Approximate Inversion of Discrete Fourier Integral Operators via Hierarchically Semiseparable Matrices

Yingzhou Li*, Jingyu Liu†

September 11, 2026

Abstract

This paper introduces a novel method for approximating the inverse of discrete Fourier integral operators (FIOs). Given an $N \times N$ matrix representation $\boldsymbol{K}$ of an FIO, the proposed algorithm consists of two stages. In the offline stage, we first construct a butterfly factorization (BF) $\widetilde{\boldsymbol{K}}$ of $\boldsymbol{K}$, which enables fast forward matrix-vector multiplication. We then construct a hierarchically semiseparable (HSS) approximation $\widetilde{\boldsymbol{G}} \approx \boldsymbol{G}$, where $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$, using fast applications of $\widetilde{\boldsymbol{K}}$ and $\widetilde{\boldsymbol{K}}^*$ to random matrices. Finally, we apply the ULV factorization to the HSS matrix $\widetilde{\boldsymbol{G}}$ to obtain an approximation $\widetilde{\boldsymbol{F}} \approx \boldsymbol{G}^{-1}$. Combining these approximations yields an approximate inverse $\boldsymbol{K}^{-1} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*$. The offline stage has complexity $\mathscr{O}(N \log^2 N)$ for 1D problems and $\mathscr{O}(N^{1.5} \log N)$ for 2D problems. In the online stage, the proposed method approximates $\boldsymbol{K}^{-1}\boldsymbol{u}$ for a given input vector $\boldsymbol{u}$ with complexity $\mathscr{O}(N \log N)$ for both 1D and 2D problems. The proposed method can be used either as a direct solver or as a preconditioner for iterative methods. Numerical results for 1D and 2D FIOs demonstrate the effectiveness of the proposed method.

Keywords Fourier integral operator, butterfly factorization, hierarchically semiseparable matrix

1 Introduction

This paper considers the d-dimensional *discrete Fourier integral operator* (FIO) of the form

$$u(\boldsymbol{x}) = \sum_{\boldsymbol{\xi} \in \Xi} a(\boldsymbol{x}, \boldsymbol{\xi}) \mathrm{e}^{2\pi \mathrm{i} \phi(\boldsymbol{x}, \boldsymbol{\xi})} f(\boldsymbol{\xi}), \quad \boldsymbol{x} \in X, \tag{1.1}$$

where $X = \{\boldsymbol{x_i} = (i_1/n, \cdots, i_d/n) : 0 \le i_1, \ldots, i_d < n\}$ and $\Xi = \{\boldsymbol{\xi_j} = (j_1, \cdots, j_d) : -n/2 \le j_1, \ldots, j_d < n/2\}$ for a positive even integer n. Here, $\boldsymbol{x}$ and $\boldsymbol{\xi}$ denote the *spatial* and *frequency* variables, respectively. The *amplitude function* $a(\boldsymbol{x}, \boldsymbol{\xi})$ is assumed to be smooth in both $\boldsymbol{x}$ and $\boldsymbol{\xi}$. The *phase function* $\phi(\boldsymbol{x}, \boldsymbol{\xi})$ is smooth in $(\boldsymbol{x}, \boldsymbol{\xi})$ for $\boldsymbol{\xi} \neq \boldsymbol{0}$ and is homogeneous of degree 1 in $\boldsymbol{\xi}$, that is, $\phi(\boldsymbol{x}, \lambda\boldsymbol{\xi}) = \lambda\phi(\boldsymbol{x}, \boldsymbol{\xi})$ for any $\lambda > 0$. Although the spatial and frequency grids X and Ξ are assumed to be uniform in this paper, the proposed method can be extended to general grids.

*School of Mathematical Sciences, Fudan University; Shanghai Key Laboratory for Contemporary Applied Mathematics, Fudan University, yingzhouli@fudan.edu.cn

†School of Mathematical Sciences, Fudan University, jyliu22@m.fudan.edu.cn

In matrix form, the FIO in (1.1) can be written as

$$\boldsymbol{u} = \boldsymbol{K}\boldsymbol{f}, \tag{1.2}$$

where $\boldsymbol{K} \in \mathbb{C}^{N\times N}$, with $N = n^d$, is the matrix representation of the FIO, and $\boldsymbol{u}$ and $\boldsymbol{f}$ are the vectors in $\mathbb{C}^N$ corresponding to u and f, respectively. In some applications, only the *forward* computation of the FIO is required, namely, computing $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{f}$ for a given $\boldsymbol{f}$. In this paper, we focus on the *inverse* computation, namely, recovering $\boldsymbol{f}$ from $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{f}$ for a given $\boldsymbol{u}$. Both the forward and inverse computations are fundamental in many applications. We refer the reader to [4, 6, 27] for applications of FIOs in radar imaging, seismic imaging, and wave propagation.

1.1 Related Work

Many algorithms have been proposed in the literature for the fast forward computation of FIOs. In [4], the authors proposed a fast method based on partitioning the frequency domain into wedges. The discrete FIO restricted to each wedge can be decomposed into two components. The first component is a nonuniform Fourier transform, which can be computed efficiently by the nonuniform fast Fourier transform (NUFFT) [1, 25]. The second component has a low-rank structure and can therefore be applied efficiently. Together, these two components reduce the cost of the forward computation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N^{5/4}\log N)$. Another class of algorithms is based on the *butterfly factorization* (BF) [18,19], which can be viewed as an algebraic version of the *butterfly algorithm* [5] and has complexity $\mathcal{O}(N\log N)$. Recently, in [14], a tensor butterfly factorization is proposed for multidimensional FIOs. By combining the BF with a tensor extension of the complementary low-rank property, the cost of the forward computation is further reduced to $\mathcal{O}(N)$.

Due to the availability of efficient forward and adjoint algorithms, one can solve the linear system (1.2) using Krylov methods such as the generalized minimal residual method (GMRES) [26], or by applying the conjugate gradient method (CG) [11] to the associated normal equations. However, convergence may be slow for ill-conditioned systems, making preconditioning necessary in practice. Moreover, fast forward representations of FIOs, such as the BF, generally cannot be inverted directly.

When dealing with matrices arising from the discretization of integral operators, the *hierarchical matrix* ($\mathcal{H}$-matrix) is a powerful tool for matrix compression and fast numerical linear algebra operations [2, 8–10]. The $\mathcal{H}$-matrix is a data-sparse representation of a dense matrix, based on a hierarchical partitioning of the matrix and low-rank approximations of certain submatrices. Related methods include the *hierarchical interpolative factorization* (HIF) [12, 13, 20] and the *hierarchically semiseparable* (HSS) matrix [24, 29]. These methods provide efficient ways to perform both forward applications and inverse operations in linear or quasi-linear time. Unfortunately, FIO matrices typically do not satisfy the low-rank admissibility conditions required by these methods.

In [7], the authors proposed a method for approximating the inverse of FIOs based on the identity $\boldsymbol{K}^{-1} = (\boldsymbol{K}^*\boldsymbol{K})^{-1}\boldsymbol{K}^*$. The matrix $\boldsymbol{K}$ is approximated by a BF $\widetilde{\boldsymbol{K}}$, while the matrix $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$ is approximated by an $\mathcal{H}$-matrix $\widetilde{\boldsymbol{G}}$. This compression is achieved by a *black-box* peeling algorithm [21], using fast applications of $\widetilde{\boldsymbol{K}}$ and $\widetilde{\boldsymbol{K}}^*$. The $\mathcal{H}$-matrix $\widetilde{\boldsymbol{G}}$ is then inverted by an HIF-based algorithm, yielding an approximation $\widetilde{\boldsymbol{F}} \approx \boldsymbol{G}^{-1}$. Finally, the product $\widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*$ is used as an approximation of $\boldsymbol{K}^{-1}$.

1.2 Contributions

The main contribution of this paper is a novel method for the approximate inversion of FIOs. Instead of using an $\mathcal{H}$-matrix to approximate $\boldsymbol{G}$ and an HIF-based algorithm for the inversion, we use an HSS matrix together with its ULV factorization [29] to achieve the same goal. Specifically, the proposed algorithm consists of two stages. The offline stage consists of three steps. First, a butterfly factorization $\widetilde{\boldsymbol{K}}$ of $\boldsymbol{K}$ is constructed using the algorithm in [18]. Second, an HSS approximation $\widetilde{\boldsymbol{G}}$ of $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$ is constructed by a randomized black-box algorithm. This algorithm modifies the method in [15] by using independent test matrices at different levels, allowing the sampling to be adapted to the numerical ranks at each level.

Third, the ULV factorization of $\widetilde{\boldsymbol{G}}$ is computed following the algorithm in [29], yielding an approximation $\widetilde{\boldsymbol{F}} \approx \boldsymbol{G}^{-1}$. Combining these approximations gives an approximation of the inverse of $\boldsymbol{K}$: $\boldsymbol{K}^{-1} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*$. The offline stage has complexity $\mathscr{O}(N \log^2 N)$ for 1D problems and $\mathscr{O}(N^{1.5} \log N)$ for 2D problems. In the online stage, for a given input vector $\boldsymbol{u}$, the approximate solution of $\boldsymbol{f} = \boldsymbol{K}^{-1}\boldsymbol{u}$ is computed as $\boldsymbol{f} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*\boldsymbol{u}$. The online stage has complexity $\mathscr{O}(N \log N)$ for both 1D and 2D problems. The proposed method can be used either as a direct solver or as a preconditioner for iterative methods. We tested the proposed method on several examples of 1D and 2D FIOs, including constant-amplitude and variable-amplitude FIOs from the generalized Radon transform [4]. The numerical results demonstrate the effectiveness and efficiency of the proposed method.

1.3 Organization

The rest of the paper is organized as follows. Section 2 reviews the butterfly factorization, the HSS matrix, and the ULV factorization, which are the main tools used in the proposed algorithm. Section 3 presents a new algorithm for the black-box construction of HSS matrices. Section 4 describes the proposed method for the approximate inversion of FIOs via HSS matrices. Section 5 reports numerical results for the proposed method. Finally, Section 6 concludes the paper.

2 Preliminaries

2.1 Notations

For a set $\mathcal{J}$, we use $|\mathcal{J}|$ to denote its cardinality. We use MATLAB notation to denote submatrices: $\boldsymbol{A}(\mathcal{R}, \mathcal{C})$ denotes the submatrix of $\boldsymbol{A}$ with row indices in $\mathcal{R}$ and column indices in $\mathcal{C}$. A colon in an index denotes all indices in that dimension; for example, $\boldsymbol{A}(:, \mathcal{C})$ denotes the submatrix of $\boldsymbol{A}$ with all rows and column indices in $\mathcal{C}$. For a matrix with a superscript $\diamond$, we write $\boldsymbol{A}^{\diamond,*}$ to denote $(\boldsymbol{A}^{\diamond})^*$.

Tree structures are used in both the BF and the HSS matrix. For a node τ in a tree T, we use $\mathrm{ch}(\tau)$ to denote the set of child nodes of τ. The level of a node in T is defined recursively as follows. For the root node ρ, $\mathrm{level}(\rho) = 0$. If τ is a nonleaf node and $\alpha \in \mathrm{ch}(\tau)$, then $\mathrm{level}(\alpha) = \mathrm{level}(\tau) + 1$. The maximum level of the tree is denoted by L.

