

A Superfast Direct Solver for Type-III Inverse Nonuniform Discrete Fourier Transform

Yingzhou Li*, Jingyu Liu†

December 4, 2025

Abstract

The nonuniform discrete Fourier transform (NUDFT) and its inverse are widely used in various fields of scientific computing. In this article, we propose a novel superfast direct inversion method for type-III NUDFT. The proposed method approximates the type-III NUDFT matrix as a product of a type-II NUDFT matrix and an HSS matrix, where the type-II NUDFT matrix is further decomposed into the product of an HSS matrix and an uniform discrete Fourier transform (DFT) matrix as in [31]. This decomposition enables both the forward application and the backward inversion to be accomplished with quasi-linear complexity. The fast inversion can serve as a high-accuracy direct solver or as an efficient preconditioner. Additionally, we provide an error bound for the approximation under specific sample distributions. Numerical results are presented to verify the relevant theoretical properties and demonstrate the efficiency of the proposed methods.

Keywords nonuniform discrete Fourier transform, hierarchically semi-separable matrix

1 Introduction

In this paper, a superfast direct solver is proposed for the *nonuniform discrete Fourier transform* (NUDFT). In one dimension, the NUDFT forward problem aims to compute the *target values*

$$f_j = \sum_{k=0}^{N-1} e^{2\pi i x_j \omega_k} u_k, \quad 0 \leq j \leq M-1, \tag{1.1}$$

where $x_0, x_1, \dots, x_{M-1} \in [0, 1]$ are *sample points*, $\omega_0, \omega_1, \dots, \omega_{N-1} \in [-1/2, N-1/2]$ are *frequencies* and $u_0, u_1, \dots, u_{N-1} \in \mathbb{C}$ are *coefficients*. We assume throughout the paper that $M \geq N$. Since the forward problem is a matrix-vector multiplication, direct computation requires $\mathcal{O}(MN)$ operations. If $M = N$, the sample points are equispaced and the frequencies are integer, i.e., $x_j = j/M$ and $\omega_k = k$, the NUDFT becomes uniform and (1.1) can be computed using the *fast Fourier transform* (FFT) algorithm in $\mathcal{O}(N \log N)$ operations [8]. Unfortunately, the FFT cannot be directly applied when either the sample points are nonequispaced or the frequencies are noninteger, or both. In these cases, the NUDFT problem can be categorized into three types according to the nonuniformity [11, 30]:

*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

- Type-I: Equispaced sample points and noninteger frequencies, i.e., $x_j = j/M$;
- type-II: Nonequispaced sample points and integer frequencies, i.e., $\omega_k = k$;
- type-III: Nonequispaced sample points and noninteger frequencies.

Various algorithms have been developed to address the NUDFT forward problem with a complexity of $\mathcal{O}(M + N \log N)$ while maintaining the desired accuracy. These algorithms are commonly referred to as the *nonuniform fast Fourier transform* (NUFFT). Some methods approximates the NUDFT matrix using the Hadamard product of the discrete Fourier transform (DFT) matrix and a low-rank matrix [1, 30], while others utilize convolution with a smooth, nearly locally supported function combined with the FFT on an oversampled grid [2, 10, 11, 13, 21, 29]. For more general problems such as the Fourier integral operator (FIO), *butterfly algorithms* have been designed for efficient forward computation [5, 23].

In this article, we are mainly interested in the inverse problem of the NUDFT:

Problem 1.1 (Inverse NUDFT, INUDFT). *Given sample points $\{x_j\}$ and frequencies $\{\omega_k\}$, determine the coefficients $\{u_k\}$ from target values $\{f_j\}$.*

Algebraically, this problem can be modeled as a least-squares problem associated with the NUDFT matrix \mathbf{A} , whose entries are given by $\mathbf{A}(j, k) = e^{2\pi i x_j \omega_k}$. Unlike the uniform case, the pseudoinverse of the NUDFT matrix is not simply equal to its adjoint, implying that the fast inverse algorithm differs from that used for the forward problem.

Another alternative perspective on the inverse problem is to view it as a function approximation task. Specifically, given values $\{f(x_j)\}$ of a function $f: [0, 1] \rightarrow \mathbb{C}$ at sample points $\{x_j\}$, we aim to determine the coefficients $\{u_k\}$ such that the expansion $\sum_k u_k e^{2\pi i \omega_k x}$ provides a good approximation of f . According to the Kadec-1/4 theorem [35], when frequencies are perturbed integers satisfying $\sup_{k \in \mathbb{Z}} |\omega_k - k| \leq \alpha$ with $0 \leq \alpha < 1/4$, the nonharmonic Fourier modes $\{e^{2\pi i \omega_k x}\}_{k \in \mathbb{Z}}$ form a Riesz basis of $L^2[0, 1]$. In this case, the problem can also be interpreted as finding the coefficients of a given function on some nonorthonormal basis.

Typically, the INUDFT problem is solved using iterative methods [12, 30] due to the efficiency of its forward computation. Direct methods have also been developed, as noted in [10, 20]. Recently, in [31], the authors proposed a superfast direct inversion method for the type-II INUDFT problem. A key property leveraged in this work is that type-II NUDFT matrix \mathbf{A} is a Vandermonde matrix, thus satisfies a Sylvester matrix equation with a low-rank right-hand side. Consequently, the transformed matrix $\tilde{\mathbf{A}} = \mathbf{AF}^{-1}$, can be compressed into a *hierarchically semi-separable* (HSS) matrix [34], where $\mathbf{F} \in \mathbb{C}^{N \times N}$ is the DFT matrix defined by $\mathbf{F}(j, k) = e^{-2\pi i j k / N}$. Ultimately, by combining the HSS least-squares solver [33] with the FFT, a direct solver for the type-II INUDFT problem is achieved. The proposed method could be naturally extended to the type-I INUDFT problem but not to the type-III INUDFT problem.

1.1 Contributions

The main purpose of this paper is to develop a direct solver for the type-III INUDFT problem. By projecting the frequencies onto the integers, the type-III NUDFT matrix is approximated as $\mathbf{A} \approx \mathbf{BH}$, where $\mathbf{B} \in \mathbb{C}^{M \times N}$ is a type-II NUDFT matrix and $\mathbf{H} \in \mathbb{C}^{N \times N}$ is a dense matrix that can be compressed into an HSS matrix. We provide a theoretical error bound for this approximation under the assumption that the samples points are independent identically distributed (i.i.d.) uniform random variables.

By applying relevant fast inversion algorithms to \mathbf{B} and \mathbf{H} , the least-squares problem associated with \mathbf{A} can be solved rapidly. To be more specific, the usage of this superfast direct solver can be divided into three stages: Construction, factorization and solution. In the construction stage, after building a fast structure \mathbf{B}_{fast} for the type-II NUDFT matrix \mathbf{B} , the matrix $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$ is compressed into its HSS form, denoted as \mathbf{H}_{HSS} . This compression is performed efficiently using an algorithm based on random sampling. In the factorization stage, the HSS matrix \mathbf{H}_{HSS} undergoes the URV factorization [33]. Once the factorization stage finishes, we use $\mathbf{A}_{\text{fast}}^\dagger = \mathbf{H}_{\text{HSS}}^{-1} \mathbf{B}_{\text{fast}}^\dagger$ as a direct solver for the least-squares problem. Additionally,

when considering the normal equation of (1.1), our solver can also serve as an efficient preconditioner in preconditioned conjugate gradient (PCG) methods. Numerical experiments are conducted to investigate the properties of the approximation and to demonstrate the efficiency of the proposed method.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we review the related definitions and algorithms related to HSS matrices, as well as the type-II INUDFT solver [31], which serves as a preliminary for the subsequent sections. Section 3 introduces our novel method on the type-III NUDFT problem. We discuss first its derivation and related theoretical results, followed by the practical implementation of the algorithm. Numerical results are presented in Section 4 to illustrate the efficiency of our methods. Finally, Section 5 concludes the paper with a discussion on future directions.

