization* and discuss it in detail in section 3.2.

3.1. Middle level factorization. The first step of the middle level factorization is to compute a rank-r approximation to every $K_{i,j}^h$. Recall that we consider one of the following two cases:

1. Only black-box routines for computing Kg and K^*g in $O(N \log N)$ operations are given.
2. Only a black-box routine for evaluating any entry of the matrix K in $O(1)$ operations is given.

The actual computation of this step proceeds differently depending on which case is under consideration. Throughout the discussion, $m = 2^h = O(N^{1/2})$ is the number of nodes in the middle level $h = L/2$, and we assume without loss of generality that N/m is an integer.

- In the first case, the rank-r approximation of each $K_{i,j}^h$ is constructed with the SVD algorithm via random matrix-vector multiplication in section 2.1. This requires us to apply $K_{i,j}^h$ and its adjoint to random Gaussian matrices of size $(N/m) \times (r+p)$, where r is the desired rank and p is an oversampling parameter. In order to take advantage of the fast algorithm for multiplying K, we construct a matrix C of size $N \times m(r+p)$. C is partitioned into an $m \times m$ block matrix with each block C_{ij} for $i, j = 0, 1, \ldots, m-1$ of size $(N/m) \times (r+p)$. In addition, C is block diagonal, and its diagonal blocks are random Gaussian matrices. This is equivalent to applying each $K_{i,j}^h$ to the same random Gaussian matrix C_{jj} for all i. We then use the fast algorithm to apply K to each column of C and store the results. Similarly, we form another random block diagonal matrix R similar to C and use the fast algorithm of applying K^* to R. This is equivalent to applying each $(K_{i,j}^h)^*$ to an $(N/m) \times (r+p)$ Gaussian random matrix R_{ii} for all $j = 0, 1, \ldots, m-1$. With $K_{i,j}^h C_{jj}$ and $(K_{i,j}^h)^* R_{ii}$ ready, we can compute the rank-r approximate SVD of $K_{i,j}^h$ following the procedure described in section 2.1.
- In the second case, it is assumed that an arbitrary entry of K can be calculated in $O(1)$ operations. We simply apply the SVD algorithm via random sampling in section 2.2 to each $K_{i,j}^h$ to construct a rank-r approximate SVD.

In either case, once the approximate SVD of $K_{i,j}^h$ is ready, it is transformed in the form

$$K_{i,j}^h \approx U_{i,j}^h S_{i,j}^h (V_{j,i}^h)^*$$

following (3). We would like to emphasize that the columns of $U_{i,j}^h$ and $V_{j,i}^h$ are scaled with the singular values of the approximate SVD so that they keep track of the importance of these columns in approximating $K_{i,j}^h$.

After calculating the approximate rank-r factorization of each $K_{i,j}^h$, we assemble these factors into three block matrices U^h, M^h, and V^h as follows:

(8)

$$
\begin{aligned}
K &\approx \begin{pmatrix}
U_{0,0}^h S_{0,0}^h (V_{0,0}^h)^* & U_{0,1}^h S_{0,1}^h (V_{1,0}^h)^* & \cdots & U_{0,m-1}^h S_{0,m-1}^h (V_{m-1,0}^h)^* \\
U_{1,0}^h S_{1,0}^h (V_{0,1}^h)^* & U_{1,1}^h S_{1,1}^h (V_{1,1}^h)^* & & U_{1,m-1}^h S_{1,m-1}^h (V_{m-1,1}^h)^* \\
\vdots & & \ddots & \\
U_{m-1,0}^h S_{m-1,0}^h (V_{0,m-1}^h)^* & U_{m-1,1}^h S_{m-1,1}^h (V_{1,m-1}^h)^* & & U_{m-1,m-1}^h S_{m-1,m-1}^h (V_{m-1,m-1}^h)^*
\end{pmatrix} \\
&= \begin{pmatrix}
U_0^h & & & \\
& U_1^h & & \\
& & \ddots & \\
& & & U_{m-1}^h
\end{pmatrix}
\begin{pmatrix}
M_{0,0}^h & M_{0,1}^h & \cdots & M_{0,m-1}^h \\
M_{1,0}^h & M_{1,1}^h & & M_{1,m-1}^h \\
\vdots & & \ddots & \\
M_{m-1,0}^h & M_{m-1,1}^h & & M_{m-1,m-1}^h
\end{pmatrix}
\begin{pmatrix}
(V_0^h)^* & & & \\
& (V_1^h)^* & & \\
& & \ddots & \\
& & & (V_{m-1}^h)^*
\end{pmatrix} \\
&= U^h M^h (V^h)^*,
\end{aligned}
$$

where

(9)
$$
\begin{aligned}
U_i^h &= \begin{pmatrix} U_{i,0}^h & U_{i,1}^h & \cdots & U_{i,m-1}^h \end{pmatrix} \in \mathbb{C}^{(N/m) \times mr}, \\
V_j^h &= \begin{pmatrix} V_{j,0}^h & V_{j,1}^h & \cdots & V_{j,m-1}^h \end{pmatrix} \in \mathbb{C}^{(N/m) \times mr},
\end{aligned}
$$