2.2 Butterfly Factorization

In this section, we briefly review the butterfly factorization (BF) for FIOs [18, 19]. Let

$$k(\boldsymbol{x}, \boldsymbol{\xi}) = a(\boldsymbol{x}, \boldsymbol{\xi}) \mathrm{e}^{2\pi \imath \phi(\boldsymbol{x}, \boldsymbol{\xi})}$$

be the kernel associated with $\boldsymbol{K}$. The spatial and frequency domains are recursively partitioned into trees T_X and T_Ξ of maximum level $L = \mathscr{O}(\log N)$. We assume that L is even. For nodes $\tau \in \mathsf{T}_X$ and $\sigma \in \mathsf{T}_\Xi$, let $\mathcal{I}_\tau$ and $\mathcal{J}_\sigma$ denote their corresponding index sets. Define the corresponding local grids by $X_\tau = \{\boldsymbol{x}_{\boldsymbol{i}} : \boldsymbol{i} \in \mathcal{I}_\tau\}$ and $\Xi_\sigma = \{\boldsymbol{\xi}_{\boldsymbol{j}} : \boldsymbol{j} \in \mathcal{J}_\sigma\}$. The complementary low-rank property [19] states that $\boldsymbol{K}(\mathcal{I}_\tau, \mathcal{J}_\sigma)$ is numerically low-rank whenever $\mathrm{level}(\tau) = \ell$ and $\mathrm{level}(\sigma) = L - \ell$. This property leads to the approximation

$$\boldsymbol{K} \approx \widetilde{\boldsymbol{K}} = \boldsymbol{U}^{[L]} \boldsymbol{B}^{[L]} \cdots \boldsymbol{B}^{[h+1]} \boldsymbol{M}^{[h]} \boldsymbol{C}^{[h+1]} \cdots \boldsymbol{C}^{[L]} \boldsymbol{V}^{[L]}, \tag{2.1}$$

where $h = L/2$ is the middle level of the trees.

The factors in (2.1) can be constructed using Chebyshev interpolation [18]. Let r be the interpolation rank. For each spatial box τ, let $\boldsymbol{y}_\tau$, $Z_\tau = \{\boldsymbol{z}_{\tau;i}\}_{i=1}^r$ and $\{p_{\tau;i}\}$ be its center, the Chebyshev points, and the associated Lagrange functions. Define the center $\boldsymbol{\eta}_\sigma$, the Chebyshev points $\Gamma_\sigma = \{\boldsymbol{\gamma}_{\sigma;j}\}_{j=1}^r$, and the Lagrange functions $\{q_{\sigma;j}\}$ analogously for each frequency box.

For $h \le \ell \le L$, the phase-modulated interpolation functions are defined by

$$u^{[\ell]}_{\tau,\sigma;i}(\boldsymbol{x}) = \mathrm{e}^{2\pi\imath(\phi(\boldsymbol{x},\boldsymbol{\eta}_\sigma)-\phi(\boldsymbol{z}_{\tau;i},\boldsymbol{\eta}_\sigma))} p_{\tau;i}(\boldsymbol{x}) \quad \text{and} \quad v^{[\ell]}_{\tau,\sigma;j}(\boldsymbol{\xi}) = \mathrm{e}^{2\pi\imath(\phi(\boldsymbol{y}_\tau,\boldsymbol{\xi})-\phi(\boldsymbol{y}_\tau,\boldsymbol{\gamma}_{\sigma;j}))} q_{\sigma;j}(\boldsymbol{\xi}).$$

For the first definition, level$(\tau) = \ell$ and level$(\sigma) = L-\ell$; for the second definition, the spatial and frequency levels are $L-\ell$ and ℓ, respectively.

At the middle level, the phase can be decomposed as

$$\phi(\boldsymbol{x},\boldsymbol{\xi}) = \phi(\boldsymbol{x},\boldsymbol{\eta}_\sigma) + \phi(\boldsymbol{y}_\tau,\boldsymbol{\xi}) + \delta_{\tau,\sigma}(\boldsymbol{x},\boldsymbol{\xi}),$$

where $\delta_{\tau,\sigma}(\boldsymbol{x},\boldsymbol{\xi}) = \phi(\boldsymbol{x},\boldsymbol{\xi}) - \phi(\boldsymbol{x},\boldsymbol{\eta}_\sigma) - \phi(\boldsymbol{y}_\tau,\boldsymbol{\xi})$. The function $\mathrm{e}^{2\pi\imath\delta_{\tau,\sigma}(\boldsymbol{x},\boldsymbol{\xi})}$ can be accurately approximated by Chebyshev interpolation [5]. Together with the smoothness of the amplitude, this gives

$$k(\boldsymbol{x},\boldsymbol{\xi}) \approx \sum_{i=1}^{r}\sum_{j=1}^{r} u^{[h]}_{\tau,\sigma;i}(\boldsymbol{x}) k(\boldsymbol{z}_{\tau;i},\boldsymbol{\gamma}_{\sigma;j}) v^{[h]}_{\tau,\sigma;j}(\boldsymbol{\xi}), \quad \boldsymbol{x} \in X_\tau,\ \boldsymbol{\xi} \in \Xi_\sigma. \tag{2.2}$$

Let $\boldsymbol{U}^{[h]}_{\tau,\sigma}$ and $\boldsymbol{V}^{[h]}_{\tau,\sigma}$ be obtained by evaluating the corresponding left and right interpolation functions on X_τ and Ξ_σ, respectively. Then

$$\boldsymbol{K}(\mathcal{I}_\tau,\mathcal{J}_\sigma) = k(X_\tau,\Xi_\sigma) \approx \boldsymbol{U}^{[h]}_{\tau,\sigma}\boldsymbol{M}^{[h]}_{\tau,\sigma}\boldsymbol{V}^{[h]}_{\tau,\sigma},$$

where $\boldsymbol{M}^{[h]}_{\tau,\sigma} = k(Z_\tau,\Gamma_\sigma)$. Assembling these local approximations gives

$$\boldsymbol{K} \approx \boldsymbol{U}^{[h]}\boldsymbol{M}^{[h]}\boldsymbol{V}^{[h]}. \tag{2.3}$$

The diagonal blocks of $\boldsymbol{U}^{[h]}$ are formed by horizontally concatenating the local left interpolation matrices over frequency nodes, while those of $\boldsymbol{V}^{[h]}$ are formed by vertically concatenating the local right interpolation matrices over spatial nodes. The nonzero blocks of the weighted permutation matrix $\boldsymbol{M}^{[h]}$ are $k(Z_\tau,\Gamma_\sigma)$.

The same interpolation is used to construct the transfer matrices. For $\ell = h,\dots,L-1$, let τ be a spatial node at level $\ell+1$ with parent α, and let σ be a frequency node at level $L-\ell-1$ with children $\beta_1,\dots,\beta_m$. For $\boldsymbol{x} \in X_\tau$, define

$$u^{[\ell]}_{\tau,\sigma}(\boldsymbol{x}) = \sum_{\beta\in\mathrm{ch}(\sigma)}\sum_{t=1}^{r} u^{[\ell]}_{\alpha,\beta;t}(\boldsymbol{x}) = \sum_{\beta\in\mathrm{ch}(\sigma)}\sum_{t=1}^{r} \mathrm{e}^{2\pi\imath\left(\phi(\boldsymbol{x},\boldsymbol{\eta}_\beta)-\phi(\boldsymbol{z}_{\alpha;t},\boldsymbol{\eta}_\beta)\right)} p_{\alpha;t}(\boldsymbol{x}). \tag{2.4}$$

Applying the phase-modulated interpolation associated with the new complementary pair (τ,σ) gives

$$u^{[\ell]}_{\tau,\sigma}(\boldsymbol{x}) \approx \sum_{i=1}^{r} u^{[\ell+1]}_{\tau,\sigma;i}(\boldsymbol{x}) b^{[\ell+1]}_{\tau,\sigma;i}, \tag{2.5}$$

where $b^{[\ell+1]}_{\tau,\sigma;i} = \sum_{\beta\in\mathrm{ch}(\sigma)}\sum_{t=1}^{r} u^{[\ell]}_{\alpha,\beta;t}(\boldsymbol{z}_{\tau;i})$. Let $\boldsymbol{U}^{[\ell]}_{\tau,\beta}$ be the corresponding blocks in $\boldsymbol{U}^{[\ell]}_{\alpha,\beta}$, and let $\boldsymbol{U}^{[\ell+1]}_{\tau,\sigma}$ be the interpolation matrix corresponding to $\{u^{[\ell+1]}_{\tau,\sigma;i}(\boldsymbol{x})\}$. Applying the same interpolation separately to each summand in (2.4) yields

$$\boldsymbol{U}^{[\ell]}_{\tau,\sigma} = \begin{bmatrix} \boldsymbol{U}^{[\ell]}_{\tau,\beta_1} & \cdots & \boldsymbol{U}^{[\ell]}_{\tau,\beta_m} \end{bmatrix} \approx \boldsymbol{U}^{[\ell+1]}_{\tau,\sigma}\boldsymbol{B}^{[\ell+1]}_{\tau,\sigma},$$

where $\boldsymbol{B}^{[\ell+1]}_{\tau,\sigma}$ collects the interpolation coefficients obtained by evaluating $u^{[\ell]}_{\alpha,\beta;t}$ at the new Chebyshev points $\boldsymbol{z}_{\tau;i}$, for $\beta \in \mathrm{ch}(\sigma)$ and $1 \le t,i \le r$. The right transfer matrices are obtained similarly by exchanging the roles of the spatial and frequency variables. Assembling the local transfer matrices gives

$$\boldsymbol{U}^{[\ell]} \approx \boldsymbol{U}^{[\ell+1]}\boldsymbol{B}^{[\ell+1]}, \quad \boldsymbol{V}^{[\ell]} \approx \boldsymbol{C}^{[\ell+1]}\boldsymbol{V}^{[\ell+1]}, \quad \ell = h,\dots,L-1. \tag{2.6}$$

Substituting (2.6) into (2.3) yields (2.1).

Both the construction and application of the BF have complexity $\mathcal{O}(N\log N)$. The adjoint has the same application cost and is obtained by reversing the order of the factors in (2.1) and taking their conjugate transposes.

2.3 HSS Matrix

We review the definition of hierarchically semiseparable (HSS) matrices in this section and their related algorithms in the next section. Since the HSS matrix in this paper is used to approximate $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$, which is Hermitian positive definite (HPD), we focus on HPD HSS matrices.

Definition 2.1 (HSS tree, [29]). *A tree T is called an HSS tree with index set $\mathcal{J}$ if each node τ of T is associated with an index set $\mathcal{J}_\tau$ satisfying the following conditions:*

(1) $\mathcal{J}_\rho = \mathcal{J}$ for the root node ρ.

(2) For a nonleaf node τ, we have $\mathcal{J}_\tau = \sqcup_{\alpha\in\mathrm{ch}(\tau)}\mathcal{J}_\alpha$, where $\sqcup$ denotes the disjoint union.

We use $N = |\mathcal{J}|$ and $N_\tau = |\mathcal{J}_\tau|$ to denote the sizes of the index sets $\mathcal{J}$ and $\mathcal{J}_\tau$, respectively.

For simplicity, throughout this paper, we assume that the HSS tree is a full binary tree; that is, each nonleaf node has exactly two children, and all leaf nodes are at the same level L. This assumption is natural for 1D problems. For 2D problems, the HSS tree is typically a quadtree, and all algorithms and results can be extended to this setting in a straightforward manner.