2 Preliminaries

2.1 HSS Matrices

HSS matrices is a special class of \mathcal{H}^2 -matrices [4, 15, 17]. They exhibit linear complexity for storage and numerical linear algebra operations, such as matrix-vector products and solving linear systems [34]. In this paper, we employ the generalized definition for possibly rectangular HSS matrices. The definition of an HSS matrix relies on a tree structure (HSS tree), which is included in the following definition.

Definition 2.1 ([33]). *An $M \times N$ matrix \mathbf{A} is called an HSS matrix with the associated HSS tree T if the following conditions hold:*

- T is a full binary tree with root node 1.
- There are two index sets I_τ and J_τ associated with each node τ of T , satisfying the following conditions
 - (1) $I_1 = [0, \dots, M - 1]$, $J_1 = [0, \dots, N - 1]$,
 - (2) For a nonleaf node τ with children α_1 and α_2 , we have $I_\tau = I_{\alpha_1} \sqcup I_{\alpha_2}$ and $J_\tau = J_{\alpha_1} \sqcup J_{\alpha_2}$, where the notation \sqcup means the disjoint union.
- There are matrices \mathbf{D}_τ and \mathbf{U}_τ , \mathbf{V}_τ , $\mathbf{B}_{\tau,\sigma}$ called HSS generators associated with each node τ and its sibling σ , satisfying the following recursion:
 - (1) For a nonleaf node τ with children α_1 and α_2 ,

$$\mathbf{D}_\tau = \mathbf{A}(I_\tau, J_\tau) = \begin{bmatrix} \mathbf{D}_{\alpha_1} & \mathbf{U}_{\alpha_1}^{\text{big}} \mathbf{B}_{\alpha_1, \alpha_2} \mathbf{V}_{\alpha_2}^{\text{big}, *} \\ \mathbf{U}_{\alpha_2}^{\text{big}} \mathbf{B}_{\alpha_2, \alpha_1} \mathbf{V}_{\alpha_1}^{\text{big}, *} & \mathbf{D}_{\alpha_2} \end{bmatrix},$$

where in the equation $\mathbf{V}_{\alpha_j}^{\text{big}, *}$ means $(\mathbf{V}_{\alpha_j}^{\text{big}})^*$. This adjoint notation will be used throughout the paper. For a leaf node τ , $\mathbf{D}_\tau = \mathbf{A}(I_\tau, J_\tau)$ is a dense matrix.

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

$$\mathbf{U}_\tau^{\text{big}} = \begin{bmatrix} \mathbf{U}_{\alpha_1}^{\text{big}} & \\ & \mathbf{U}_{\alpha_2}^{\text{big}} \end{bmatrix} \mathbf{U}_\tau \quad \text{and} \quad \mathbf{V}_\tau^{\text{big}} = \begin{bmatrix} \mathbf{V}_{\alpha_1}^{\text{big}} & \\ & \mathbf{V}_{\alpha_2}^{\text{big}} \end{bmatrix} \mathbf{V}_\tau.$$

In some literatures [34] \mathbf{U}_τ and \mathbf{V}_τ are partitioned as

$$\mathbf{U}_\tau = \begin{bmatrix} \mathbf{R}_{\alpha_1} \\ \mathbf{R}_{\alpha_2} \end{bmatrix} \quad \text{and} \quad \mathbf{V}_\tau = \begin{bmatrix} \mathbf{W}_{\alpha_1} \\ \mathbf{W}_{\alpha_2} \end{bmatrix}.$$

Therefore the recursion can be reformulated as

$$\mathbf{U}_\tau^{\text{big}} = \begin{bmatrix} \mathbf{U}_{\alpha_1}^{\text{big}} \mathbf{R}_{\alpha_1} \\ \mathbf{U}_{\alpha_2}^{\text{big}} \mathbf{R}_{\alpha_2} \end{bmatrix} \quad \text{and} \quad \mathbf{V}_\tau^{\text{big}} = \begin{bmatrix} \mathbf{V}_{\alpha_1}^{\text{big}} \mathbf{W}_{\alpha_1} \\ \mathbf{V}_{\alpha_2}^{\text{big}} \mathbf{W}_{\alpha_2} \end{bmatrix}.$$

For a leaf node τ , we define $\mathbf{U}_\tau^{\text{big}} = \mathbf{U}_\tau$ and $\mathbf{V}_\tau^{\text{big}} = \mathbf{V}_\tau$. All matrices \mathbf{U}_τ and \mathbf{V}_τ are assumed to be orthonormal.

A key observation regarding HSS matrices is that $\mathbf{U}_\tau^{\text{big}}$ and $\mathbf{V}_\tau^{\text{big},*}$ are bases for the column or row spaces of the corresponding blocks $\mathbf{A}(I_\tau, J_\tau^c)$ and $\mathbf{A}(I_\tau^c, J_\tau)$, respectively. Here $I_\tau^c = I_1 \setminus I_\tau$ and $J_\tau^c = J_1 \setminus J_\tau$. This is often called the *shared basis property* or *nested basis property*. Matrices $\mathbf{U}_\tau^{\text{big}}$ and $\mathbf{V}_\tau^{\text{big}}$ are called *basis matrices*. Due to the recursion of basis matrices, it suffices to store the matrices \mathbf{U}_τ , \mathbf{V}_τ , \mathbf{B}_τ for all corresponding nodes, along with the matrices \mathbf{D} for leaf nodes. The *HSS rank* of \mathbf{A} is defined as the maximum numerical rank among all the HSS blocks of \mathbf{A} . A well-known result indicates that the complexity for noth storage and matrix-vector product of an HSS matrix with HSS rank k is $\mathcal{O}(k(M+N))$ [6]. When the context is clear, we may omit the index sets I_τ and J_τ in the subscript, using τ directly to represent the corresponding block. For instance, $\mathbf{A}_{\tau,\sigma}$ and $\mathbf{A}_{\tau,\sigma^c}$ represent $\mathbf{A}(I_\tau, J_\sigma)$ and $\mathbf{A}(I_\tau, J_\sigma^c)$ respectively.

The hierarchical structure of HSS matrices is often represented in a different form known as the *telescoping factorization* [25, 26]. For each level $1 \leq \ell \leq L$ in the HSS tree, we define $\mathbf{U}^{(\ell)} = \text{diag}(\mathbf{U}_\tau : \text{level}(\tau) = \ell)$ and $\mathbf{V}^{(\ell)} = \text{diag}(\mathbf{V}_\tau : \text{level}(\tau) = \ell)$. For each level $0 \leq \ell \leq L-1$ and a node τ in level ℓ with children α_1 and α_2 , we introduce

$$\mathbf{B}_\tau = \begin{bmatrix} \mathbf{0} & \mathbf{B}_{\alpha_1, \alpha_2} \\ \mathbf{B}_{\alpha_2, \alpha_1} & \mathbf{0} \end{bmatrix}$$

and define $\mathbf{B}^{(\ell)} = \text{diag}(\mathbf{B}_\tau : \text{level}(\tau) = \ell)$. We define $\mathbf{D}^{(L)} = \text{diag}(\mathbf{D}_\tau : \text{level}(\tau) = L)$ for level L . Furthermore, we set $\mathbf{A}^{(L)} = \mathbf{A}$, then \mathbf{A} can be recursively factorized as:

$$\begin{aligned} \mathbf{A}^{(L)} &= \mathbf{U}^{(L)} \mathbf{A}^{(L-1)} \mathbf{V}^{(L),*} + \mathbf{D}^{(L)}, \\ \mathbf{A}^{(\ell)} &= \mathbf{U}^{(\ell)} \mathbf{A}^{(\ell-1)} \mathbf{V}^{(\ell),*} + \mathbf{B}^{(\ell)}, \quad 1 \leq \ell \leq L-1, \\ \mathbf{A}^{(0)} &= \mathbf{B}^{(0)}. \end{aligned} \tag{2.1}$$