and $M^h \in \mathbb{C}^{(m^2 r) \times (m^2 r)}$ is a weighted permutation matrix. Each submatrix $M_{i,j}^h$ is itself an $m \times m$ block matrix with block size $r \times r$, where all blocks are zero except that the (j,i) block is equal to the diagonal matrix $S_{i,j}^h$. It is obvious that there are only $O(N)$ nonzero entries in M^h. See Figure 3 for an example of a middle level factorization of a 64×64 matrix with $r = 1$.
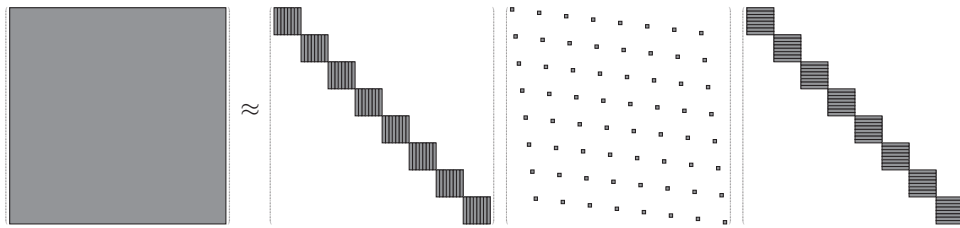

FIG. 3. *The middle level factorization of a 64×64 complementary low-rank matrix $K \approx U^3 M^3 (V^3)^*$ assuming $r = 1$. Gray blocks indicate nonzero blocks. U^3 and V^3 are block diagonal matrices with eight blocks. The diagonal blocks of U^3 and V^3 are assembled according to (9) as indicated by black rectangles. M^3 is an 8×8 block matrix with each block $M_{i,j}^3$ itself an 8×8 block matrix containing a diagonal weight matrix on the (j,i) block.*

3.2. Recursive factorization. In this section, we will recursively factorize

(10)
$$
U^\ell \approx U^{\ell+1} G^\ell
$$

for $\ell = h, h+1, \ldots, L-1$ and

(11)
$$
(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^*
$$

for $\ell = h, h+1, \ldots, L-1$. After these recursive factorizations, we can obtain the following butterfly factorization by substituting these factorizations into (8):

(12)
$$
K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*.
$$

3.2.1. Recursive factorization of U^h. Each factorization at level ℓ in (10) results from the low-rank property of $K_{i,j}^\ell$ for $\ell \geq L/2$. When $\ell = h$, recall that

$$U^h = \begin{pmatrix} U_0^h & & & \\ & U_1^h & & \\ & & \ddots & \\ & & & U_{m-1}^h \end{pmatrix}$$

and

$$U_i^h = \begin{pmatrix} U_{i,0}^h & U_{i,1}^h & \cdots & U_{i,m-1}^h \end{pmatrix}$$

with each $U_{i,j}^h \in \mathbb{C}^{(N/m) \times r}$. We split U_i^h and each $U_{i,j}^h$ into halves by row, i.e.,

$$U_i^h = \begin{pmatrix} U_i^{h,t} \\ \hline U_i^{h,b} \end{pmatrix} \quad \text{and} \quad U_{i,j}^h = \begin{pmatrix} U_{i,j}^{h,t} \\ \hline U_{i,j}^{h,b} \end{pmatrix},$$

where the superscript t denotes the top half and b denotes the bottom half of a matrix. Then we have

$$(13) \qquad U_i^h = \begin{pmatrix} U_{i,0}^{h,t} & U_{i,1}^{h,t} & \cdots & U_{i,m-1}^{h,t} \\ \hline U_{i,0}^{h,b} & U_{i,1}^{h,b} & \cdots & U_{i,m-1}^{h,b} \end{pmatrix}.$$

Notice that, for each $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, m/2 - 1$, the columns of

$$(14) \qquad \begin{pmatrix} U_{i,2j}^{h,t} & U_{i,2j+1}^{h,t} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} U_{i,2j}^{h,b} & U_{i,2j+1}^{h,b} \end{pmatrix}$$

in (13) are in the column space of $K_{2i,j}^{h+1}$ and $K_{2i+1,j}^{h+1}$, respectively. By the complementary low-rank property of the matrix K, $K_{2i,j}^{h+1}$ and $K_{2i+1,j}^{h+1}$ are numerical low-rank. Hence $(U_{i,2j}^{h,t} U_{i,2j+1}^{h,t})$ and $(U_{i,2j}^{h,b} U_{i,2j+1}^{h,b})$ are numerically low-rank matrices in $\mathbb{C}^{(N/2m) \times 2r}$. Compute their rank-$r$ approximations by the standard truncated SVD, transform them into the form of (5), and denote them as