Definition 2.2 (HSS matrix, [29]). *Let T be an HSS tree with index set $\mathcal{J}$. A matrix $\boldsymbol{H} \in \mathbb{C}^{|\mathcal{J}|\times|\mathcal{J}|}$ is called an HSS matrix associated with T if there exist matrices $\boldsymbol{D}_\tau$, $\boldsymbol{U}_\tau$, and $\boldsymbol{B}_{\tau,\sigma}$, called HSS generators, associated with the nodes of T and satisfying the following conditions:*

(1) For a leaf node τ, $\boldsymbol{D}_\tau = \boldsymbol{H}(\mathcal{J}_\tau,\mathcal{J}_\tau) \in \mathbb{C}^{N_\tau\times N_\tau}$ is a dense Hermitian matrix, and $\boldsymbol{U}_\tau^{\mathrm{big}} = \boldsymbol{U}_\tau \in \mathbb{C}^{N_\tau\times r_\tau}$ is a dense matrix.

(2) For a nonleaf node τ with children α_1 and α_2,

$$\boldsymbol{D}_\tau = \boldsymbol{H}(\mathcal{J}_\tau,\mathcal{J}_\tau) = \begin{bmatrix} \boldsymbol{D}_{\alpha_1} & \boldsymbol{U}_{\alpha_1}^{\mathrm{big}}\boldsymbol{B}_{\alpha_2,\alpha_1}^*\boldsymbol{U}_{\alpha_2}^{\mathrm{big},*} \\ \boldsymbol{U}_{\alpha_2}^{\mathrm{big}}\boldsymbol{B}_{\alpha_2,\alpha_1}\boldsymbol{U}_{\alpha_1}^{\mathrm{big},*} & \boldsymbol{D}_{\alpha_2} \end{bmatrix}.$$

For a nonleaf, nonroot node τ with children α_1 and α_2, the matrix $\boldsymbol{U}_\tau^{\mathrm{big}}$ satisfies the recursion

$$\boldsymbol{U}_\tau^{\mathrm{big}} = \begin{bmatrix} \boldsymbol{U}_{\alpha_1}^{\mathrm{big}} & \\ & \boldsymbol{U}_{\alpha_2}^{\mathrm{big}} \end{bmatrix} \boldsymbol{U}_\tau \in \mathbb{C}^{N_\tau\times r_\tau},$$

where $\boldsymbol{U}_\tau$ is a dense matrix of size $(r_{\alpha_1}+r_{\alpha_2})\times r_\tau$.

By definition, $\boldsymbol{H}$ is a Hermitian matrix. Furthermore, for any nonroot node τ, the matrix $\boldsymbol{U}_\tau^{\mathrm{big}}$ forms a basis for the off-diagonal block $\boldsymbol{H}(\mathcal{J}_\tau,\mathcal{J}_\tau^{\mathrm{c}})$, where $\mathcal{J}_\tau^{\mathrm{c}} = \mathcal{J}\backslash\mathcal{J}_\tau$. This is often referred to as the *shared basis property*, and $\boldsymbol{H}(\mathcal{J}_\tau,\mathcal{J}_\tau^{\mathrm{c}})$ is called the *HSS block* associated with τ. The *HSS rank* of $\boldsymbol{H}$ is defined as the maximum rank among all HSS blocks, that is, $r = \max_{\tau\neq\rho} r_\tau$. When the context is clear, we omit the index sets and use node labels directly to denote the corresponding blocks. For example, $\boldsymbol{H}_{\tau,\tau^{\mathrm{c}}} = \boldsymbol{H}(\mathcal{J}_\tau,\mathcal{J}_\tau^{\mathrm{c}})$.

The hierarchical structure of an HSS matrix is often represented in an alternative form known as the *telescoping factorization* [23]. Specifically, define the block diagonal matrices

$$\begin{aligned} \boldsymbol{U}^{[\ell]} &= \mathrm{diag}\big(\boldsymbol{U}_\tau : \mathrm{level}(\tau) = \ell\big), \quad 1 \le \ell \le L, \\ \boldsymbol{B}^{[\ell]} &= \mathrm{diag}\big(\boldsymbol{B}_\tau : \mathrm{level}(\tau) = \ell\big), \quad \boldsymbol{B}_\tau = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{B}_{\alpha_2,\alpha_1}^* \\ \boldsymbol{B}_{\alpha_2,\alpha_1} & \boldsymbol{0} \end{bmatrix}, \quad 0 \le \ell \le L-1, \\ \boldsymbol{D}^{[L]} &= \mathrm{diag}\big(\boldsymbol{D}_\tau : \mathrm{level}(\tau) = L\big). \end{aligned} \tag{2.7}$$

Then the HSS matrix $\boldsymbol{H}$ admits the recursive factorization

$$\begin{aligned} \boldsymbol{H}^{[L]} &= \boldsymbol{U}^{[L]}\boldsymbol{H}^{[L-1]}\boldsymbol{U}^{[L],*} + \boldsymbol{D}^{[L]}, \\ \boldsymbol{H}^{[\ell]} &= \boldsymbol{U}^{[\ell]}\boldsymbol{H}^{[\ell-1]}\boldsymbol{U}^{[\ell],*} + \boldsymbol{B}^{[\ell]}, \quad 1 \le \ell \le L-1, \\ \boldsymbol{H}^{[0]} &= \boldsymbol{B}^{[0]}, \end{aligned} \tag{2.8}$$

where $\boldsymbol{H}^{[L]} = \boldsymbol{H}$.

2.4 ULV Factorization of the HSS Matrix

Given the HSS representation of an HPD matrix, the HSS matrix can be inverted efficiently by the ULV factorization [29]. The ULV factorization can be viewed as a generalization of the Cholesky factorization. Using the notation in Definition 2.2, the ULV factorization is performed in a bottom-up manner. For a leaf node τ, suppose that $\boldsymbol{D}_\tau \in \mathbb{C}^{N_\tau \times N_\tau}$ and $\boldsymbol{U}_\tau \in \mathbb{C}^{N_\tau \times r_\tau}$. The first step, called *basis elimination*, is to compute a unitary matrix $\boldsymbol{Q}_\tau \in \mathbb{C}^{N_\tau \times N_\tau}$ such that

$$\boldsymbol{U}_\tau = \boldsymbol{Q}_\tau \begin{bmatrix} \boldsymbol{0} \\ \widehat{\boldsymbol{U}}_{\tau;2} \end{bmatrix}, \tag{2.9}$$

where $\widehat{\boldsymbol{U}}_{\tau;2} \in \mathbb{C}^{r_\tau \times r_\tau}$ is a dense matrix. This can be achieved by applying the QL factorization to $\boldsymbol{U}_\tau$. The diagonal block $\boldsymbol{D}_\tau$ is then transformed as $\check{\boldsymbol{D}}_\tau = \boldsymbol{Q}_\tau^* \boldsymbol{D}_\tau \boldsymbol{Q}_\tau$.

Partition $\check{\boldsymbol{D}}_\tau$ conformally with the partition of $\boldsymbol{U}_\tau$ in (2.9) as

$$\check{\boldsymbol{D}}_\tau = \begin{bmatrix} \check{\boldsymbol{D}}_{\tau;1,1} & \check{\boldsymbol{D}}_{\tau;2,1}^* \\ \check{\boldsymbol{D}}_{\tau;2,1} & \check{\boldsymbol{D}}_{\tau;2,2} \end{bmatrix}. \tag{2.10}$$

A *partial factorization* is then performed to eliminate the $(1,1)$ block of $\check{\boldsymbol{D}}_\tau$. More precisely, the Cholesky factorization of $\check{\boldsymbol{D}}_{\tau;1,1}$ is computed as $\check{\boldsymbol{D}}_{\tau;1,1} = \boldsymbol{L}_{\tau;1,1} \boldsymbol{L}_{\tau;1,1}^*$. Then (2.10) can be decomposed as

$$\check{\boldsymbol{D}}_\tau = \begin{bmatrix} \boldsymbol{L}_{\tau;1,1} & \\ \widehat{\boldsymbol{D}}_{\tau;2,1} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \\ & \widehat{\boldsymbol{D}}_{\tau;2,2} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{\tau;1,1} & \\ \widehat{\boldsymbol{D}}_{\tau;2,1} & \boldsymbol{I} \end{bmatrix}^*, \tag{2.11}$$

where $\widehat{\boldsymbol{D}}_{\tau;2,1} = \check{\boldsymbol{D}}_{\tau;2,1} \boldsymbol{L}_{\tau;1,1}^{-*}$ and $\widehat{\boldsymbol{D}}_{\tau;2,2} = \check{\boldsymbol{D}}_{\tau;2,2} - \widehat{\boldsymbol{D}}_{\tau;2,1} \widehat{\boldsymbol{D}}_{\tau;2,1}^*$ is the Schur complement.

After basis elimination and partial factorization have been performed for all leaf nodes, the algorithm proceeds to level $L-1$. For a nonleaf node τ at level $L-1$ with children α_1 and α_2, a *merge* step is performed to update the HSS generators of τ as

$$\boldsymbol{D}_\tau^+ = \begin{bmatrix} \widehat{\boldsymbol{D}}_{\alpha_1;2,2} & \widehat{\boldsymbol{U}}_{\alpha_1;2} \boldsymbol{B}_{\alpha_2,\alpha_1}^* \widehat{\boldsymbol{U}}_{\alpha_2;2}^* \\ \widehat{\boldsymbol{U}}_{\alpha_2;2} \boldsymbol{B}_{\alpha_2,\alpha_1} \widehat{\boldsymbol{U}}_{\alpha_1;2}^* & \widehat{\boldsymbol{D}}_{\alpha_2;2,2} \end{bmatrix}, \quad \boldsymbol{U}_\tau^+ = \begin{bmatrix} \widehat{\boldsymbol{U}}_{\alpha_1;2} & \\ & \widehat{\boldsymbol{U}}_{\alpha_2;2} \end{bmatrix} \boldsymbol{U}_\tau. \tag{2.12}$$

The children α_1 and α_2 can then be "removed", and the node τ can be treated as a leaf node with the updated HSS generators $\boldsymbol{D}_\tau^+$ and $\boldsymbol{U}_\tau^+$. These steps are repeated from bottom to top, with $\boldsymbol{D}_\tau$ and $\boldsymbol{U}_\tau$ replaced by $\boldsymbol{D}_\tau^+$ and $\boldsymbol{U}_\tau^+$, until the root node ρ is reached. At the root node, the Cholesky factorization $\boldsymbol{D}_\rho^+ = \boldsymbol{L}_\rho \boldsymbol{L}_\rho^*$ is computed directly.