2.2 A Black-Box Construction Algorithm for HSS Matrices

In this section we review the randomized algorithm [22] for computing the HSS compression of a given matrix \mathbf{A} , under the limitation that interactions with the matrix can only occur through applying it or its (conjugate) transpose on vectors. This scenario is often referred to as the *black-box* setting [24]. The HSS compression of a matrix \mathbf{A} is constructed from the information contained in $\{\Omega, \mathbf{A}\Omega, \Psi, \mathbf{A}^*\Psi\}$ where Ω and Ψ are two random *test matrices* and $\mathbf{Y} = \mathbf{A}\Omega$ and $\mathbf{Z} = \mathbf{A}^*\Psi$ are corresponding *sample matrices*. It is worth noting that our algorithm here is slightly different from the one presented in [22], as the original one was proposed for hierarchical block-separable (HBS) matrices.

Suppose $\mathbf{A} \in \mathbb{C}^{N \times N}$ is an HSS matrix with HSS rank k and leaf node size m . Let r be a number slightly larger than k , and let Ω and Ψ be $N \times s$ Gaussian test matrices, where the number of samples $s \geq \max(r+m, 3r)$. The construction can be divided into two steps: First, we conduct a bottom-up process to compute all basis matrices \mathbf{U} and \mathbf{V} . Some auxiliary matrices are also computed in this step. Next, a top-down process is employed to complete the construction by assigning matrices \mathbf{B} and \mathbf{D} .

We begin by describing how to compute matrices \mathbf{U}_τ and \mathbf{V}_τ for a leaf node τ in level $\ell = L$. By considering the row block corresponding to τ in the equation $\mathbf{Y} = \mathbf{A}\Omega$, we have

$$\mathbf{Y}_\tau = \mathbf{A}_{\tau,\tau} \Omega_\tau + \mathbf{A}_{\tau,\tau^c} \Omega_{\tau^c} \tag{2.2}$$

The key observation is that if there exists an orthonormal matrix $\mathbf{P}_\tau \in \mathbb{C}^{s \times r}$ that belongs to the null space of Ω_τ , such that $\Omega_\tau \mathbf{P}_\tau = 0$, multiplying both sides of (2.2) by \mathbf{P}_τ on the right yields $\mathbf{Y}_\tau \mathbf{P}_\tau = \mathbf{A}_{\tau,\tau^c} \Omega_{\tau^c} \mathbf{P}_\tau$.

Such a matrix \mathbf{P}_τ can be obtained by first performing a QR factorization with column pivoting (QRCP) on $\boldsymbol{\Omega}_\tau^*$, then taking the last r columns from the factor \mathbf{Q} . Notably, the contribution from $\mathbf{A}_{\tau,\tau}$ is removed, allowing $\mathbf{Y}_\tau \mathbf{P}_\tau$ to be viewed as a random sampling of \mathbf{A}_{τ,τ^c} . Consequently, the basis matrix \mathbf{U}_τ can be constructed by taking the first r columns from the factor \mathbf{Q} in the QRCP of $\mathbf{Y}_\tau \mathbf{P}_\tau$. A similar procedure using Ψ and Z yields the matrix \mathbf{V}_τ .

With matrices \mathbf{U}_τ and \mathbf{V}_τ of τ in hand, the diagonal block can be decomposed into two parts:

$$\mathbf{D}_\tau = \mathbf{A}_{\tau,\tau} = \mathbf{U}_\tau \mathbf{U}_\tau^* \mathbf{A}_{\tau,\tau} \mathbf{V}_\tau \mathbf{V}_\tau^* + (\mathbf{A}_{\tau,\tau} - \mathbf{U}_\tau \mathbf{U}_\tau^* \mathbf{A}_{\tau,\tau} \mathbf{V}_\tau \mathbf{V}_\tau^*) =: \mathbf{U}_\tau \hat{\mathbf{D}}_\tau \mathbf{V}_\tau^* + \check{\mathbf{D}}_\tau. \quad (2.3)$$

The first term represents the projection of $\mathbf{A}_{\tau,\tau}$ onto basis matrices, while the second term captures the remaining part. We will address the computation of $\hat{\mathbf{D}}_\tau$ later in the top-down process and focus on constructing of $\check{\mathbf{D}}_\tau$ first. By rewriting (2.3), we obtain

$$\check{\mathbf{D}}_\tau = \mathbf{A}_{\tau,\tau} - \mathbf{U}_\tau \mathbf{U}_\tau^* \mathbf{A}_{\tau,\tau} \mathbf{V}_\tau \mathbf{V}_\tau^* = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{A}_{\tau,\tau} + \mathbf{U}_\tau \mathbf{U}_\tau^* \mathbf{A}_{\tau,\tau} (\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*).$$

The only task now is to compute $(\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{A}_{\tau,\tau}$ and $\mathbf{A}_{\tau,\tau} (\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*)$. Multiplying $(\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*)$ on the left of (2.2) and using the fact that \mathbf{U}_τ forms the column basis of \mathbf{A}_{τ,τ^c} , it can be verified that $(\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{Y}_\tau = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{A}_{\tau,\tau} \boldsymbol{\Omega}_\tau$. Therefore, $(\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{A}_{\tau,\tau} = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{Y}_\tau \boldsymbol{\Omega}_\tau^\dagger$ can be recovered through a least-squares solving. A similar process yields $\mathbf{A}_{\tau,\tau} (\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*) = ((\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*) \mathbf{Z}_\tau \boldsymbol{\Psi}_\tau^\dagger)^*$. Combining these results, the final expression for $\check{\mathbf{D}}_\tau$ is obtained as: $\check{\mathbf{D}}_\tau = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{Y}_\tau \boldsymbol{\Omega}_\tau^\dagger + \mathbf{U}_\tau \mathbf{U}_\tau^* ((\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*) \mathbf{Z}_\tau \boldsymbol{\Psi}_\tau^\dagger)^*$.

Define $\mathbf{A}^{(L)} = \mathbf{A}$. Suppose ℓ is the current level to be considered, where $0 \leq \ell \leq L - 1$. By extracting the basis matrices in level $\ell + 1$, the matrix $\mathbf{A}^{(\ell+1)}$ is factorized as follows:

$$\mathbf{A}^{(\ell+1)} = \mathbf{U}^{(\ell+1)} \mathbf{A}^{(\ell)} \mathbf{V}^{(\ell+1),*} + \check{\mathbf{D}}^{(\ell+1)}, \quad (2.4)$$

where

$$\begin{aligned} \mathbf{U}^{(\ell+1)} &= \text{diag}(\mathbf{U}_\tau : \text{level}(\tau) = \ell + 1), \\ \mathbf{V}^{(\ell+1)} &= \text{diag}(\mathbf{V}_\tau : \text{level}(\tau) = \ell + 1), \\ \check{\mathbf{D}}^{(\ell+1)} &= \text{diag}(\check{\mathbf{D}}_\tau : \text{level}(\tau) = \ell + 1). \end{aligned}$$

Therefore, $\mathbf{A}^{(\ell)} = \mathbf{U}^{(\ell+1),*} \mathbf{A}^{(\ell+1)} \mathbf{V}^{(\ell+1)}$. If we “eliminate” all nodes in level $\ell + 1$, it can be verified that $\mathbf{A}^{(\ell)}$ remains an HSS matrix with a maximum level ℓ . The only requirement is to update the generators at level ℓ . Specifically, for every node τ in level ℓ with children α_1 and α_2 , τ becomes a leaf node of the new HSS tree, with corresponding generators given by

$$\mathbf{D}_\tau \leftarrow \begin{bmatrix} \hat{\mathbf{D}}_{\alpha_1} & \mathbf{B}_{\alpha_1, \alpha_2} \\ \mathbf{B}_{\alpha_2, \alpha_1} & \hat{\mathbf{D}}_{\alpha_2} \end{bmatrix}. \quad (2.5)$$