$$(15) \qquad \begin{pmatrix} U_{i,2j}^{h,t} & U_{i,2j+1}^{h,t} \end{pmatrix} \approx U_{2i,j}^{h+1} G_{2i,j}^h \quad \text{and} \quad \begin{pmatrix} U_{i,2j}^{h,b} & U_{i,2j+1}^{h,b} \end{pmatrix} \approx U_{2i+1,j}^{h+1} G_{2i+1,j}^h$$

for $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, m/2-1$. The matrices in (15) can be assembled into two new sparse matrices such that

$$U^h \approx U^{h+1} G^h = \begin{pmatrix} U_0^{h+1} & & & \\ & U_1^{h+1} & & \\ & & \ddots & \\ & & & U_{2m-1}^{h+1} \end{pmatrix} \begin{pmatrix} G_0^h & & & \\ & G_1^h & & \\ & & \ddots & \\ & & & G_{m-1}^h \end{pmatrix},$$

where

$$U_i^{h+1} = \begin{pmatrix} U_{i,0}^{h+1} & U_{i,1}^{h+1} & \cdots & U_{i,m/2-1}^{h+1} \end{pmatrix}$$

for $i = 0, 1, \ldots, 2m-1$, and

$$
G_i^h = \left(\begin{array}{cccc|cccc}
G_{2i,0}^h & & & & & & & \\
& G_{2i,1}^h & & & & & & \\
& & \ddots & & & & & \\
& & & G_{2i,m/2-1}^h & & & & \\
\hline
G_{2i+1,0}^h & & & & & & & \\
& G_{2i+1,1}^h & & & & & & \\
& & \ddots & & & & & \\
& & & G_{2i+1,m/2-1}^h & & & &
\end{array}\right)
$$

for $i = 0, 1, \ldots, m-1$.

Since there are $O(1)$ nonzero entries in each $G_{i,j}^h$ and there are $O(N)$ such submatrices, there are only $O(N)$ nonzero entries in G^h. See Figure 4 (top) for an example of the factorization $U^h \approx U^{h+1} G^h$ for the left factor U^h with $L = 6$, $h = 3$, and $r = 1$ in Figure 3.
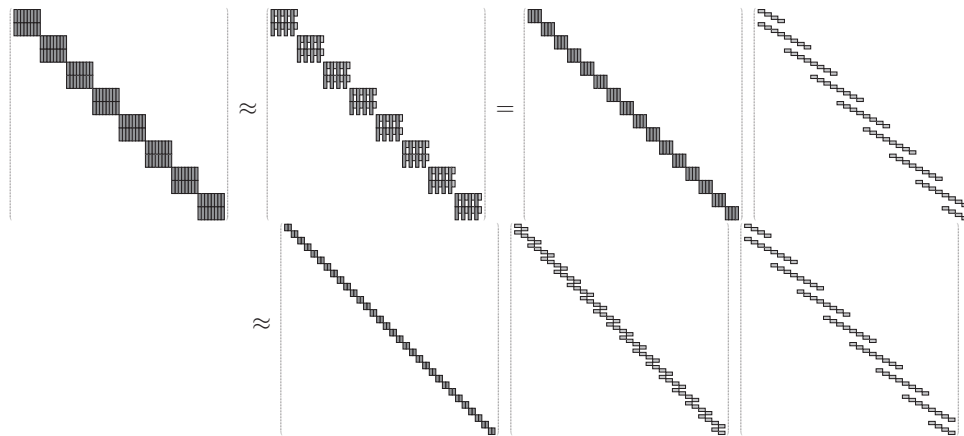

FIG. 4. *The recursive factorization of U^3 in Figure 3. Gray factors are matrices inheriting the complementary low-rank property. Top row: left matrix: U^3 with each diagonal block partitioned into smaller blocks according to* (13) *as indicated by black rectangles; middle-left matrix: low-rank approximations of submatrices in U^3 given by* (15)*; middle-right matrix: U^4; right matrix: G^3. Bottom row: U^4 from the top row is further factorized into $U^4 \approx U^5 G^4$, giving $U^3 \approx U^5 G^4 G^3$.*

Similarly, for any ℓ between h and $L-1$, we can factorize $U^\ell \approx U^{\ell+1} G^\ell$, because the columns in $(U_{i,2j}^{\ell,t} U_{i,2j+1}^{\ell,t})$ and $(U_{i,2j}^{\ell,b} U_{i,2j+1}^{\ell,b})$ are in the column space of the numerically low-rank matrices $K_{2i,j}^{\ell+1}$ and $K_{2i+1,j}^{\ell+1}$, respectively. Computing the rank-r approximations via the standard truncated SVD and transforming them into the form of (5) gives

$$(16) \qquad \left(U_{i,2j}^{\ell,t} \quad U_{i,2j+1}^{\ell,t}\right) \approx U_{2i,j}^{\ell+1} G_{2i,j}^\ell \quad \text{and} \quad \left(U_{i,2j}^{\ell,b} \quad U_{i,2j+1}^{\ell,b}\right) \approx U_{2i+1,j}^{\ell+1} G_{2i+1,j}^\ell$$