Once the ULV factorization has been computed, the linear system $\boldsymbol{H}\boldsymbol{c} = \boldsymbol{y}$ can be solved efficiently by a bottom-up pass followed by a top-down pass. First, the vector $\boldsymbol{y}$ is partitioned and assigned to the leaf nodes of the HSS tree. Starting from each leaf node τ, the vector $\boldsymbol{y}_\tau$ is updated by

$$\widehat{\boldsymbol{y}}_\tau = \boldsymbol{Q}_\tau^* \boldsymbol{y}_\tau = \begin{bmatrix} \widehat{\boldsymbol{y}}_{\tau;1} \\ \widehat{\boldsymbol{y}}_{\tau;2} \end{bmatrix}.$$

Then the vector $\widehat{\boldsymbol{b}}_{\tau;1}$ is computed by solving $\boldsymbol{L}_{\tau;1,1} \widehat{\boldsymbol{b}}_{\tau;1} = \widehat{\boldsymbol{y}}_{\tau;1}$, and the vector $\widehat{\boldsymbol{y}}_{\tau;2}$ is updated as $\widehat{\boldsymbol{y}}_{\tau;2} \leftarrow \widehat{\boldsymbol{y}}_{\tau;2} - \widehat{\boldsymbol{D}}_{\tau;2,1} \widehat{\boldsymbol{b}}_{\tau;1}$. Next, for a nonleaf node τ at level $L-1$ with children α_1 and α_2, the vector $\boldsymbol{y}_\tau^+$ is obtained by merging $\widehat{\boldsymbol{y}}_{\alpha_1;2}$ and $\widehat{\boldsymbol{y}}_{\alpha_2;2}$ as

$$\boldsymbol{y}_\tau^+ = \begin{bmatrix} \widehat{\boldsymbol{y}}_{\alpha_1;2} \\ \widehat{\boldsymbol{y}}_{\alpha_2;2} \end{bmatrix}.$$

This process is repeated from bottom to top, with $\boldsymbol{y}_\tau$ replaced by $\boldsymbol{y}_\tau^+$, until the root node ρ is reached. At the root node, the vectors $\boldsymbol{b}_\rho$ and $\boldsymbol{c}_\rho$ are obtained by solving the triangular systems $\boldsymbol{L}_\rho \boldsymbol{b}_\rho = \boldsymbol{y}_\rho^+$ and $\boldsymbol{L}_\rho^* \boldsymbol{c}_\rho = \boldsymbol{b}_\rho$, respectively.

A top-down pass is then performed to compute the solution $\boldsymbol{c}$ from the root node to the leaf nodes. Starting from the root node ρ with children α_1 and α_2, the vectors $\widehat{\boldsymbol{c}}_{\alpha_1;2}$ and $\widehat{\boldsymbol{c}}_{\alpha_2;2}$ are obtained from the partition

$$\boldsymbol{c}_\rho = \begin{bmatrix} \widehat{\boldsymbol{c}}_{\alpha_1;2} \\ \widehat{\boldsymbol{c}}_{\alpha_2;2} \end{bmatrix}.$$

Then, for $1 \leq \ell \leq L$ and every node τ at level ℓ, the vector $\widehat{\boldsymbol{c}}_{\tau;1}$ is obtained by solving

$$\boldsymbol{L}^*_{\tau;1,1}\widehat{\boldsymbol{c}}_{\tau;1} = \widehat{\boldsymbol{b}}_{\tau;1} - \widehat{\boldsymbol{D}}^*_{\tau;2,1}\widehat{\boldsymbol{c}}_{\tau;2},$$

and the vector $\boldsymbol{c}_\tau$ is given by

$$\boldsymbol{c}_\tau = \boldsymbol{Q}_\tau \begin{bmatrix} \widehat{\boldsymbol{c}}_{\tau;1} \\ \widehat{\boldsymbol{c}}_{\tau;2} \end{bmatrix}.$$

If τ is a nonleaf node with children α_1 and α_2, then $\widehat{\boldsymbol{c}}_{\alpha_1;2}$ and $\widehat{\boldsymbol{c}}_{\alpha_2;2}$ are obtained from the partition of $\boldsymbol{c}_\tau$. Finally, the solution $\boldsymbol{c}$ is obtained by collecting the vectors $\boldsymbol{c}_\tau$ over all leaf nodes τ.

3 A Black-box Construction of the HSS Matrix

In this section, we present a new algorithm for the black-box construction of a Hermitian HSS matrix. The algorithm is based on the method proposed in [15]. Using the notation in Definition 2.2, the HSS matrix $\boldsymbol{H}$ is constructed by sequentially applying $\boldsymbol{H}$ to random matrices. Unlike the method in [15], the proposed algorithm uses independent random matrices at each level, allowing the sampling to be adapted to the numerical ranks at that level. The construction consists of two steps. First, a bottom-up process is performed to compute all basis matrices $\boldsymbol{U}_\tau$ and residual blocks $\breve{\boldsymbol{D}}_\tau$. Next, a top-down process is used to complete the construction by assigning the matrices $\boldsymbol{B}_{\alpha_2,\alpha_1}$ and $\boldsymbol{D}_\tau$. In this section, we assume that the rank of the HSS blocks at level ℓ is r_ℓ, with $r_0 = 0$, and that all leaf nodes have size N_L. We define $m_L = N_L$ and $m_\ell = 2r_{\ell+1}$ for $0 \leq \ell \leq L-1$, and refer to m_ℓ as the "effective leaf size" at level ℓ.

3.1 Construction on the Leaf Nodes

Starting from the leaf level L, let $s_L = r_L + m_L + p$, where p is an oversampling parameter, e.g., $p = 5$. Let $\boldsymbol{\Omega}^{[L]} \in \mathbb{C}^{|\mathcal{J}| \times s_L}$ be a random matrix with i.i.d. Gaussian entries, and let $\boldsymbol{Y}^{[L]} = \boldsymbol{H}\boldsymbol{\Omega}^{[L]}$. We call $\boldsymbol{\Omega}^{[L]}$ and $\boldsymbol{Y}^{[L]}$ the *test matrix* and *sample matrix* at level L, respectively. For each leaf node τ at level L, by considering the row block corresponding to τ in the equation $\boldsymbol{Y}^{[L]} = \boldsymbol{H}\boldsymbol{\Omega}^{[L]}$, we obtain

$$\boldsymbol{Y}^{[L]}_\tau = \boldsymbol{D}_\tau \boldsymbol{\Omega}^{[L]}_\tau + \boldsymbol{H}_{\tau,\tau^{\mathrm{c}}} \boldsymbol{\Omega}^{[L]}_{\tau^{\mathrm{c}}}. \tag{3.1}$$

Let $\boldsymbol{\Gamma}^{[L]}_\tau \in \mathbb{C}^{s_L \times (r_L+p)}$ be an orthonormal basis for the null space of $\boldsymbol{\Omega}^{[L]}_\tau$, so that $\boldsymbol{\Omega}^{[L]}_\tau \boldsymbol{\Gamma}^{[L]}_\tau = \boldsymbol{0}$. Such a matrix can be computed by performing a complete QR factorization with column pivoting (QRCP) on $\boldsymbol{\Omega}^{[L],*}_\tau$ and taking the last $r_L + p$ columns of the Q-factor. Multiplying both sides of (3.1) by $\boldsymbol{\Gamma}^{[L]}_\tau$ gives

$$\boldsymbol{Y}^{[L]}_\tau \boldsymbol{\Gamma}^{[L]}_\tau = \boldsymbol{H}_{\tau,\tau^{\mathrm{c}}} \boldsymbol{\Omega}^{[L]}_{\tau^{\mathrm{c}}} \boldsymbol{\Gamma}^{[L]}_\tau.$$

Since $\boldsymbol{\Gamma}^{[L]}_\tau$ is orthonormal, the matrix $\boldsymbol{Y}^{[L]}_\tau \boldsymbol{\Gamma}^{[L]}_\tau$ can be viewed as a sample matrix for $\boldsymbol{H}_{\tau,\tau^{\mathrm{c}}}$ with test matrix $\boldsymbol{\Omega}^{[L]}_{\tau^{\mathrm{c}}} \boldsymbol{\Gamma}^{[L]}_\tau$. Consequently, $\boldsymbol{U}_\tau$ can be constructed by computing an orthonormal basis for the column space of $\boldsymbol{Y}^{[L]}_\tau \boldsymbol{\Gamma}^{[L]}_\tau$, for example, by QRCP. In practice, the basis is truncated according to the prescribed accuracy.

Subsequently, the diagonal block $\boldsymbol{D}_\tau$ is decomposed into two parts:

$$\begin{aligned} \boldsymbol{D}_\tau &= \boldsymbol{U}_\tau (\boldsymbol{U}^*_\tau \boldsymbol{D}_\tau \boldsymbol{U}_\tau) \boldsymbol{U}^*_\tau + \left(\boldsymbol{D}_\tau - \boldsymbol{U}_\tau \boldsymbol{U}^*_\tau \boldsymbol{D}_\tau \boldsymbol{U}_\tau \boldsymbol{U}^*_\tau \right) \\ &= \boldsymbol{U}_\tau \widehat{\boldsymbol{D}}_\tau \boldsymbol{U}^*_\tau + \breve{\boldsymbol{D}}_\tau, \end{aligned} \tag{3.2}$$

where $\widehat{\boldsymbol{D}}_\tau = \boldsymbol{U}_\tau^* \boldsymbol{D}_\tau \boldsymbol{U}_\tau$ and $\check{\boldsymbol{D}}_\tau = \boldsymbol{D}_\tau - \boldsymbol{U}_\tau \widehat{\boldsymbol{D}}_\tau \boldsymbol{U}_\tau^*$. The first term represents the projection of $\boldsymbol{D}_\tau$ onto the column space of $\boldsymbol{U}_\tau$, while the second term is the residual. We will discuss the computation of $\widehat{\boldsymbol{D}}_\tau$ later and first focus on the computation of $\check{\boldsymbol{D}}_\tau$. By rewriting $\check{\boldsymbol{D}}_\tau$ as

$$\begin{aligned}\check{\boldsymbol{D}}_\tau &= \boldsymbol{D}_\tau - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^* \boldsymbol{D}_\tau \boldsymbol{U}_\tau \boldsymbol{U}_\tau^* \\ &= (\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{D}_\tau + \boldsymbol{U}_\tau \boldsymbol{U}_\tau^* \boldsymbol{D}_\tau (\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*),\end{aligned}$$

it suffices to compute $(\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{D}_\tau$. Multiplying both sides of (3.1) by $(\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*)$ and using the fact that $\boldsymbol{U}_\tau$ is an orthonormal basis for $\boldsymbol{H}_{\tau,\tau^c}$, we obtain

$$(\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{D}_\tau \boldsymbol{\Omega}_\tau^{[L]} = (\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{Y}_\tau^{[L]}.$$

Therefore,

$$(\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{D}_\tau = (\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{Y}_\tau^{[L]} \boldsymbol{\Omega}_\tau^{[L],\dagger}.$$

Consequently, the residual block is computed as

$$\check{\boldsymbol{D}}_\tau = (\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{Y}_\tau^{[L]} \boldsymbol{\Omega}_\tau^{[L],\dagger} + \boldsymbol{U}_\tau \boldsymbol{U}_\tau^* \big((\boldsymbol{I} - \boldsymbol{U}_\tau \boldsymbol{U}_\tau^*) \boldsymbol{Y}_\tau^{[L]} \boldsymbol{\Omega}_\tau^{[L],\dagger} \big)^*.$$

3.2 Construction on the Nonleaf Nodes

Suppose $1 \le \ell \le L-1$ and that the basis matrices for all nodes at levels $\ell+1, \ldots, L$ have been computed. Define $\boldsymbol{U}^{[\ell+1]} = \operatorname{diag}\big(\boldsymbol{U}_\tau : \operatorname{level}(\tau) = \ell+1\big)$. By extracting the basis matrices, we obtain the following recursive decomposition of $\boldsymbol{H}^{[\ell+1]}$, starting from $\boldsymbol{H}^{[L]} = \boldsymbol{H}$:

$$\boldsymbol{H}^{[\ell+1]} = \boldsymbol{U}^{[\ell+1]} \boldsymbol{H}^{[\ell]} \boldsymbol{U}^{[\ell+1],*} + \check{\boldsymbol{D}}^{[\ell+1]}, \tag{3.3}$$

where $\boldsymbol{H}^{[\ell]} = \boldsymbol{U}^{[\ell+1],*} \boldsymbol{H}^{[\ell+1]} \boldsymbol{U}^{[\ell+1]}$ and $\check{\boldsymbol{D}}^{[\ell+1]} = \operatorname{diag}\big(\check{\boldsymbol{D}}_\tau : \operatorname{level}(\tau) = \ell+1\big)$. If all nodes at level $\ell+1$ are "eliminated", then $\boldsymbol{H}^{[\ell]}$ remains an HSS matrix with maximum level ℓ, but with updated generators. Specifically, for every node τ at level ℓ with children α_1 and α_2, the node τ becomes a leaf node of the reduced HSS tree, with its diagonal generator updated as

$$\boldsymbol{D}_\tau \leftarrow \begin{bmatrix} \widehat{\boldsymbol{D}}_{\alpha_1} & \boldsymbol{B}_{\alpha_2,\alpha_1}^* \\ \boldsymbol{B}_{\alpha_2,\alpha_1} & \widehat{\boldsymbol{D}}_{\alpha_2} \end{bmatrix}.$$