Let $\boldsymbol{\Omega}^{(L)} = \boldsymbol{\Omega}$ and $\mathbf{Y}^{(L)} = \mathbf{Y}$. By Multiplying (2.4) with $\boldsymbol{\Omega}^{(\ell+1)}$ and rearranging the equation, we obtain

$$\mathbf{U}^{(\ell+1),*} (\mathbf{Y}^{(\ell+1)} - \check{\mathbf{D}}^{(\ell+1)} \boldsymbol{\Omega}^{(\ell+1)}) = \mathbf{A}^{(\ell)} (\mathbf{V}^{(\ell+1),*} \boldsymbol{\Omega}^{(\ell+1)}).$$

This can be interpreted as a randomized sampling of $\mathbf{A}^{(\ell)}$ using the new test and sampling matrices $\boldsymbol{\Omega}^{(\ell)}$ and $\mathbf{Y}^{(\ell)}$ from their blocks in level ℓ . More specifically, let τ be a node and α_1 and α_2 be its children. The matrices $\boldsymbol{\Omega}_\tau^{(\ell)}$ and $\mathbf{Y}_\tau^{(\ell)}$ are defined as

$$\boldsymbol{\Omega}_\tau^{(\ell)} = \begin{bmatrix} \mathbf{V}_{\alpha_1}^* \boldsymbol{\Omega}_{\alpha_1}^{(\ell+1)} \\ \mathbf{V}_{\alpha_2}^* \boldsymbol{\Omega}_{\alpha_2}^{(\ell+1)} \end{bmatrix} \quad \text{and} \quad \mathbf{Y}_\tau^{(\ell)} = \begin{bmatrix} \mathbf{U}_{\alpha_1}^* (\mathbf{Y}_{\alpha_1}^{(\ell+1)} - \check{\mathbf{D}}_{\alpha_1} \boldsymbol{\Omega}_{\alpha_1}^{(\ell+1)}) \\ \mathbf{U}_{\alpha_2}^* (\mathbf{Y}_{\alpha_2}^{(\ell+1)} - \check{\mathbf{D}}_{\alpha_2} \boldsymbol{\Omega}_{\alpha_2}^{(\ell+1)}) \end{bmatrix}.$$

Matrices $\boldsymbol{\Psi}_\tau^{(\ell)}$ and $\mathbf{Z}_\tau^{(\ell)}$ are defined analogously. By using formulas as before, we can compute $\check{\mathbf{D}}_\tau$, \mathbf{U}_τ and \mathbf{V}_τ . This procedure is iteratively repeated until reaching the root node. If τ is the root node, we directly solve $\mathbf{D}_\tau = \mathbf{Y}_\tau \boldsymbol{\Omega}_\tau^\dagger$.

Next, a top-down procedure is performed to construct matrices \mathbf{B} and \mathbf{D} . For a level $0 \leq \ell \leq L - 1$ and a node τ on level ℓ , the matrices $\mathbf{A}_\tau^{(\ell+1)}$, $\mathbf{U}_\tau^{(\ell+1)}$ and $\mathbf{V}_\tau^{(\ell+1)}$ are defined to be the submatrix of $\mathbf{A}^{(\ell+1)}$, $\mathbf{U}^{(\ell+1)}$ and $\mathbf{V}^{(\ell+1)}$ corresponding to the node τ respectively. By comparing the factorization (2.4) with (2.1), we find that the only difference is that, for a node τ in level ℓ , the matrix $\mathbf{B}_\tau^{(\ell)}$ in (2.1) has a structure in which the diagonal blocks corresponding to its children are zero. Therefore, the key idea here is to “push” the diagonal part into the its children blocks until we reach the leaf nodes.

For $0 \leq \ell \leq L - 1$, suppose τ is a nonleaf node on level ℓ with children α_1 and α_2 and

$$\mathbf{A}_\tau^{(\ell)} = \mathbf{D}_\tau = \begin{bmatrix} \mathbf{D}_{\tau;\alpha_1,\alpha_1} & \mathbf{D}_{\tau;\alpha_1,\alpha_2} \\ \mathbf{D}_{\tau;\alpha_2,\alpha_1} & \mathbf{D}_{\tau;\alpha_2,\alpha_2} \end{bmatrix},$$

in which the matrix \mathbf{D}_τ has been computed on the previous level. The corresponding block with respect to τ in the factorization (2.4) can be expressed as

$$\begin{aligned} & \mathbf{U}_\tau^{(\ell+1)} \mathbf{A}_\tau^{(\ell)} \mathbf{V}_\tau^{(\ell+1)*} + \check{\mathbf{D}}_\tau^{(\ell+1)} \\ &= \begin{bmatrix} \mathbf{U}_{\alpha_1} \\ \mathbf{U}_{\alpha_2} \end{bmatrix} \begin{bmatrix} \mathbf{D}_{\tau;\alpha_1,\alpha_1} & \mathbf{D}_{\tau;\alpha_1,\alpha_2} \\ \mathbf{D}_{\tau;\alpha_2,\alpha_1} & \mathbf{D}_{\tau;\alpha_2,\alpha_2} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{\alpha_1} \\ \mathbf{V}_{\alpha_2} \end{bmatrix}^* + \begin{bmatrix} \check{\mathbf{D}}_{\alpha_1} & \check{\mathbf{D}}_{\alpha_2} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{U}_{\alpha_1} \\ \mathbf{U}_{\alpha_2} \end{bmatrix} \begin{bmatrix} & \mathbf{D}_{\tau;\alpha_1,\alpha_2} \\ \mathbf{D}_{\tau;\alpha_2,\alpha_1} & \end{bmatrix} \begin{bmatrix} \mathbf{V}_{\alpha_1} \\ \mathbf{V}_{\alpha_2} \end{bmatrix}^* + \begin{bmatrix} \mathbf{D}_{\alpha_1} & \\ & \mathbf{D}_{\alpha_2} \end{bmatrix}, \end{aligned}$$

where $\mathbf{D}_\alpha = \check{\mathbf{D}}_\alpha + \mathbf{U}_\alpha \mathbf{D}_{\tau;\alpha,\alpha} \mathbf{V}_\alpha^*$. Note that the first part in the last equation no longer contributes to the diagonal part. When τ is a non-leaf node, the update of \mathbf{D}_τ is attributed to the updates of corresponding matrices \mathbf{B} and $\hat{\mathbf{D}}$ of its children, as seen in (2.5). When τ is a leaf node, we obtain the final form of \mathbf{D}_τ . This process is repeated at each node of level ℓ for $\ell = 0, \dots, L - 1$, ultimately constructing the HSS structure of \mathbf{A} .

The algorithm requires $\mathcal{O}(k)$ sample points and has a complexity of $\mathcal{O}(k^2 M + k T_{\text{mult}})$, where T_{mult} is the complexity of apply \mathbf{A} and \mathbf{A}^* to a vector. Therefore, it is particularly suitable for problems where fast computations of matrix-vector product are available.