for $i = 0, 1, \ldots, 2^\ell - 1$ and $j = 0, 1, \ldots, 2^{L-\ell-1} - 1$. After assembling these factoriza-

tions together, we obtain

$$U^\ell \approx U^{\ell+1} G^\ell = \begin{pmatrix} U_0^{\ell+1} & & & \\ & U_1^{\ell+1} & & \\ & & \ddots & \\ & & & U_{2^{\ell+1}-1}^{\ell+1} \end{pmatrix} \begin{pmatrix} G_0^\ell & & & \\ & G_1^\ell & & \\ & & \ddots & \\ & & & G_{2^\ell-1}^\ell \end{pmatrix},$$

where

$$U_i^{\ell+1} = \begin{pmatrix} U_{i,0}^{\ell+1} & U_{i,1}^{\ell+1} & \cdots & U_{i,2^{L-\ell-1}-1}^{\ell+1} \end{pmatrix}$$

for $i = 0, 1, \ldots, 2^{\ell+1} - 1$, and

$$G_i^\ell = \left(\begin{array}{ccccc} G_{2i,0}^\ell & & & & \\ & G_{2i,1}^\ell & & & \\ & & \ddots & & \\ & & & & G_{2i,2^{L-\ell-1}-1}^\ell \\ \hline G_{2i+1,0}^\ell & & & & \\ & G_{2i+1,1}^\ell & & & \\ & & \ddots & & \\ & & & & G_{2i+1,2^{L-\ell-1}-1}^\ell \end{array} \right)$$

for $i = 0, 1, \ldots, 2^\ell - 1$.

After $L - h$ steps of recursive factorizations

$$U^\ell \approx U^{\ell+1} G^\ell$$

for $\ell = h, h+1, \ldots, L-1$, we obtain the recursive factorization of U^h as

$$U^h \approx U^L G^{L-1} \cdots G^h. \tag{17}$$

See Figure 4 (bottom) for an example of a recursive factorization for the left factor U^h with $L = 6$, $h = 3$, and $r = 1$ in Figure 3.

Similar to the analysis of G^h, it is also easy to check that there are only $O(N)$ nonzero entries in each G^ℓ in (17). Since there are $O(N)$ diagonal blocks in U^L and each block contains $O(1)$ entries, there are $O(N)$ nonzero entries in U^L.

3.2.2. Recursive factorization of V^h. The recursive factorization of V^h is similar to that of U^h. In each step of the factorization

$$(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^*,$$

we take advantage of the low-rank property of the row space of $K_{i,2j}^{L-\ell-1}$ and $K_{i,2j+1}^{L-\ell-1}$ to obtain rank-r approximations. Applying the exact same procedure of section 3.2.1 now to V^ℓ leads to the recursive factorization $V^h \approx V^L H^{L-1} \cdots H^h$ or, equivalently,

$$(V^h)^* \approx (H^h)^* \cdots (H^{L-1})^* (V^L)^*, \tag{18}$$

with all factors containing only $O(N)$ nonzero entries. See Figure 5 for an example of a recursive factorization $(V^h)^* \approx (H^h)^* \cdots (H^{L-2})^* (V^{L-1})^*$ for the left factor V^h with $L = 6$, $h = 3$, and $r = 1$ in Figure 3.
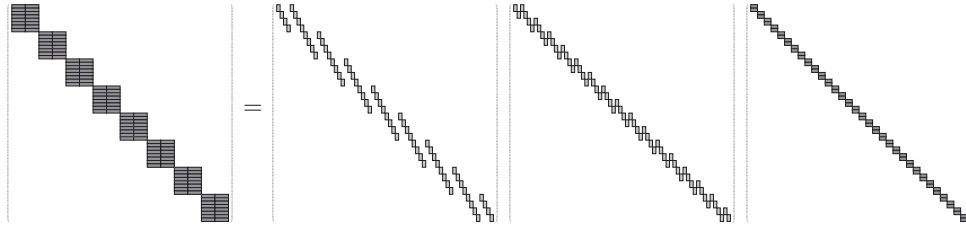
FIG. 5. *The recursive factorization* $(V^3)^* \approx (H^3)^*(H^4)^*(V^5)^*$ *of* $(V^3)^*$ *in Figure* 3.

Given the recursive factorization of U^h and $(V^h)^*$ in (17) and (18), we reach the butterfly factorization

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

where all factors are sparse matrices with $O(N)$ nonzero entries. For a given input vector $g \in \mathbb{C}^N$, the $O(N^2)$ matrix-vector multiplication $u = Kg$ can be approximated by a sequence of $O(\log N)$ sparse matrix-vector multiplications given by the butterfly factorization.

3.3. Complexity analysis. The complexity analysis of the construction of a butterfly factorization naturally consists of two parts: the middle level factorization and the recursive factorization.