Let $s_\ell = r_\ell + m_\ell + p$, and let $\boldsymbol{\Omega}^{[\ell]} \in \mathbb{C}^{2^\ell m_\ell \times s_\ell}$ be the test matrix at level ℓ. The corresponding sample matrix is defined as $\boldsymbol{Y}^{[\ell]} = \boldsymbol{H}^{[\ell]} \boldsymbol{\Omega}^{[\ell]}$. Then

$$\boldsymbol{Y}^{[\ell]} = \boldsymbol{H}^{[\ell]} \boldsymbol{\Omega}^{[\ell]} = \boldsymbol{U}^{[\ell+1],*} \boldsymbol{H}^{[\ell+1]} \boldsymbol{U}^{[\ell+1]} \boldsymbol{\Omega}^{[\ell]} = \Big(\boldsymbol{U}^{[\ell+1],*} \cdots \boldsymbol{U}^{[L],*} \Big) \boldsymbol{H} \Big(\boldsymbol{U}^{[L]} \cdots \boldsymbol{U}^{[\ell+1]} \Big) \boldsymbol{\Omega}^{[\ell]}.$$

That is, the sample matrix $\boldsymbol{Y}^{[\ell]}$ can be obtained by first applying the basis matrices from level $\ell+1$ up to level L to the test matrix $\boldsymbol{\Omega}^{[\ell]}$, then applying $\boldsymbol{H}$ to the resulting matrix, and finally applying the adjoint basis matrices from level L down to level $\ell+1$. Once $\boldsymbol{Y}^{[\ell]}$ is obtained, the basis matrices and residual matrices for all nodes at level ℓ can be computed by a procedure similar to that in Section 3.1. At the root node, we only need to solve for the updated diagonal block $\boldsymbol{H}^{[0]}$ from $\boldsymbol{Y}^{[0]} = \boldsymbol{H}^{[0]} \boldsymbol{\Omega}^{[0]}$.

3.3 Postprocessing

After computing all matrices $\boldsymbol{U}_\tau$ and $\check{\boldsymbol{D}}_\tau$, a top-down procedure is performed to construct the matrices $\boldsymbol{B}$ and $\boldsymbol{D}$, as summarized in [17]. For level $0 \le \ell \le L-1$, suppose that τ is a nonleaf node at level ℓ with

children α_1 and α_2. We define $\boldsymbol{D}_\tau^{[\ell]}$ and $\boldsymbol{U}_\tau^{[\ell+1]}$ as the submatrices of $\boldsymbol{D}^{[\ell]}$ and $\boldsymbol{U}^{[\ell+1]}$ corresponding to the node τ, respectively. The equation (3.3) corresponding to the node τ is

$$\begin{aligned}\boldsymbol{H}_\tau^{[\ell+1]} &= \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{H}_\tau^{[\ell]}\boldsymbol{U}_\tau^{[\ell+1],*} + \breve{\boldsymbol{D}}_\tau^{[\ell+1]} \\ &= \boldsymbol{U}_\tau^{[\ell+1]}\Big(\boldsymbol{U}_\tau^{[\ell]}\boldsymbol{H}_\tau^{[\ell-1],+}\boldsymbol{U}_\tau^{[\ell],*} + \boldsymbol{D}_\tau^{[\ell]}\Big)\boldsymbol{U}_\tau^{[\ell+1],*} + \breve{\boldsymbol{D}}_\tau^{[\ell+1]} \\ &= \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{U}_\tau^{[\ell]}\boldsymbol{H}_\tau^{[\ell-1],+}\boldsymbol{U}_\tau^{[\ell],*}\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{D}_\tau^{[\ell]}\boldsymbol{U}_\tau^{[\ell+1],*} + \breve{\boldsymbol{D}}_\tau^{[\ell+1]}.\end{aligned} \tag{3.4}$$

Here, $\boldsymbol{H}_\tau^{[\ell-1],+}$ is consistent with the telescoping factorization in (2.8). For the root node ρ, the matrices are initialized as $\boldsymbol{D}_\rho^{[0]} = \boldsymbol{H}^{[0]}$, $\boldsymbol{U}_\rho^{[0]} = \boldsymbol{I}$, and $\boldsymbol{H}_\rho^{[-1],+} = \boldsymbol{0}$. Compared with the telescoping factorization (2.8), the only difference is that the diagonal blocks of $\boldsymbol{D}_\tau^{[\ell]}$ are nonzero. Partition $\boldsymbol{D}_\tau^{[\ell]}$ as

$$\boldsymbol{D}_\tau^{[\ell]} = \begin{bmatrix}\boldsymbol{A}_{\tau;\alpha_1,\alpha_1} & \boldsymbol{A}_{\tau;\alpha_2,\alpha_1}^* \\ \boldsymbol{A}_{\tau;\alpha_2,\alpha_1} & \boldsymbol{A}_{\tau;\alpha_2,\alpha_2}\end{bmatrix}.$$

Then the second and third terms in (3.4) can be rewritten as

$$\begin{aligned}&\boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{D}_\tau^{[\ell]}\boldsymbol{U}_\tau^{[\ell+1],*} + \breve{\boldsymbol{D}}_\tau^{[\ell+1]} \\ &\quad= \begin{bmatrix}\boldsymbol{U}_{\alpha_1} & \\ & \boldsymbol{U}_{\alpha_2}\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_{\tau;\alpha_1,\alpha_1} & \boldsymbol{A}_{\tau;\alpha_2,\alpha_1}^* \\ \boldsymbol{A}_{\tau;\alpha_2,\alpha_1} & \boldsymbol{A}_{\tau;\alpha_2,\alpha_2}\end{bmatrix}\begin{bmatrix}\boldsymbol{U}_{\alpha_1} & \\ & \boldsymbol{U}_{\alpha_2}\end{bmatrix}^* + \begin{bmatrix}\breve{\boldsymbol{D}}_{\alpha_1} & \\ & \breve{\boldsymbol{D}}_{\alpha_2}\end{bmatrix} \\ &\quad= \begin{bmatrix}\boldsymbol{U}_{\alpha_1} & \\ & \boldsymbol{U}_{\alpha_2}\end{bmatrix}\begin{bmatrix} & \boldsymbol{B}_{\alpha_2,\alpha_1}^* \\ \boldsymbol{B}_{\alpha_2,\alpha_1} & \end{bmatrix}\begin{bmatrix}\boldsymbol{U}_{\alpha_1} & \\ & \boldsymbol{U}_{\alpha_2}\end{bmatrix}^* + \begin{bmatrix}\boldsymbol{D}_{\alpha_1} & \\ & \boldsymbol{D}_{\alpha_2}\end{bmatrix} \\ &\quad= \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{B}_\tau^{[\ell]}\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{D}_\tau^{[\ell+1]},\end{aligned}$$

where $\boldsymbol{B}_{\alpha_2,\alpha_1} = \boldsymbol{A}_{\tau;\alpha_2,\alpha_1}$ and $\boldsymbol{D}_\alpha = \breve{\boldsymbol{D}}_\alpha + \boldsymbol{U}_\alpha\boldsymbol{A}_{\tau;\alpha,\alpha}\boldsymbol{U}_\alpha^*$ for each $\alpha \in \mathrm{ch}(\tau)$. Therefore, equation (3.4) can be reformulated as

$$\begin{aligned}\boldsymbol{H}_\tau^{[\ell+1]} &= \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{U}_\tau^{[\ell]}\boldsymbol{H}_\tau^{[\ell-1],+}\boldsymbol{U}_\tau^{[\ell],*}\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{B}_\tau^{[\ell]}\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{D}_\tau^{[\ell+1]} \\ &= \boldsymbol{U}_\tau^{[\ell+1]}\Big(\boldsymbol{U}_\tau^{[\ell]}\boldsymbol{H}_\tau^{[\ell-1],+}\boldsymbol{U}_\tau^{[\ell],*} + \boldsymbol{B}_\tau^{[\ell]}\Big)\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{D}_\tau^{[\ell+1]} \\ &= \boldsymbol{U}_\tau^{[\ell+1]}\boldsymbol{H}_\tau^{[\ell],+}\boldsymbol{U}_\tau^{[\ell+1],*} + \boldsymbol{D}_\tau^{[\ell+1]},\end{aligned}$$

where $\boldsymbol{H}_\tau^{[\ell],+} = \boldsymbol{U}_\tau^{[\ell]}\boldsymbol{H}_\tau^{[\ell-1],+}\boldsymbol{U}_\tau^{[\ell],*} + \boldsymbol{B}_\tau^{[\ell]}$ is consistent with the telescoping factorization in (2.8). The process is repeated until the leaf nodes are reached. When τ is a leaf node, the desired matrix $\boldsymbol{D}_\tau$ is obtained after the update, and the process terminates. Note that the matrices $\boldsymbol{A}_{\tau;\alpha,\alpha}$ in the above discussion are precisely the matrices $\widehat{\boldsymbol{D}}_\alpha$ defined in (3.2).

4 Approximate Inversion of FIOs

4.1 An Inversion Algorithm Based on BF and HSS Approximations

We now consider the problem of inverting the discrete FIO matrix $\boldsymbol{K}$ in (1.2). Suppose that $\boldsymbol{K}$ is nonsingular. Then its inverse can be expressed as

$$\boldsymbol{K}^{-1} = \big(\boldsymbol{K}^*\boldsymbol{K}\big)^{-1}\boldsymbol{K}^*. \tag{4.1}$$

In [7], the authors showed that the matrix $\boldsymbol{K}^*\boldsymbol{K}$ can be approximated by an $\mathcal{H}$-matrix. Our numerical experiments indicate that $\boldsymbol{K}^*\boldsymbol{K}$ can be further compressed using an HSS matrix. Motivated by this observation, equation (4.1) suggests a method for approximating $\boldsymbol{K}^{-1}$. First, the matrix $\boldsymbol{K}$ is approximated

by a BF $\widetilde{\boldsymbol{K}} \approx \boldsymbol{K}$. Then, an HSS approximation $\widetilde{\boldsymbol{G}}$ of $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$ is constructed by the black-box construction algorithm in Section 3, where the BF approximation $\widetilde{\boldsymbol{K}}^*\widetilde{\boldsymbol{K}}$ is used to perform fast matrix-vector products. We assume that the BF approximation $\widetilde{\boldsymbol{K}}$ is nonsingular and that the HSS approximation $\widetilde{\boldsymbol{G}}$ is HPD. Finally, the ULV factorization of $\widetilde{\boldsymbol{G}}$ is computed, yielding an approximation $\widetilde{\boldsymbol{F}} \approx \boldsymbol{G}^{-1}$. Combining these approximations gives

$$\boldsymbol{K}^{-1} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*. \tag{4.2}$$

The algorithm is summarized in Algorithm 1.