2.3 A Superfast Least-Squares Solver for HSS Matrices

In this section, we review the superfast HSS least-squares solver proposed in [31, 33]. The problem we consider is defined as $\min_{\mathbf{u} \in \mathbb{C}^N} \|\mathbf{A}\mathbf{u} - \mathbf{f}\|$ where $\mathbf{A} \in \mathbb{R}^{M \times N}$ is an HSS matrix associated with an HSS tree whose maximum level is L . The key algorithm is the URV factorization of HSS matrices, which can be viewed as a generalized QR factorization or the transpose version of the ULV factorization [6, 34]. The URV factorization of \mathbf{A} has the form

$$\mathbf{A} = \mathbf{Z}^{(L)} \cdots \mathbf{Z}^{(1)} \mathbf{Z}^{(0)} \mathbf{T}^{(0)} \mathbf{T}^{(1)} \mathbf{W}^{(1)*} \cdots \mathbf{T}^{(L)} \mathbf{W}^{(L)*}, \quad (2.6)$$

where $\{\mathbf{Z}^{(\ell)}\}$, $\{\mathbf{W}^{(\ell)}\}$ are block diagonal unitary matrices, $\{\mathbf{T}^{(\ell)}\}$ are block upper-triangular matrices. The computation of the factorization usually costs $\mathcal{O}(k^2(M+N))$ operations, where k is the HSS rank. Since each matrix in (2.6) is easily inverted, \mathbf{A}^\dagger can be efficiently applied on a vector with $\mathcal{O}(k(M+N))$ operations.

2.3.1 Factorization

We give a brief description of the algorithm. For convenience we assume that all HSS blocks of \mathbf{A} have the exact rank k . The factorization begins from each leaf node. Suppose the number of rows and columns of a leaf node τ are m_τ and n_τ respectively. If $m_\tau \gg n_\tau$, a *size reduction* step is taken to decrease the row size of the problem. This introduces zeros into \mathbf{D}_τ and \mathbf{U}_τ through a QR decomposition

$$\frac{k}{m_\tau} \begin{bmatrix} \mathbf{U}_\tau & \mathbf{D}_\tau \end{bmatrix} = \Omega_\tau \begin{bmatrix} \check{\mathbf{U}}_\tau & \check{\mathbf{D}}_\tau \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} n_\tau & \\ & m_\tau - n_\tau - k \end{bmatrix},$$

where $\Omega_\tau \in \mathbb{C}^{m_\tau \times m_\tau}$ is unitray. The reduced row size, i.e., the number of rows of $\tilde{\mathbf{U}}_\tau$ and $\tilde{\mathbf{D}}_\tau$, is denoted by $\check{m}_\tau = n_\tau + k$. This strategy can also be applied on the nonleaf nodes if necessary. If no size reduction is adapted, we set Ω_τ to be the empty and let $\tilde{\mathbf{U}}_\tau = \mathbf{U}_\tau$, $\tilde{\mathbf{D}}_\tau = \mathbf{D}_\tau$ and $\check{m}_\tau = m_\tau$.

Next, we introduce zeros into $\mathbf{V}_\tau \in \mathbb{C}^{n_\tau \times k}$ by a reverse QR factorization

$$\mathbf{V}_\tau = \mathbf{Q}_\tau \begin{bmatrix} \mathbf{0} \\ \hat{\mathbf{V}}_{\tau;2} \end{bmatrix}, \quad (2.7)$$

where $\mathbf{Q}_\tau \in \mathbb{C}^{n_\tau \times n_\tau}$ is unitary and $\hat{\mathbf{V}}_{\tau;2} \in \mathbb{C}^{k \times k}$. By leveraging the shared basis property of the HSS matrix, multiplying each leaf block τ by \mathbf{Q}_τ on the right zeros out corresponding columns of the node τ . See Figure 2.1a for an illustration. The diagonal block is then modified by $\tilde{\mathbf{D}}_\tau = \tilde{\mathbf{D}}_\tau \mathbf{Q}_\tau$.

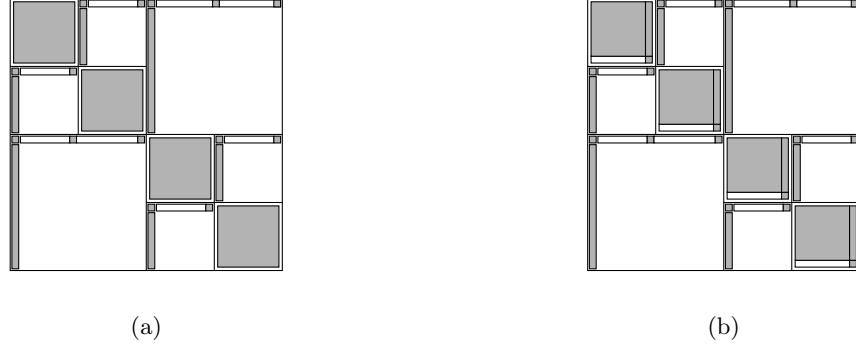


Figure 2.1: An illustration of the URV factorization. (A) Introducing zeros into off-diagonal columns. (B) QR factorization on diagonal blocks.

By partitioning the diagonal block as

$$\tilde{\mathbf{D}}_\tau = \begin{bmatrix} \tilde{\mathbf{D}}_{\tau;1,1} & \tilde{\mathbf{D}}_{\tau;1,2} \\ \tilde{\mathbf{D}}_{\tau;2,1} & \tilde{\mathbf{D}}_{\tau;2,2} \end{bmatrix},$$

our goal is to decouple the rows regarding to the nonzero block in (2.7). This can be done by a QR factorization of the first column block:

$$\begin{bmatrix} \tilde{\mathbf{D}}_{\tau;1,1} \\ \tilde{\mathbf{D}}_{\tau;2,1} \end{bmatrix} = \mathbf{P}_\tau \begin{bmatrix} \hat{\mathbf{D}}_{\tau;1,1} \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{P}_\tau \in \mathbb{C}^{\check{m}_\tau \times \check{m}_\tau}$ is unitary. After multiplying \mathbf{P}_τ^* on the left, the diagonal block transforms

$$\hat{\mathbf{D}}_\tau = \mathbf{P}_\tau^* \tilde{\mathbf{D}}_\tau = \begin{bmatrix} \hat{\mathbf{D}}_{\tau;1,1} & \hat{\mathbf{D}}_{\tau;1,2} \\ \mathbf{0} & \hat{\mathbf{D}}_{\tau;2,2} \end{bmatrix}$$

and the basis matrix \mathbf{U}_τ is modified to

$$\mathbf{P}_\tau^* \mathbf{U}_\tau = \begin{bmatrix} \hat{\mathbf{U}}_{\tau;1} \\ \hat{\mathbf{U}}_{\tau;2} \end{bmatrix}.$$

Figure 2.1b illustrates the results after the elimination on diagonal blocks.

Now, it is remarkable that if partitioning $\tau = \tau_1 \sqcup \tau_2$ conformably for all leaf nodes τ , the subproblem with respect to τ_2 is entirely decoupled from τ_1 . Moreover, for a nonleaf node τ with children α_1 and α_2 ,

by merging appropriate blocks and defining

$$\begin{aligned}\mathbf{D}_\tau &= \begin{bmatrix} \hat{\mathbf{D}}_{\alpha_1;2,2} & \hat{\mathbf{U}}_{\alpha_1;2} \mathbf{B}_{\alpha_1,\alpha_2} \hat{\mathbf{V}}_{\alpha_2;2} \\ \hat{\mathbf{U}}_{\alpha_2;2} \mathbf{B}_{\alpha_2,\alpha_1} \hat{\mathbf{V}}_{\alpha_1;2} & \hat{\mathbf{D}}_{\alpha_2;2,2} \end{bmatrix}, \\ \mathbf{U}_\tau &= \begin{bmatrix} \hat{\mathbf{U}}_{\alpha_1;2} \mathbf{R}_{\alpha_1} \\ \hat{\mathbf{U}}_{\alpha_2;2} \mathbf{R}_{\alpha_2} \end{bmatrix}, \quad \mathbf{V}_\tau = \begin{bmatrix} \hat{\mathbf{V}}_{\alpha_1;2} \mathbf{W}_{\alpha_1} \\ \hat{\mathbf{V}}_{\alpha_2;2} \mathbf{W}_{\alpha_2} \end{bmatrix},\end{aligned}\tag{2.8}$$

then the children α_1 and α_2 can be “eliminated” and τ becomes a new leaf with the associated generators defined in (2.8). After that, the remaining matrix still has the HSS structure but with a maximum level $L - 1$. This enables us to recursively factor the merged HSS matrix until the root node is the only remaining node. At that point, we only need to perform a QR decomposition on the root node τ :

$$\mathbf{D}_\tau = \mathbf{P}_\tau \begin{bmatrix} \hat{\mathbf{D}}_{\tau;1,1} \\ \mathbf{0} \end{bmatrix}.$$

2.3.2 Solution

Once the URV factorization computed, we are ready to solve the least-squares problem. The algorithm can be divided into two steps similar to the application of QR factorization. The first step applies unitary transforms to the right-hand side and the second step uses backward substitution to solve a sequence of hierarchical triangular systems.