The complexity of the middle level factorization depends on which one of the cases is under consideration.

- For the first case, the approximate SVDs are determined by the application of K and K^* to Gaussian random matrices in $\mathbb{C}^{N \times N^{1/2}(r+p)}$ and the rank-r approximations of K_{ij}^h for each (i,j) pair. Assume that each matrix-vector multiplication by K or K^* via the given black-box routines requires $O(C_K(N))$ operations (which is at least $O(N)$). Then the dominant cost is due to applying K and K^* $O(N^{1/2})$ times, which yields an overall computational complexity of $O(C_K(N)N^{1/2})$.
- In the second case, the approximate SVDs are computed via random sampling for each K_{ij}^h of the $O(N)$ pairs (i,j). The complexity of performing randomized sampling for each such block is $O(N^{1/2})$. Hence, the overall computational complexity is $O(N^{3/2})$.

In the recursive factorization, U^ℓ at level ℓ consists of $O(2^\ell)$ diagonal blocks of size $O(N/2^\ell) \times O(N/2^\ell)$. In each diagonal block, there are $O(N/2^\ell)$ factorizations in (16). Since the operation complexity of performing one factorization in (16) is $O(N/2^\ell)$, it takes $O(N^2/2^\ell)$ operations to factorize U^ℓ. Summing up the operations at all levels gives the total complexity for recursively factorizing U^h:

$$(20) \qquad \sum_{\ell=h}^{L-1} O(N^2/2^\ell) = O(N^{3/2}).$$

Similarly, the operation complexity for recursively compressing V^h is also $O(N^{3/2})$.

The memory peak of the butterfly factorization occurs in the middle level factorization since we have to store the initial factorization in (8). There are $O(N^{3/2})$

nonzero entries in U^h and V^h, and $O(N)$ in M^h. Hence, the total memory complexity is $O(N^{3/2})$. The total operation complexity for constructing the butterfly factorization is summarized in Table 1.

TABLE 1
Computational complexity and memory complexity of the butterfly factorization. $C_K(N)$ is the operation complexity of one application of K or K^*. In most of the cases encountered, $C_K(N) = O(N \log N)$.

		Randomized SVD	Randomized sampling
Factorization complexity	Middle level factorization	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
	Recursive factorization	$O(N^{3/2})$	
	Total	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
Memory complexity		$O(N^{3/2})$	$O(N \log N)$
Application complexity		$O(N \log N)$	

It is worth pointing out that the memory complexity can be reduced to $O(N \log N)$ when we apply the randomized sampling method to construct each block in the initial factorization in (8) separately. Instead of factorizing U^h and V^h at the end of the middle level factorization, we can factorize the left and right factors U_i^h and V_i^h in (8) on the fly to avoid storing all factors in (8). For a fixed i, we generate U_i^h from K_{ij}^h for all j and recursively factorize U_i^h. The memory cost is $O(N)$ for storing U_i^h and $O(N^{1/2} \log N)$ for storing the sparse matrices after the recursive factorization. Repeating this process for $i = 1, \ldots, N^{1/2}$ gives the complete factorization of U^h. The factorization of V^h is conducted similarly. The total memory complexity is $O(N \log N)$.

The operation and memory complexity for the application of the butterfly factorization are governed by the number of nonzero entries in the factorization: $O(N \log N)$.

4. Numerical results. This section presents three numerical examples to demonstrate the effectiveness of the algorithms proposed above. The first example is an FIO in [1], and the second example is a special function transform in [14]. Both examples provide an explicit kernel function that becomes a one-dimensional complementary low-rank matrix after discretization. This allows us to apply the butterfly factorization construction algorithm with random sampling. The computational complexity and the memory cost are $O(N^{3/2})$ and $O(N \log N)$ in this case.

The third example is a composition of two FIOs for which an explicit kernel function of their composition is not available. Since we can apply either the butterfly algorithm in [1] or the butterfly factorization to evaluate these FIOs one by one, a fast algorithm for computing the composition is available. We apply the butterfly factorization construction algorithm with random matrix-vector multiplication to this example, which requires $O(N^{3/2} \log N)$ operations and $O(N^{3/2})$ memory cost.

Our implementation is in MATLAB. The numerical results were obtained on a server computer with a 2.0 GHz CPU. The additive oversampling parameter is $p = 5$, and the multiplicative oversampling parameter is $q = 3$.

Let $\{u^d(x), x \in X\}$ and $\{u^a(x), x \in X\}$ denote the results given by the direct matrix-vector multiplication and the butterfly factorization. The accuracy of applying

the butterfly factorization algorithm is estimated by the following relative error:

$$(21) \qquad \epsilon^a = \sqrt{\frac{\sum_{x \in S} |u^a(x) - u^d(x)|^2}{\sum_{x \in S} |u^d(x)|^2}},$$

where S is a point set of size 256 randomly sampled from X.