Algorithm 1 Approximate inversion of the FIO

Input: Space and frequency grids X and Ξ, amplitude function $a(\boldsymbol{x}, \boldsymbol{\xi})$, phase function $\phi(\boldsymbol{x}, \boldsymbol{\xi})$, and accuracy parameter ε.

Output: Approximate inverse $\boldsymbol{K}^{-1} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*$.

1: Construct the BF $\widetilde{\boldsymbol{K}}$ of $\boldsymbol{K}$.
2: Construct the HSS approximation $\widetilde{\boldsymbol{G}}$ using the black-box construction algorithm in Section 3, where the BF approximation $\widetilde{\boldsymbol{K}}^*\widetilde{\boldsymbol{K}}$ is used to perform fast matrix-vector products.
3: Compute the ULV factorization $\widetilde{\boldsymbol{F}}$ of the HSS matrix $\widetilde{\boldsymbol{G}}$.

The approximation (4.2) can be used as a direct solver for the linear system (1.2). The HSS inverse factor $\widetilde{\boldsymbol{F}}$ can also serve as a preconditioner for the approximate normal equation. More specifically, the approximate normal equation associated with the linear system (1.2) is given by

$$\widetilde{\boldsymbol{K}}^*\widetilde{\boldsymbol{K}}\boldsymbol{f} = \widetilde{\boldsymbol{K}}^*\boldsymbol{u}. \tag{4.3}$$

The ULV factorization $\widetilde{\boldsymbol{F}}$ is then used as a preconditioner for the conjugate gradient (CG) method applied to the HPD matrix $\widetilde{\boldsymbol{K}}^*\widetilde{\boldsymbol{K}}$.

4.2 Complexity Analysis

For both 1D and 2D problems, the BF approximation $\widetilde{\boldsymbol{K}}$ can be constructed in $\mathcal{O}(N \log N)$ time [18], and both $\widetilde{\boldsymbol{K}}$ and $\widetilde{\boldsymbol{K}}^*$ can be applied to a vector in $\mathcal{O}(N \log N)$ time. It remains to analyze the complexity of the black-box construction of the HSS matrix, the ULV factorization, and the solution of the HSS system.

For 1D problems, let $N = 2^L N_L$ be the matrix size, where L is the maximum level of the HSS tree and N_L is the leaf size, which is independent of N. For 2D problems, let $n = 2^L n_L$, where L is the maximum level of the HSS tree and n_L is independent of N. Then $N = n^2 = 4^L N_L$ is the matrix size, where $N_L = n_L^2$ is the leaf size. For both cases, suppose that the rank of the HSS blocks at level $1 \le \ell \le L$ is r_ℓ, with $r_0 = 0$. The effective leaf size is defined as follows. At the leaf level, $m_L = N_L$. For $0 \le \ell \le L-1$, we set $m_\ell = 2r_{\ell+1}$ for 1D problems and $m_\ell = 4r_{\ell+1}$ for 2D problems. The number of samples at level ℓ is $s_\ell = r_\ell + m_\ell + p$, where the oversampling parameter p is a small constant, e.g., $p = 5$. We assume that $r_\ell = \mathcal{O}(1)$ for 1D problems and $r_\ell = \mathcal{O}(n/2^\ell)$ for 2D problems. Under this assumption, $m_\ell = \mathcal{O}(1)$ and $s_\ell = \mathcal{O}(1)$ for 1D problems, while $m_\ell = \mathcal{O}(n/2^\ell)$ and $s_\ell = \mathcal{O}(n/2^\ell)$ for 2D problems.

The complexity of the black-box construction algorithm in Section 3 can be analyzed as follows. We first consider the 1D problem. For level $0 \le \ell \le L$, the cost of applying $\boldsymbol{H}^{[\ell]}$ to the test matrix $\boldsymbol{\Omega}^{[\ell]}$ is

$$\mathcal{O}\left(\left(T_{\mathrm{BF}} + \sum_{j=\ell+1}^{L} 2^j m_j r_j\right) s_\ell\right),$$

where T_{BF} is the cost of applying the BF $\widetilde{\boldsymbol{K}}$ or $\widetilde{\boldsymbol{K}}^*$ to a vector. For $1 \le \ell \le L$ and a node τ at level ℓ, the cost of computing $\boldsymbol{U}_\tau$ is $\mathcal{O}(s_\ell m_\ell^2 + s_\ell m_\ell r_\ell + m_\ell r_\ell^2)$, and the cost of computing $\check{\boldsymbol{D}}_\tau$ is $\mathcal{O}(s_\ell m_\ell^2 + m_\ell^2 r_\ell)$.

For the root node, the cost of computing $\boldsymbol{D}_\rho$ is $\mathscr{O}(s_0 m_0^2)$. The postprocessing step in Section 3.3 has cost $\mathscr{O}\big(\sum_{\ell=1}^{L} 2^\ell (m_\ell r_\ell^2 + m_\ell^2 r_\ell)\big)$. Therefore, for 1D problems, the total cost is

$$\begin{aligned} t_{\mathrm{cHSS;1d}} &= \mathscr{O}\left(\sum_{\ell=0}^{L}\left(T_{\mathrm{BF}} + \sum_{j=\ell+1}^{L} 2^j m_j r_j\right) s_\ell + \sum_{\ell=0}^{L} 2^\ell \left(s_\ell m_\ell^2 + s_\ell m_\ell r_\ell + m_\ell r_\ell^2 + m_\ell^2 r_\ell\right)\right) \\ &= \mathscr{O}\left(\sum_{\ell=0}^{L}\left(N \log N + 2^L\right)\right) = \mathscr{O}(N \log^2 N). \end{aligned}$$

Similarly, for 2D problems, the total cost is

$$\begin{aligned} t_{\mathrm{cHSS;2d}} &= \mathscr{O}\left(\sum_{\ell=0}^{L}\left(T_{\mathrm{BF}} + \sum_{j=\ell+1}^{L} 4^j m_j r_j\right) s_\ell + \sum_{\ell=0}^{L} 4^\ell \left(s_\ell m_\ell^2 + s_\ell m_\ell r_\ell + m_\ell r_\ell^2 + m_\ell^2 r_\ell\right)\right) \\ &= \mathscr{O}\left(\sum_{\ell=0}^{L}\left(\left(N \log N + n^2 (L - \ell)\right)\frac{n}{2^\ell}\right) + \sum_{\ell=0}^{L} 4^\ell \left(\frac{n}{2^\ell}\right)^3\right) = \mathscr{O}(N^{1.5} \log N). \end{aligned}$$

The ULV factorization and storage complexities follow from the rank-dependent analysis in [28, Theorem 6.1]. For 1D problems, the bounded-rank assumption corresponds to case 1 with $p = 0$, which gives $t_{\mathrm{fHSS;1d}} = \mathscr{O}(N)$ and an $\mathscr{O}(N)$ storage requirement for the ULV factors. For 2D problems, adapting the levelwise argument of case 2(c) from the binary tree considered there to the quadtree used here gives $t_{\mathrm{fHSS;2d}} = \mathscr{O}(N^{1.5})$ and an $\mathscr{O}(N \log N)$ storage requirement. Since each stored local factor is applied only a constant number of times during an HSS solve, the solve cost has the same asymptotic order as the corresponding storage requirement. Therefore, $t_{\mathrm{sHSS;1d}} = \mathscr{O}(N)$ and $t_{\mathrm{sHSS;2d}} = \mathscr{O}(N \log N)$. These results are summarized in Table 4.1.

Dimension	cBF	aBF	cHSS	fHSS	sHSS
1D	$\mathscr{O}(N \log N)$	$\mathscr{O}(N \log N)$	$\mathscr{O}(N \log^2 N)$	$\mathscr{O}(N)$	$\mathscr{O}(N)$
2D	$\mathscr{O}(N \log N)$	$\mathscr{O}(N \log N)$	$\mathscr{O}(N^{1.5} \log N)$	$\mathscr{O}(N^{1.5})$	$\mathscr{O}(N \log N)$

Table 4.1: Complexity of the main steps in the proposed approximate inversion algorithm.

5 Numerical Results

In this section, we present numerical results for the proposed algorithm for approximating the inverse of FIOs. The phase functions in the FIOs take the form $\phi(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{x} \cdot \boldsymbol{\xi} + \rho(\boldsymbol{x}, \boldsymbol{\xi})$, where $\rho(\boldsymbol{x}, \boldsymbol{\xi})$ is periodic in $\boldsymbol{x}$. We solve the linear system (1.2) through the approximate normal equation (4.3). For the direct solver, the solution is computed by solving the HSS system using the ULV factorization $\widetilde{\boldsymbol{F}}$. For the iterative solver, we use the MATLAB command `pcg`, and the solution is computed by the conjugate gradient (CG) method either without preconditioning or with the ULV factorization $\widetilde{\boldsymbol{F}}$ as a preconditioner. For 1D problems, we let $N = n$ range over $\{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\}$. For 2D problems, we let n range over $\{64, 128, 256, 512\}$ and set $N = n^2$. The real and imaginary parts of $\boldsymbol{f}$ have independent standard Gaussian entries, and the right-hand side is set to $\boldsymbol{u} = \widetilde{\boldsymbol{K}}\boldsymbol{f}$. This choice assesses the HSS-based inversion separately from the BF approximation error, which is reported independently. All algorithms are implemented in MATLAB R2023b, and the BF is computed using the code in [16]. All experiments are carried out on a server with an Intel Gold 6226R CPU at 2.90 GHz and 1000.6 GB of RAM.

The following quantities are used to evaluate the performance of the algorithms. We denote the walltimes for BF construction, HSS construction, ULV factorization, and HSS solution by t_{cBF}, t_{cHSS}, t_{fHSS}, and t_{sHSS}, respectively. All walltimes are reported in seconds. The total offline setup cost is $t_{\mathrm{offline}} = t_{\mathrm{cBF}}+t_{\mathrm{cHSS}}+t_{\mathrm{fHSS}}$, while t_{sHSS} measures the online HSS solve time. The tolerance parameters for the BF construction and HSS construction are fixed at 10^{-8} and 10^{-3}, respectively. The relative approximation errors are defined as

$$e_{\mathrm{cBF}} = \frac{\|\widetilde{\boldsymbol{K}}\boldsymbol{f} - \boldsymbol{K}\boldsymbol{f}\|_2}{\|\boldsymbol{K}\boldsymbol{f}\|_2}, \quad e_{\mathrm{cHSS}} = \frac{\|\widetilde{\boldsymbol{G}}\boldsymbol{f} - \widetilde{\boldsymbol{K}}^*\widetilde{\boldsymbol{K}}\boldsymbol{f}\|_2}{\|\widetilde{\boldsymbol{G}}\boldsymbol{f}\|_2}.$$

We estimate e_{cBF} by restricting both norms to 256 randomly sampled output entries, without explicitly forming $\boldsymbol{K}$. For a computed solution $\widehat{\boldsymbol{f}}$, the relative error is $e_{\mathrm{s}} = \|\boldsymbol{f} - \widehat{\boldsymbol{f}}\|_2/\|\boldsymbol{f}\|_2$, with e_{direct} denoting the direct solver error. For CG with or without preconditioning, the relative residual tolerance is 10^{-12}. For CG applied to (4.3), the walltime, iteration count, and solution error are denoted by $(t_{\mathrm{iter}}, n_{\mathrm{iter}}, e_{\mathrm{iter}})$ without preconditioning and $(t_{\mathrm{piter}}, n_{\mathrm{piter}}, e_{\mathrm{piter}})$ with preconditioning by $\widetilde{\boldsymbol{F}}$.