The first step involves a bottom-up traversal of the HSS tree. Starting from each leaf node τ , let \mathbf{f}_τ denote the block of the right-hand side corresponding to the row index I_τ . If size reduction is applied to τ , we define

$$\check{\mathbf{f}}_\tau = \Omega_\tau^* \mathbf{f}_\tau = \begin{bmatrix} \check{\mathbf{f}}_{\tau;1} \\ \check{\mathbf{f}}_{\tau;2} \end{bmatrix}.$$

If there is no size reduction step, $\check{\mathbf{f}}_{\tau;1}$ is defined as \mathbf{f}_τ itself. Then we multiply \mathbf{P}_τ^* to $\check{\mathbf{f}}_{\tau;1}$:

$$\hat{\mathbf{f}}_\tau = \mathbf{P}_\tau^* \check{\mathbf{f}}_{\tau;1} = \begin{bmatrix} \hat{\mathbf{f}}_{\tau;1} \\ \hat{\mathbf{f}}_{\tau;2} \end{bmatrix}.$$

Next, we merge the vectors on each parent τ with children α_1 and α_2 . That is, we set

$$\mathbf{f}_\tau = \begin{bmatrix} \hat{\mathbf{f}}_{\alpha_1;2} \\ \hat{\mathbf{f}}_{\alpha_2;2} \end{bmatrix}.$$

We continue applying the same procedure as before until τ is the root node.

The second step is a top-down procedure. Beginning at the root τ , we solve the triangular system

$$\hat{\mathbf{D}}_{\tau;1,1} \mathbf{u}_\tau = \mathbf{f}_\tau.$$

For its children α_1 and α_2 , we partition the solution as

$$\mathbf{u}_\tau = \begin{bmatrix} \hat{\mathbf{u}}_{\alpha_1;2} \\ \hat{\mathbf{u}}_{\alpha_2;2} \end{bmatrix}$$

and compute $\hat{\mathbf{g}}_{\alpha_1} = \mathbf{B}_{\alpha_1,\alpha_2} \hat{\mathbf{V}}_{\alpha_2;2} \hat{\mathbf{u}}_{\alpha_2;2}$ and $\hat{\mathbf{g}}_{\alpha_2} = \mathbf{B}_{\alpha_2,\alpha_1} \hat{\mathbf{V}}_{\alpha_1;2} \hat{\mathbf{u}}_{\alpha_1;2}$. For a nonroot node τ , we solve the triangular system

$$\hat{\mathbf{D}}_{\tau;1,1} \hat{\mathbf{u}}_{\tau;1} = \hat{\mathbf{f}}_{\tau;1} - \hat{\mathbf{D}}_{\tau;1,2} \hat{\mathbf{u}}_{\tau;2} - \hat{\mathbf{U}}_{\tau;1} \hat{\mathbf{g}}_\tau$$

and define

$$\mathbf{u}_\tau = \mathbf{Q}_\tau \begin{bmatrix} \hat{\mathbf{u}}_{\tau;1} \\ \hat{\mathbf{u}}_{\tau;2} \end{bmatrix}.$$

If τ is a nonleaf node with children α_1 and α_2 , we partition \mathbf{u}_τ and compute $\hat{\mathbf{g}}_{\alpha_1} = \mathbf{B}_{\alpha_1,\alpha_2} \hat{\mathbf{V}}_{\alpha_2;2} \hat{\mathbf{u}}_{\alpha_2;2} + \mathbf{R}_{\alpha_1} \hat{\mathbf{g}}_\tau$ and $\hat{\mathbf{g}}_{\alpha_2} = \mathbf{B}_{\alpha_2,\alpha_1} \hat{\mathbf{V}}_{\alpha_1;2} \hat{\mathbf{u}}_{\alpha_1;2} + \mathbf{R}_{\alpha_2} \hat{\mathbf{g}}_\tau$. Repeating this process until we reach the leaf node, then the solution \mathbf{u} is obtained by collecting vectors \mathbf{u}_τ for all leaf nodes τ .

2.4 A Superfast Direct Solver of Type-II INUDFT Problem

The type-II NUDFT matrix \mathbf{A} , defined by $\mathbf{A}(j, k) = e^{2\pi i x_j k} = \gamma_j^k$ with $\gamma_j = e^{2\pi i x_j}$ on the unit circle \mathbb{S} , is a Vandermonde matrix and satisfies the Sylvester equation

$$\boldsymbol{\Gamma}\mathbf{A} - \mathbf{A}\mathbf{C} = \mathbf{a}\mathbf{e}_{N-1}^*, \quad (2.9)$$

where $\boldsymbol{\Gamma} = \text{diag}(\gamma_0, \gamma_1, \dots, \gamma_{M-1}) \in \mathbb{C}^{M \times M}$ is diagonal, $\mathbf{C} = [\mathbf{e}_1, \dots, \mathbf{e}_{N-1}, \mathbf{e}_0]$ is the shift-down circulant matrix and $a_j = \gamma_j^N - 1$. Here \mathbf{e}_k is the standard unit vector in \mathbb{C}^N with $\mathbf{e}_k(\ell) = \delta_{k,\ell}$. The circulant matrix \mathbf{C} can be diagonalized using the DFT matrix \mathbf{F} , expressed as $\mathbf{C} = \mathbf{F}^{-1} \mathbf{D}_{\mathbf{F}\mathbf{e}_1} \mathbf{F}$, where $\mathbf{D}_{\mathbf{F}\mathbf{e}_1} = \text{diag}(\mathbf{F}\mathbf{e}_1)$ is the diagonal matrix corresponding to the DFT of \mathbf{e}_1 . By multiplying \mathbf{F}^{-1} to the right of (2.9) and denoting $\tilde{\mathbf{A}} = \mathbf{A}\mathbf{F}^{-1}$, the equation becomes

$$\boldsymbol{\Gamma}\tilde{\mathbf{A}} - \tilde{\mathbf{A}}\mathbf{D}_{\mathbf{F}\mathbf{e}_1} = \mathbf{a}\mathbf{b}^*, \quad (2.10)$$

where $\mathbf{b} = \mathbf{F}^{-*}\mathbf{e}_{N-1}$.

In [31], the authors utilize the displacement structure (2.10) to establish the low-rank property of $\tilde{\mathbf{A}}$ and provide an estimate of the numerical rank [3]. Specifically, the ρ -rank of a matrix \mathbf{M} is defined as $\text{rank}_\rho(\mathbf{M}) := \min\{k : \|\mathbf{X} - \mathbf{M}\|_2 \leq \rho\|\mathbf{M}\|_2, \text{rank}(\mathbf{X}) = k\}$. It is shown that the ρ -rank of the HSS approximation $\tilde{\mathbf{A}}_{\text{HSS}}$ admits an upper bound of $k = \mathcal{O}(\log(1/\rho)\log N)$ (Theorem 3.2 in [31]). Additionally, by exploiting the displacement structure, the generators of the HSS matrix can be computed using the factorized alternating direction implicit (fADI) method [32], resulting in an overall complexity of $\mathcal{O}(k^2M)$ for the compression.