Example 1. Our first example is to evaluate a one-dimensional FIO of the following form:

$$(22) \qquad u(x) = \int_{\mathbb{R}} e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi) d\xi,$$

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

$$(23) \qquad \Phi(x,\xi) = x \cdot \xi + c(x)|\xi|, \quad c(x) = (2 + \sin(2\pi x))/8.$$

The discretization of (22) is

$$(24) \qquad u(x_i) = \sum_{\xi_j} e^{2\pi\imath\Phi(x_i,\xi_j)} \widehat{f}(\xi_j), \quad i,j = 1,2,\ldots,N,$$

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

$$(25) \qquad x_i = (i-1)/N \quad \text{and} \quad \xi_j = j - 1 - N/2.$$

Equation (24) can be represented in a matrix form as $u = Kg$, where $u_i = u(x_i)$, $K_{ij} = e^{2\pi\imath\Phi(x_i,\xi_j)}$, and $g_j = \widehat{f}(\xi_j)$. The matrix K satisfies the complementary low-rank property as proved in [1, 8]. The explicit kernel function of K allows us to use the construction algorithm with random sampling. Table 2 summarizes the results of this example for different grid sizes N and truncation ranks r.

Example 2. Next, we provide an example of a special function transform. This example can be further applied to accelerate the Fourier–Bessel transform that is important in many real applications. 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,

$$(26) \qquad u(x_i) = \sum_{j=1}^{N} H_{j-1}^{(1)}(x_i) g_j, \quad i = 1, 2, \ldots, N,$$

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

$$(27) \qquad x_i = N + \frac{2\pi}{3}(i-1)$$

and are bounded away from zero. It is demonstrated in [14] that (26) can be represented via $u = Kg$, where K satisfies the complementary low-rank property, $u_i = u(x_i)$, and $K_{ij} = H_{j-1}^{(1)}(x_i)$. The entries of matrix K can be calculated efficiently, and

TABLE 2

Numerical results for the FIO given in (24). N is the size of the matrix; r is the fixed rank in the low-rank approximations; T_{Factor} is the factorization time of the butterfly factorization; T_d is the running time of the direct evaluation; T_a is the application time of the butterfly factorization; T_d/T_a is the speed-up factor.

N, r	ϵ^a	T_{Factor}(min)	T_d(sec)	T_a(sec)	T_d/T_a
1024, 4	2.49e-05	2.92e-01	2.30e-01	3.01e-02	7.65e+00
4096, 4	4.69e-05	1.62e+00	2.64e+00	4.16e-02	6.35e+01
16384, 4	5.77e-05	1.22e+01	2.28e+01	1.84e-01	1.24e+02
65536, 4	6.46e-05	8.10e+01	2.16e+02	1.02e+00	2.12e+02
262144, 4	7.13e-05	4.24e+02	3.34e+03	4.75e+00	7.04e+02
1024, 6	1.57e-08	1.81e-01	1.84e-01	1.20e-02	1.54e+01
4096, 6	3.64e-08	1.55e+00	2.56e+00	6.42e-02	3.98e+01
16384, 6	6.40e-08	1.25e+01	2.43e+01	3.01e-01	8.08e+01
65536, 6	6.53e-08	9.04e+01	2.04e+02	1.77e+00	1.15e+02
262144, 6	6.85e-08	5.45e+02	3.68e+03	8.62e+00	4.27e+02
1024, 8	5.48e-12	1.83e-01	1.78e-01	1.63e-02	1.09e+01
4096, 8	1.05e-11	1.98e+00	2.71e+00	8.72e-02	3.11e+01
16384, 8	2.09e-11	1.41e+01	3.34e+01	5.28e-01	6.33e+01
65536, 8	2.62e-11	1.17e+02	2.10e+02	2.71e+00	7.75e+01
262144, 8	4.13e-11	6.50e+02	3.67e+03	1.52e+01	2.42e+02

TABLE 3

Numerical results with the matrix given by (26).

N, r	ϵ^a	T_{Factor}(min)	T_d(sec)	T_a(sec)	T_d/T_a
1024, 4	2.35e-06	8.78e-01	8.30e-01	1.06e-02	7.86e+01
4096, 4	5.66e-06	5.02e+00	5.30e+00	2.83e-02	1.87e+02
16384, 4	6.86e-06	3.04e+01	5.51e+01	1.16e-01	4.76e+02
65536, 4	7.04e-06	2.01e+02	7.59e+02	6.38e-01	1.19e+03
1024, 6	2.02e-08	4.31e-01	7.99e-01	9.69e-03	8.25e+01
4096, 6	4.47e-08	6.61e+00	5.41e+00	4.52e-02	1.20e+02
16384, 6	5.95e-08	4.19e+01	5.62e+01	1.61e-01	3.48e+02
65536, 6	7.86e-08	2.76e+02	7.60e+02	1.01e+00	7.49e+02

the construction algorithm with random sampling is applied to accelerate the evaluation of the sum (26). Table 3 summarizes the results of this example for different grid sizes N and truncation ranks r.