5.1 1D FIO With Variable Amplitude

We begin with a 1D FIO whose amplitude and phase functions are given by

$$a(\boldsymbol{x},\boldsymbol{\xi}) = \sum_{k=1}^{m} \exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{x}_k)^2 + ((\boldsymbol{\xi}-\boldsymbol{\xi}_k)/N)^2}{\sigma^2}\right),$$

$$\phi(\boldsymbol{x},\boldsymbol{\xi}) = \boldsymbol{x}\cdot\boldsymbol{\xi} + \rho(\boldsymbol{x},\boldsymbol{\xi}), \quad \rho(\boldsymbol{x},\boldsymbol{\xi}) = \frac{2+\sin(2\pi\boldsymbol{x})}{8}|\boldsymbol{\xi}|,$$

where $\boldsymbol{x}_k$ and $\boldsymbol{\xi}_k$ are randomly generated points in $[0,1]\times[-N/2, N/2]$ for $1 \le k \le m$, and σ^2 is a positive constant. In our experiments, m is set to 10 and σ^2 is set to 0.1.

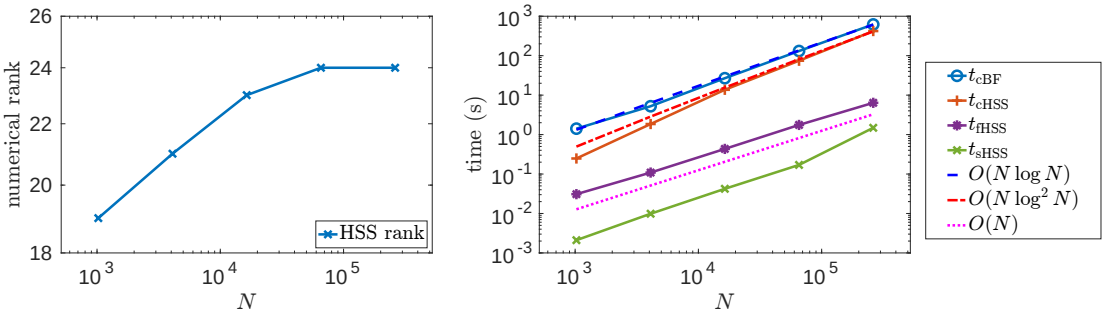

Figure 5.1: HSS rank and time scaling of the direct solver for a 1D FIO with variable amplitude.

Figure 5.1 and Table 5.1 report the HSS ranks, walltimes, and accuracy of the direct solver. The HSS ranks remain essentially bounded as N increases, which is consistent with the rank assumption used in the complexity analysis in Section 4.2. The observed walltimes also agree with the predicted scaling: the BF construction and application scale as $\mathcal{O}(N \log N)$, while the HSS construction and the ULV factorization exhibit the expected $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N)$ behavior, respectively. In this example, the BF construction takes the largest part of the time. The HSS approximation and the solution computed by the direct solver are stable for all tested problem sizes, with e_{cHSS} on the order of 10^{-5} and e_{s} on the order of 10^{-4}.

Table 5.2 summarizes the results of the iterative solver. Without preconditioning, CG requires about 160 iterations for all values of N. In contrast, preconditioning CG with $\widetilde{\boldsymbol{F}}$ reduces the iteration count to only 4 or 5, with solution errors below 3×10^{-12} for all tested problem sizes. Moreover, the iteration time

N	t_{cBF}	e_{cBF}	t_{cHSS}	e_{cHSS}	t_{fHSS}	t_{sHSS}	e_{direct}
2^{10}	1.4e+00	3.7e-06	2.5e-01	2.0e-05	3.1e-02	2.1e-03	1.2e-04
2^{12}	5.2e+00	3.2e-06	1.9e+00	2.9e-05	1.1e-01	9.9e-03	1.7e-04
2^{14}	2.7e+01	3.6e-06	1.4e+01	3.0e-05	4.3e-01	4.2e-02	1.9e-04
2^{16}	1.3e+02	5.3e-06	7.4e+01	3.1e-05	1.8e+00	1.7e-01	1.9e-04
2^{18}	6.2e+02	4.2e-06	4.2e+02	3.3e-05	6.4e+00	1.5e+00	2.1e-04

Table 5.1: Results of the direct solver for a 1D FIO with variable amplitude.

N	t_{iter}	n_{iter}	e_{iter}	t_{offline}	t_{piter}	n_{piter}	e_{piter}
2^{10}	5.0e-01	141	5.7e-12	1.7e+00	3.5e-02	4	1.5e-14
2^{12}	2.9e+00	159	5.3e-12	7.2e+00	1.4e-01	4	5.1e-13
2^{14}	1.5e+01	164	6.0e-12	4.1e+01	6.3e-01	4	6.8e-13
2^{16}	7.3e+01	166	5.9e-12	2.1e+02	2.8e+00	4	2.2e-12
2^{18}	3.2e+02	166	6.1e-12	1.0e+03	2.0e+01	5	1.0e-13

Table 5.2: Results of the iterative solver for a 1D FIO with variable amplitude.

t_{iter} for unpreconditioned CG is nearly 20 times that for preconditioned CG across all tested problem sizes. These results show that the proposed approximate inverse is an effective preconditioner for the normal equation.

5.2 2D FIO With Constant Amplitude

In this example, we consider a constant amplitude function $a(\boldsymbol{x}, \boldsymbol{\xi}) = 1$ and the phase function

$$\phi(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{x} \cdot \boldsymbol{\xi} + \rho(\boldsymbol{x}, \boldsymbol{\xi}), \quad \rho(\boldsymbol{x}, \boldsymbol{\xi}) = \sqrt{c_1^2(\boldsymbol{x})\xi_1^2 + c_2^2(\boldsymbol{x})\xi_2^2},$$
$$c_1(\boldsymbol{x}) = \frac{2 + \sin(2\pi x_1)\sin(2\pi x_2)}{16}, \quad c_2(\boldsymbol{x}) = \frac{2 + \cos(2\pi x_1)\cos(2\pi x_2)}{16}.$$

N	t_{cBF}	e_{cBF}	t_{cHSS}	e_{cHSS}	t_{fHSS}	t_{sHSS}	e_{direct}
64^2	2.6e+01	3.4e-05	1.6e+01	8.5e-04	3.7e-01	1.8e-02	9.7e-04
128^2	1.3e+02	3.0e-05	2.4e+02	1.2e-03	3.0e+00	8.4e-02	1.4e-03
256^2	1.6e+03	1.6e-05	3.2e+03	1.6e-03	1.9e+01	3.9e-01	1.8e-03
512^2	8.7e+03	2.2e-05	7.3e+04	1.9e-03	1.1e+02	1.9e+00	2.1e-03

Table 5.3: Results of the direct solver for a 2D FIO with constant amplitude.

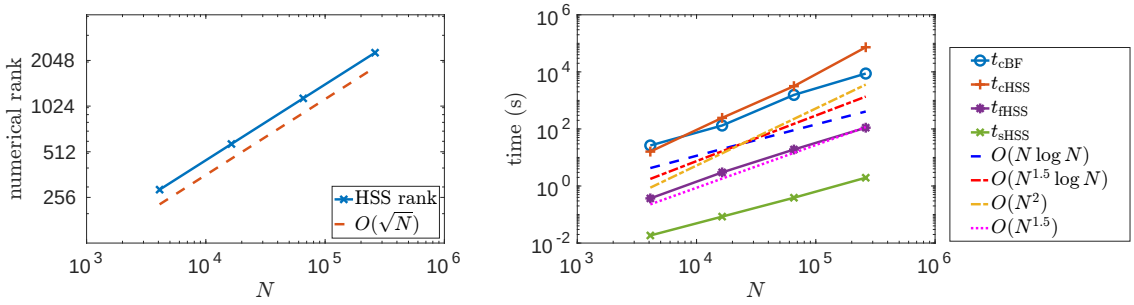

Figure 5.2: HSS rank and time scaling of the direct solver for a 2D FIO with constant amplitude.

N	t_{iter}	n_{iter}	e_{iter}	t_{offline}	t_{piter}	n_{piter}	e_{piter}
64^2	3.5e+00	30	4.3e-13	4.3e+01	8.0e-01	5	7.2e-13
128^2	3.1e+01	31	8.8e-13	3.8e+02	7.1e+00	6	2.7e-14
256^2	2.0e+02	32	7.4e-13	4.7e+03	4.6e+01	6	3.0e-13
512^2	1.1e+03	32	9.3e-13	8.2e+04	2.4e+02	6	7.0e-13

Table 5.4: Results of the iterative solver for a 2D FIO with constant amplitude.

Figure 5.2 and Table 5.3 report the HSS ranks, walltimes, and accuracy of the direct solver. The observed HSS ranks of the constructed HSS representations grow consistently with the expected $\mathcal{O}(\sqrt{N})$ scaling. The walltimes for BF construction, HSS ULV factorization, and solving the HSS system also agree with the complexity estimates in Section 4.2. The HSS construction time, however, exhibits an empirical scaling closer to $\mathcal{O}(N^2)$, rather than the predicted $\mathcal{O}(N^{1.5} \log N)$ scaling. This discrepancy is caused by the practical implementation of the BF in (2.1). In the implementation, the super-index of the outermost factors $\boldsymbol{U}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{V}$ is not L, but rather $\max\{L-3, h\}$, where $L = \log_2 n$. Consequently, the number of $\boldsymbol{B}$ and $\boldsymbol{C}$ factors in the BF is

$$\max\{L-3, h\} - h = \max\{\text{floor}(L/2) - 3, 0\}.$$

For $N \leq 128^2$, we have $L \leq 7$, and hence no $\boldsymbol{B}$ or $\boldsymbol{C}$ factors are present. For $N = 256^2$ and 512^2, we have $L = 8$ and 9, respectively, so only one $\boldsymbol{B}$ factor and one $\boldsymbol{C}$ factor are present. In these regimes, the application cost of the BF deteriorates to $\mathcal{O}(N^{1.5})$, which in turn makes the HSS construction time scale empirically like $\mathcal{O}(N^2)$. Note that the HSS construction becomes the most time-consuming part in this example. Despite this implementation-dependent slowdown, the relative errors of the HSS approximation and the direct solver remain on the order of 10^{-3} for all tested problem sizes, indicating that the HSS construction and the direct solver are stable.

Table 5.4 summarizes the results of the iterative solver. Without preconditioning, CG requires about 30 iterations for all values of N, suggesting that the condition number of the approximate normal equation (4.3) remains moderate and does not grow significantly with N. Preconditioning CG with $\widetilde{\boldsymbol{F}}$ further reduces the iteration count to 5 or 6, while maintaining solution errors below 10^{-12} for all tested problem sizes. The iteration time t_{iter} for unpreconditioned CG is nearly 5 times that for preconditioned CG. These results show that the proposed approximate inverse remains an effective preconditioner for 2D FIOs with constant amplitude.

5.3 2D FIO With Variable Amplitude

In this example, the amplitude and phase functions are given by

$$a(\boldsymbol{x},\boldsymbol{\xi}) = H_0^{(1)}(2\pi\rho(\boldsymbol{x},\boldsymbol{\xi}))\mathrm{e}^{-2\pi\imath\rho(\boldsymbol{x},\boldsymbol{\xi})},$$
$$\phi(\boldsymbol{x},\boldsymbol{\xi}) = \boldsymbol{x}\cdot\boldsymbol{\xi} + \rho(\boldsymbol{x},\boldsymbol{\xi}), \quad \rho(\boldsymbol{x},\boldsymbol{\xi}) = c(\boldsymbol{x})\|\boldsymbol{\xi}\|, \quad c(\boldsymbol{x}) = \frac{3+\sin(2\pi x_1)\sin(2\pi x_2)}{8},$$

where $H_0^{(1)}$ denotes the Hankel function of the first kind of order 0, and $\|\cdot\|$ denotes the Euclidean norm.