Here we present an alternative perspective on the low-rank property. Direct calculations show that the entries of $\tilde{\mathbf{A}}$ are explicitly given by

$$\tilde{\mathbf{A}}(j, k) = \frac{1}{N} \frac{(\gamma_j^N - 1)\zeta_N^k}{\gamma_j - \zeta_N^k}, \quad (2.11)$$

where $\zeta_N = e^{2\pi i/N}$. Consequently, $\tilde{\mathbf{A}}$ is a kernel matrix corresponding to kernel function

$$k(z, w) = \frac{1}{N} \frac{(z^N - 1)w}{z - w}.$$

evaluated at the points $\{z_j = \gamma_j\}$ and $\{w_k = \zeta_N^k\}$ for rows and columns. This kernel can be easily shown to possess the low-rank property by pole expansion. Therefore, $\tilde{\mathbf{A}}$ can be approximated by an HSS matrix $\tilde{\mathbf{A}}_{\text{HSS}}$. Using the same techniques in [16, 18], the HSS ρ -rank is bounded by $k = \mathcal{O}(\log(1/\rho)\log N)$, consistent with the above discussion. Moreover, the HSS compression can be obtained using *interpolative decomposition* (ID) [7, 14] together with the *proxy surface* technique [9, 18, 26, 27], resulting in $\mathcal{O}(k^2M)$ complexity, which is comparable with the fADI-based algorithm. With the HSS approximation $\tilde{\mathbf{A}}_{\text{HSS}}$ in hand, the least-squares solver in Section 2.3 is applied to it. This leads to the least-squares solver for type-II INUDFT problem.

To conclude, a fast structure in the type-II matrix is constructed as

$$\mathbf{A}_{\text{fast}} = \tilde{\mathbf{A}}_{\text{HSS}} \mathbf{F}, \quad (2.12)$$

$$\mathbf{A}_{\text{fast}}^\dagger = \mathbf{F}^{-1} \tilde{\mathbf{A}}_{\text{HSS}}^\dagger. \quad (2.13)$$

We summarize the whole procedure in Algorithm 1. The algorithm consists of three stages: Construction, factorization and solution. In the construction stage, the HSS compression of $\tilde{\mathbf{A}}$ is constructed by the fADI-based method with a complexity of $\mathcal{O}(k^2M)$ where k is the HSS rank of $\tilde{\mathbf{A}}_{\text{HSS}}$. In the factorization stage, the URV factorization of $\tilde{\mathbf{A}}_{\text{HSS}}$ is computed. Once the factorization is finished, the solution \mathbf{u} can be obtained using FFT and $\tilde{\mathbf{A}}_{\text{HSS}}^\dagger$. The complexities of the factorization and solution are $\mathcal{O}(k^2M)$ and $\mathcal{O}(kM + N \log N)$ respectively. Usually, $k = \mathcal{O}(\log(1/\rho)\log N)$ for some accuracy parameter ρ . Therefore, the proposed type-II INUDFT solver has the complexity of $\mathcal{O}(M \log^2 N)$ for construction and factorization and $\mathcal{O}(M \log N)$ for solution.

Algorithm 1: A direct solver for type-II INUDFT problem.

Input: Sample points $\{x_j\}_{j=0}^{M-1}$ and target values $\{f_j\}_{j=0}^{M-1}$, accuracy parameter $\rho \in (0, 1)$. Output: Coefficients $\{u_k\}_{k=0}^{N-1}$. <i>/* Construction.</i>	<i>*/</i>
1 Approximate $\tilde{\mathbf{A}} = \mathbf{A}\mathbf{F}^{-1}$ by an HSS matrix $\tilde{\mathbf{A}}_{\text{HSS}}$ using the fADI-based construction with accuracy parameter ρ ; <i>/* Factorization.</i>	<i>*/</i>
2 Compute the URV factorization of $\tilde{\mathbf{A}}_{\text{HSS}}^\dagger$; <i>/* Solution.</i>	<i>*/</i>
3 Solve $\mathbf{v} = \arg \min_{\mathbf{w}} \ \tilde{\mathbf{A}}_{\text{HSS}}\mathbf{w} - \mathbf{f}\ _2$; 4 Compute $\mathbf{u} = \mathbf{F}\mathbf{v}$ using FFT;	

3 A Superfast Direct Solver of Type-III INUDFT Problem

In this section, we propose a superfast direct solver for the type-III INUDFT problem. We begin by providing some theoretical results in Section 3.1, followed by a discussion of the detailed algorithm and its implementation in Section 3.2.

3.1 Theoretical Motivation

The square integrable functions on $[0, 1]$ form a Hilbert space $L^2[0, 1]$ with the inner product defined by $\langle f, g \rangle := \int_0^1 f(x)g(x)dx$. A typically used orthonormal basis of $L^2[0, 1]$ is given by $\{e^{2\pi i \ell x}\}_{\ell \in \mathbb{Z}}$. We begin by expanding the function $e^{2\pi i \omega_k x}$ in terms of this basis:

$$e^{2\pi i \omega_k x} = \sum_{\ell \in \mathbb{Z}} \langle e^{2\pi i \omega_k x}, e^{2\pi i \ell x} \rangle e^{2\pi i \ell x} = \sum_{\ell \in \mathbb{Z}} Q_{\ell, k} e^{2\pi i \ell x}, \quad (3.1)$$

where the coefficients are given by

$$Q_{\ell, k} = \int_0^1 e^{2\pi i (\omega_k - \ell)x} dx = e^{\pi i (\omega_k - \ell)} \frac{\sin(\pi(\omega_k - \ell))}{\pi(\omega_k - \ell)} = G(\omega_k - \ell). \quad (3.2)$$

The assembled matrix \mathbf{Q} can be viewed as a kernel matrix with the kernel function defined by

$$G(x) := e^{\pi i x} \frac{\sin(\pi x)}{\pi x}. \quad (3.3)$$

Since the kernel G is smooth and not highly oscillatory, \mathbf{Q} can be compressed into an HSS matrix. Moreover, as the function $G(x)$ decays when $|x|$ is large, truncating the series in (3.1) by considering the leading $2R$ terms leads to

$$e^{2\pi i \omega_k x} = \sum_{\ell \in \mathbb{Z}} Q_{\ell, k} e^{2\pi i \ell x} = \sum_{|\ell - \omega_k| \leq R} Q_{\ell, k} e^{2\pi i \ell x} + \sum_{|\ell - \omega_k| > R} Q_{\ell, k} e^{2\pi i \ell x},$$

where the first and second term in the right-hand side serve as the approximation and the error respectively. The L^2 norm of the error, denoted as $\text{err}_k(x)$, is given by Parseval's identity:

$$\|\text{err}_k\|_{L^2}^2 = \sum_{|\ell - \omega_k| > R} |Q_{\ell, k}|^2 = \sum_{|\ell - \omega_k| > R} \frac{\sin^2(\pi(\omega_k - \ell))}{\pi^2(\omega_k - \ell)^2}.$$