From Tables 2 and 3, we note that the accuracy of the butterfly factorization is well controlled by the max rank r. For a fixed rank r, the accuracy is almost independent of N. In practical applications, one can set the desired ϵ and increase the truncation rank r until the relative error reaches ϵ.

The tables for Examples 1 and 2 also provide numerical evidence for the asymptotic complexity of the proposed algorithms. The construction algorithm based on random sampling is of computational complexity $O(N^{3/2})$. When we quadruple the problem size, the running time of the construction sextuples and is better than we expect. The reason is that in the random sampling method, the computation of a middle matrix requires pseudoinverses of $r \times r$ matrices whose complexity is $O(r^3)$ with a large prefactor. Hence, when N is not large, the running time will be dominated by the $O(r^3 N)$ computation of middle matrices. The numbers also show that the application complexity of the butterfly factorization is $O(N \log N)$ with a prefactor much smaller than that of the butterfly algorithm with Chebyshev interpolation [1].

In Example 1, when the relative error is $\epsilon \approx 10^{-5}$, the butterfly factorization truncates the low-rank submatrices with rank 4, whereas the butterfly algorithm with Chebyshev interpolation uses nine Chebyshev grid points. The speed-up factors are 200 on average.

Example 3. In this example, we consider a composition of two FIOs, which is the discretization of the following operator:

$$(28) \qquad u(x) = \int_{\mathbb{R}} e^{2\pi\iota\Phi_2(x,\eta)} \int_{\mathbb{R}} e^{-2\pi\iota y\eta} \int_{\mathbb{R}} e^{2\pi\iota\Phi_1(y,\xi)} \widehat{f}(\xi) d\xi dy d\eta.$$

For simplicity, we consider the same phase function $\Phi_1 = \Phi_2 = \Phi$ as given by (23). By the discussion of Example 1 for one FIO, we know that the discrete analogue of the composition (28) can be represented as

$$u = KFKFf =: KFKg, \quad \text{with } g = Ff,$$

where F is the standard Fourier transform in matrix form, K is the same matrix as in Example 1, $u_i = u(x_i)$, and $g_j = \widehat{f}(\xi_j)$. Under mild assumptions as discussed in [7], the composition of two FIOs is an FIO. Hence, the new kernel matrix $\tilde{K} = KFK$ again satisfies the complementary low-rank property, though typically with slightly increased ranks.

Notice that it is not reasonable to compute the matrix $\tilde{K}$ directly. However, we have the fast Fourier transform (FFT) to apply F, and the butterfly factorization that we have built for K in Example 1 to apply K. Therefore, the construction algorithm with random matrix-vector multiplication is applied to factorize $\tilde{K}$.

Since the direct evaluation of each u_i takes $O(N^2)$ operations, the exact solution $\{u_i^d\}_{i\in S}$ for a selected set S is infeasible for large N. We apply the butterfly factorization of K and the FFT to evaluate $\{u_i\}_{i\in S}$ as an approximation to the exact solution $\{u_i^d\}_{i\in S}$. These approximations are compared to the results $\{u_i^a\}_{i\in S}$ that are given by applying the butterfly factorization of $\tilde{K}$. Table 4 summarizes the results of this example for different grid sizes N and truncation ranks r.

TABLE 4
Numerical results for the composition of two FIOs.

N, r	ϵ^a	T_{Factor}(min)	T_d(sec)	T_a(sec)	T_d/T_a
1024, 4	1.40e-02	3.26e-01	3.64e-01	4.74e-03	7.69e+01
4096, 4	1.96e-02	4.20e+00	6.59e+00	2.52e-02	2.62e+02
16384, 4	2.34e-02	4.65e+01	3.75e+01	1.15e-01	3.25e+02
65536, 4	2.18e-02	4.33e+02	3.73e+02	6.79e-01	5.49e+02
1024, 8	6.62e-05	3.65e-01	3.64e-01	8.25e-03	4.42e+01
4096, 8	8.67e-05	4.94e+00	6.59e+00	5.99e-02	1.10e+02
16384, 8	1.43e-04	6.23e+01	3.75e+01	3.47e-01	1.08e+02
65536, 8	1.51e-04	6.91e+02	3.73e+02	1.76e+00	2.12e+02
1024, 12	1.64e-08	4.79e-01	3.64e-01	1.48e-02	2.46e+01
4096, 12	1.05e-07	6.35e+00	6.59e+00	1.12e-01	5.88e+01
16384, 12	2.55e-07	7.58e+01	3.75e+01	7.64e-01	4.91e+01
65536, 12	2.69e-07	7.63e+02	3.73e+02	4.39e+00	8.49e+01

Table 4 shows the numerical results of the butterfly factorization of $\tilde{K}$. The accuracy improves as we increase the truncation rank r. Comparing Table 4 with Table 2, we notice that, for a fixed accuracy, the rank used in the butterfly factorization of the composition of FIOs should be larger than the rank used in a single FIO

butterfly factorization. This is expected since the composition is in general more complicated than the individual FIOs. T_{Factor} grows on average by a factor of 10 when we quadruple the problem size. This agrees with the estimated $O(N^{3/2} \log N)$ computational complexity for constructing the butterfly factorization. The column T_a shows that the empirical application time of our factorization is close to the estimated complexity $O(N \log N)$.