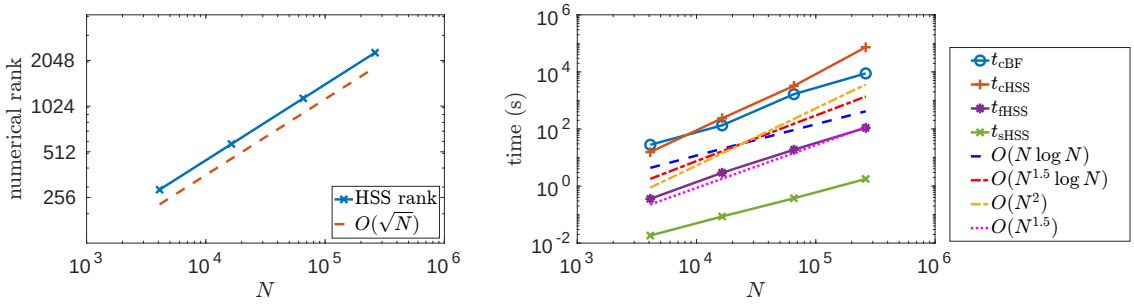

Figure 5.3: HSS rank and time scaling of the direct solver for a 2D FIO with variable amplitude.

N	t_{cBF}	e_{cBF}	t_{cHSS}	e_{cHSS}	t_{fHSS}	t_{sHSS}	e_{direct}
64^2	2.8e+01	2.1e-03	1.5e+01	1.3e-03	3.6e-01	1.8e-02	3.8e-03
128^2	1.3e+02	5.3e-04	2.4e+02	1.6e-03	2.9e+00	8.6e-02	3.9e-03
256^2	1.7e+03	2.6e-04	3.2e+03	1.6e-03	1.9e+01	3.7e-01	3.4e-03
512^2	8.8e+03	2.3e-04	7.3e+04	1.8e-03	1.1e+02	1.8e+00	3.2e-03

Table 5.5: Results of the direct solver for a 2D FIO with variable amplitude.

N	t_{iter}	n_{iter}	e_{iter}	t_{offline}	t_{piter}	n_{piter}	e_{piter}
64^2	1.3e+01	114	2.0e-12	4.4e+01	8.9e-01	6	1.1e-12
128^2	1.6e+02	161	1.5e-12	3.7e+02	7.2e+00	6	1.4e-12
256^2	1.3e+03	223	1.8e-12	4.9e+03	4.8e+01	6	8.7e-13
512^2	1.1e+04	317	1.4e-12	8.2e+04	2.5e+02	6	6.8e-13

Table 5.6: Results of the iterative solver for a 2D FIO with variable amplitude.

Figure 5.3 and Table 5.5 report the HSS ranks, walltimes, and accuracy of the direct solver. The observed HSS ranks of the constructed HSS representations grow consistently with the expected $\mathcal{O}(\sqrt{N})$ scaling, as in the constant-amplitude case. The walltimes of all components of the direct solver, except for HSS construction, agree with the complexity estimates in Section 4.2. As in Section 5.2, the empirical

$\mathcal{O}(N^2)$ scaling of the HSS construction time is caused by the implementation-dependent cost of applying the BF. The most time-consuming part of the direct solver is again the HSS construction. The relative errors of the HSS approximation and the solution computed by the direct solver remain on the order of 10^{-3} for all tested problem sizes, indicating that the HSS construction and the direct solver are stable.

Table 5.6 summarizes the results of the iterative solver. Without preconditioning, the number of CG iterations increases from 114 to 317 as N increases, suggesting that the conditioning of the approximate normal equation (4.3) deteriorates with the problem size. In contrast, using $\widetilde{\boldsymbol{F}}$ as a preconditioner reduces the iteration count to 6 for all tested problem sizes, while maintaining a solution error on the order of 10^{-12}. For $N = 256^2$ and 512^2, the iteration time t_{iter} for unpreconditioned CG is nearly 27 and 44 times that for preconditioned CG, respectively. These results demonstrate that the proposed preconditioner is effective for this problem, and the speedup becomes more significant as N increases.

6 Conclusions

In this paper, we introduced an algorithm for approximating the inverse of the discrete FIO $\boldsymbol{K}$. The offline stage of the algorithm consists of three steps: constructing a BF approximation $\widetilde{\boldsymbol{K}} \approx \boldsymbol{K}$, constructing an HSS approximation $\widetilde{\boldsymbol{G}} \approx \boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$, and computing the ULV factorization of $\widetilde{\boldsymbol{G}}$ to obtain $\widetilde{\boldsymbol{F}} \approx \boldsymbol{G}^{-1}$. Using these offline components, the approximate inverse of the discrete FIO is applied in the online stage as $\boldsymbol{K}^{-1} \approx \widetilde{\boldsymbol{F}}\widetilde{\boldsymbol{K}}^*$. For the HSS construction, we developed a black-box method that only requires matrix-vector products with the target matrix. This method is efficient in the present setting because these matrix-vector products can be performed rapidly using the BF approximation.

The proposed algorithm provides an HSS-based alternative to the $\mathcal{H}$-matrix and HIF approach in [7]. It combines the nested shared bases of the HSS representation with a black-box construction algorithm and ULV factorization. The HSS representation reuses basis information across levels, and its generators support a recursive ULV factorization. The black-box construction uses independent random test matrices at different levels, allowing the sampling to be adapted to the numerical ranks at each level.

For 1D FIOs, the offline costs for constructing the BF approximation, constructing the HSS approximation, and computing the ULV factorization are $\mathcal{O}(N \log N)$, $\mathcal{O}(N \log^2 N)$, and $\mathcal{O}(N)$, respectively, while the online solution cost is $\mathcal{O}(N \log N)$. For 2D FIOs, the corresponding offline costs are $\mathcal{O}(N \log N)$, $\mathcal{O}(N^{1.5} \log N)$, and $\mathcal{O}(N^{1.5})$, respectively, while the online solution cost is $\mathcal{O}(N \log N)$. The resulting approximate inverse can be used as a direct solver, while $\widetilde{\boldsymbol{F}}$ can be used as a preconditioner for CG applied to the approximate normal equation. In the reported experiments, this preconditioner kept the iteration count small and nearly independent of the problem size, while substantially reducing the iteration time relative to unpreconditioned CG.

There are several directions for future work. First, the proposed algorithm can be extended to higher-dimensional FIOs in a straightforward manner by using the same framework. However, in higher dimensions, the HSS ranks may grow with the problem size, leading to higher computational complexity. Second, a parallel implementation of the proposed algorithm could be developed to further reduce the walltime, especially for large-scale problems. Indeed, the main components of the algorithm, including BF construction, HSS construction, and ULV factorization, are all well suited for parallelization. Third, establishing the HSS property of the matrix $\boldsymbol{G}$ and analyzing the corresponding HSS ranks are important theoretical questions. Such results would provide a rigorous foundation for the complexity analysis of the proposed algorithm. Finally, in [7], the authors used $\mathcal{H}$-matrices with strong admissibility to approximate $\boldsymbol{G} = \boldsymbol{K}^*\boldsymbol{K}$ for 2D problems. Their experiments indicated that the ranks of the admissible blocks grow approximately as $\mathcal{O}(\log N)$. It would therefore be interesting to investigate whether $\mathcal{H}^2$-matrix approximations [10] and their inverses [3, 22] can be used for high-dimensional problems. This may lead to a more efficient algorithm for approximating the inverse of FIOs in higher dimensions.

References

[1] A. H. Barnett, J. Magland, and L. af Klinteberg, *A parallel nonuniform fast Fourier transform library based on an "exponential of semicircle" kernel*, SIAM J. Sci. Comput., 41 (2019), pp. C479–C504.

[2] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Boundary Elem., 27 (2003), pp. 405–422.

[3] W. Boukaram, D. Keyes, X. Li, Y. Liu, and G. Turkiyyah, *Linear complexity $\mathscr{H}^2$ direct solver for fine-grained parallel architectures*, IMA J. Numer. Anal., (2026), p. drag030.

[4] E. Candès, L. Demanet, and L. Ying, *Fast computation of Fourier integral operators*, SIAM J. Sci. Comput., 29 (2007), pp. 2464–2493.

[5] ———, *A fast butterfly algorithm for the computation of Fourier integral operators*, Multiscale Model. Simul., 7 (2009), pp. 1727–1750.

[6] M. Cheney and B. Borden, *Fundamentals of radar imaging*, SIAM, Philadelphia, PA, 2009.

[7] J. Feliu-Fabá and L. Ying, *Approximate inversion of discrete Fourier integral operators*, J. Comput. Phys., 446 (2021), p. 110654.

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

[9] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[10] W. Hackbusch, B. N. Khoromskij, and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, 2000, pp. 9–29.

[11] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 409–436.

[12] K. L. Ho and L. Ying, *Hierarchical interpolative factorization for elliptic operators: Differential equations*, Commun. Pure Appl. Math., 69 (2016), pp. 1415–1451.

[13] ———, *Hierarchical interpolative factorization for elliptic operators: Integral equations*, Commun. Pure Appl. Math., 69 (2016), pp. 1314–1353.

[14] P. M. Kielstra, T. Shi, H. Luo, J. Qian, and Y. Liu, *A linear-complexity tensor butterfly algorithm for compressing high-dimensional oscillatory integral operators*, Multiscale Model. Simul., 23 (2025), pp. 864–893.

[15] J. Levitt and P.-G. Martinsson, *Linear-complexity black-box randomized compression of rank-structured matrices*, SIAM J. Sci. Comput., 46 (2024), pp. A1747–A1763.

[16] Y. Li. `https://github.com/YingzhouLi/FastBF.m`, 2016.

[17] Y. Li and J. Liu, *A superfast direct solver for type-III inverse nonuniform discrete Fourier transform*, arXiv preprint arXiv:2512.03733, (2025).

[18] Y. Li and H. Yang, *Interpolative butterfly factorization*, SIAM J. Sci. Comput., 39 (2017), pp. A503–A531.

[19] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, *Butterfly factorization*, Multiscale Model. Simul., 13 (2015), pp. 714–732.

[20] Y. Li and L. Ying, *Distributed-memory hierarchical interpolative factorization*, Res. Math. Sci., 4 (2017), p. 12.

[21] 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.

[22] Q. Ma, S. Deshmukh, and R. Yokota, *Scalable linear time dense direct solver for 3-D problems without trailing sub-matrix dependencies*, in SC22, 2022, pp. 1–12.

[23] P.-G. Martinsson, *Fast direct solvers for elliptic PDEs*, SIAM, Philadelphia, PA, 2019.

[24] P.-G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.

[25] D. Potts, G. Steidl, and M. Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, in Modern Sampling Theory: Mathematics and Applications, J. J. Benedetto and P. J. S. G. Ferreira, eds., Birkhäuser Boston, Boston, MA, 2001, pp. 247–270.

[26] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, PA, 2003.

[27] W. W. Symes, *The seismic reflection inverse problem*, Inverse Probl., 25 (2009), p. 123008.

[28] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.

[29] 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.