5. Conclusion and discussion. This paper introduces a butterfly factorization as a data-sparse approximation of complementary low-rank matrices. More precisely, it represents such an $N \times N$ dense matrix as a product of $O(\log N)$ sparse matrices. The factorization can be built efficiently if either a fast algorithm for applying the matrix and its adjoint is available or an explicit expression for the entries of the matrix is given. The butterfly factorization gives rise to highly efficient matrix-vector multiplications with $O(N \log N)$ operation and memory complexity. The butterfly factorization is also useful when an existing butterfly algorithm is repeatedly applied, because the application of the butterfly factorization is significantly faster than preexisting butterfly algorithms.

The method proposed here is a first step in computing data-sparse approximations for butterfly algorithms. As we mentioned earlier, the proposed butterfly factorization can be easily generalized to higher dimensions, which is especially relevant in imaging science. Another interesting direction is to invert a matrix via the butterfly factorization. While the numerical results of this paper include a couple of examples, it is natural to consider other important classes of transforms, such as the nonuniform Fourier transform and the Legendre functions associated with the spherical harmonic transform.

REFERENCES

[1] E. CANDÈS, L. DEMANET, AND L. YING, *A fast butterfly algorithm for the computation of Fourier integral operators*, Multiscale Model. Simul., 7 (2009), pp. 1727–1750.

[2] L. DEMANET AND L. YING, *Fast wave computation via Fourier integral operators*, Math. Comp., 81 (2012), pp. 1455–1486.

[3] B. ENGQUIST AND L. YING, *A fast directional algorithm for high frequency acoustic scattering in two dimensions*, Commun. Math. Sci., 7 (2009), pp. 327–345.

[4] W. HACKBUSCH, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[5] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[6] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[7] L. HÖRMANDER, *Fourier integral operators. I*, Acta Math., 127 (1971), pp. 79–183.

[8] Y. LI, H. YANG, AND L. YING, *A multiscale butterfly algorithm for multidimensional Fourier integral operators*, Multiscale Model. Simul., 13 (2015), pp. 614–631.

[9] E. LIBERTY, F. WOOLFE, P.-G. MARTINSSON, V. ROKHLIN, AND M. TYGERT, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.

[10] L. LIN, J. LU, AND L. YING, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.

[11] M. W. MAHONEY AND P. DRINEAS, *CUR matrix decompositions for improved data analysis*, Proc. Natl. Acad. Sci., 106 (2009), pp. 697–702.

[12] P. G. MARTINSSON, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.

[13] E. MICHIELSSEN AND A. BOAG, *A multilevel matrix decomposition algorithm for analyzing scattering from large structures*, IEEE Trans. Antennas and Propagation, 44 (1996), pp. 1086–1093.

[14] M. O'NEIL, F. WOOLFE, AND V. ROKHLIN, *An algorithm for the rapid evaluation of special function transforms*, Appl. Comput. Harmon. Anal., 28 (2010), pp. 203–226.

[15] J. POULSON, L. DEMANET, N. MAXWELL, AND L. YING, *A parallel butterfly algorithm*, SIAM J. Sci. Comput., 36 (2014), pp. C49–C65.

[16] D. S. SELJEBOTN, *Wavemoth—Fast spherical harmonic transforms by butterfly matrix compression*, Astrophys. J. Suppl. Ser., 199 (2012), 5.

[17] D. O. TRAD, T. J. ULRYCH, AND M. D. SACCHI, *Accurate interpolation with high-resolution time-variant Radon transforms*, Geophysics, 67 (2002), pp. 644–656.

[18] M. TYGERT, *Fast algorithms for spherical harmonic expansions*, III, J. Comput. Phys., 229 (2010), pp. 6181–6192.

[19] F. WOOLFE, E. LIBERTY, V. ROKHLIN, AND M. TYGERT, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.

[20] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.

[21] H. YANG AND L. YING, *A fast algorithm for multilinear operators*, Appl. Comput. Harmon. Anal., 33 (2012), pp. 148–158.

[22] B. YAZICI, L. WANG, AND K. DUMAN, *Synthetic aperture inversion with sparsity constraints*, in Proceedings of the 2011 International Conference on Electromagnetics in Advanced Applications (ICEAA), IEEE, Piscataway, NJ, 2011, pp. 1404–1407.

[23] L. YING, *Sparse Fourier transform via butterfly algorithm*, SIAM J. Sci. Comput., 31 (2009), pp. 1678–1694.