Let $\omega_k = p_k + \delta_k$, where p_k is the nearest integer of ω_k and $|\delta_k| \leq 1/2$. Then, when $R \geq 2$, we have the following estimation for the upper bound of $\|\text{err}_k\|_{L^2}^2$:

$$\begin{aligned}\|\text{err}_k\|_{L^2}^2 &\leq 2 \sum_{\ell=R}^{+\infty} \frac{\sin^2(\pi(\delta_k - \ell))}{\pi^2(\delta_k - \ell)^2} = 2 \sum_{\ell=R}^{+\infty} \frac{\sin^2(\pi\delta_k)}{\pi^2(\delta_k - \ell)^2} \leq \frac{2}{\pi^2} \sum_{\ell=R}^{+\infty} \frac{1}{(\ell - 1/2)^2} \\ &\leq \frac{2}{\pi^2} \sum_{\ell=R}^{+\infty} \int_{\ell-1}^{\ell} \frac{1}{(x - 1/2)^2} dx = \frac{2}{\pi^2} \int_{R-1}^{\infty} \frac{1}{(x - 1/2)^2} dx = \frac{2}{\pi^2} \frac{1}{R - 3/2}.\end{aligned}\quad (3.4)$$

Now consider the type-III NUDFT matrix. Each entry in the NUDFT matrix can be approximated as

$$\mathbf{A}(j, k) = e^{2\pi i x_j \omega_k} \approx \sum_{-\bar{R} \leq \ell \leq \bar{N}-1+R} e^{2\pi i x_j \ell} Q_{\ell, k}.$$

Denote $I_{N;R} = \{\ell \in \mathbb{Z} : -R \leq \ell \leq N-1+R\}$. The approximation can be reformulated in matrix form as

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{E}, \quad (3.5)$$

where

$$\mathbf{B} = (e^{2\pi i x_j \ell})_{j, \ell}, \quad \mathbf{Q} = (Q_{\ell, k})_{\ell, k}, \quad \text{and} \quad \mathbf{E} = (\sum_{\ell \notin I_{N;R}} Q_{\ell, k} e^{2\pi i \ell x_j})_{j, k}$$

are the corresponding type-II NUDFT matrix, coefficient matrix and error matrix respectively. To estimate the approximation error, we assume that the sample points $\{x_j\}$ are independent and identically distributed (i.i.d.) uniformly random variables on $[0, 1]$. The statistical properties of the random matrix \mathbf{E} can be obtained through direct calculations. Precisely, we have

$$\mathbb{E}\mathbf{E}(j, k) = \sum_{\ell \notin I_{N;R}} Q_{\ell, k} \int_0^1 e^{2\pi i \ell x} dx = 0$$

and

$$\begin{aligned}\mathbb{E}\|\mathbf{E}\|_{\text{F}}^2 &= \mathbb{E} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} \left| \sum_{\ell \notin I_{N;R}} Q_{\ell, k} e^{2\pi i \ell x_j} \right|^2 = M \mathbb{E} \sum_{k=0}^{N-1} \left| \sum_{\ell \notin I_{N;R}} Q_{\ell, k} e^{2\pi i \ell x_0} \right|^2 \\ &= M \int_0^1 \sum_{k=0}^{N-1} \left| \sum_{\ell \notin I_{N;R}} Q_{\ell, k} e^{2\pi i \ell x} \right|^2 dx = M \sum_{k=0}^{N-1} \sum_{\ell \notin I_{N;R}} |Q_{\ell, k}|^2 \\ &\leq MN \frac{2}{\pi^2} \frac{1}{R - 3/2},\end{aligned}$$

where we apply Parseval's identity and the estimate from (3.4). We summarize the results in the following proposition by noting that $\|\mathbf{A}\|_{\text{F}}^2 = MN$.

Proposition 3.1. Suppose $\mathbf{A} \in \mathbb{C}^{M \times N}$ is the type-III NUDFT matrix defined as $\mathbf{A}(j, k) = e^{2\pi i x_j \omega_k}$, where x_j are i.i.d. uniform random variables in $[0, 1]$ and $\omega_k \in [-1/2, N-1/2]$. Given $R \geq 2$, let \mathbf{B} and \mathbf{Q} be the matrices defined in (3.5). Then the error matrix \mathbf{E} satisfies $\mathbb{E}\mathbf{E} = \mathbf{0}$ and

$$\mathbb{E}\|\mathbf{E}\|_{\text{F}}^2 \leq 2/(\pi^2(R - 3/2)) \|\mathbf{A}\|_{\text{F}}^2.$$

3.2 Algorithm

While the approximation derived from (3.5) is straightforward, it is not the used form in our practical algorithm. One main reason is that the error of this approximation can be quite large. Our strategy is to replace the matrix \mathbf{Q} by the best candidate \mathbf{H} in the approximation $\|\mathbf{A} - \mathbf{B}\mathbf{H}\|_{\text{F}}$. That is, $\mathbf{H} = \mathbf{B}^\dagger \mathbf{Q}$. We

will later demonstrate that this leads to a reprojection of the error matrix \mathbf{E} against the type-II NUDFT matrix \mathbf{B} . In this section, we describe our practical superfast direct solver for type-III INUDFT problem. The overall algorithm consists of three stages: Construction, factorization and solution. We will discuss each stage in detail in the rest of this section and the overall pseudocode can be found in Algorithm 2.

Algorithm 2: A direct solver for type-III INUDFT problem.

Input:	Sample points $\{x_j\}_{j=0}^{M-1}$, frequencies $\{\omega_k\}_{k=0}^{N-1}$, target values $\{f_j\}_{j=0}^{M-1}$, accuracy parameter $\rho \in (0, 1)$ and rank parameter k .
Output:	Coefficients $\{u_k\}_{k=0}^{N-1}$.
/* Construction.	
1	Construct a superfast direct solver $\mathbf{B}_{\text{fast}} = \tilde{\mathbf{B}}_{\text{HSS}} \mathbf{F}$ and $\mathbf{B}_{\text{fast}}^\dagger = \mathbf{F}^{-1} \tilde{\mathbf{B}}_{\text{HSS}}^\dagger$ by Algorithm 1 for the type-II NUDFT matrix \mathbf{B} with accuracy parameter τ ;
2	Set $r = k + p$ ($p = 5$ or $p = 10$) and $s \geq \max(r + m, 3r)$ where m is the max leaf size in the HSS tree corresponding to \mathbf{H}_{HSS} ;
3	Generate $N \times s$ Gaussian random matrices Ω and Ψ ;
4	Compute $\mathbf{Y} = \mathbf{B}_{\text{fast}}^\dagger \mathbf{A} \Omega$ and $\mathbf{Z} = \mathbf{A}^* \mathbf{B}_{\text{fast}}^{\dagger,*} \Psi$ using the type-II NUDFT direct solver and NUFFT;
5	Compress $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$ into an HSS matrix \mathbf{H}_{HSS} using $\{\Omega, \mathbf{Y}, \Psi, \mathbf{Z}\}$;
/* Factorization.	
6	Compute the URV factorization of \mathbf{H}_{HSS} ;
/* Solution.	
7	Calculate $\mathbf{u} = \mathbf{H}_{\text{HSS}}^{-1} \mathbf{B}_{\text{fast}}^\dagger \mathbf{f}$;

3.2.1 Construction

Suppose the type-III NUDFT matrix $\mathbf{A} \in \mathbb{C}^{M \times N}$ is defined by $\mathbf{A}(j, k) = e^{2\pi i x_j \omega_k}$ for $x_j \in [0, 1]$ and $\omega_k \in [-1/2, N - 1/2]$. Assuming $R \geq 0$ and $N + 2R \leq M$, let $\mathbf{B} \in \mathbb{C}^{M \times (N+2R)}$ be the type-II NUDFT matrix given by $\mathbf{B}(j, \ell) = e^{2\pi i x_j \ell}$ for $-R \leq \ell \leq N - 1 + R$. Since \mathbf{B} is a Vandermonde matrix, it has full column rank and $\mathbf{B}^\dagger \mathbf{B} = \mathbf{I}$. Multiplying \mathbf{B}^\dagger on the left of (3.5) and denoting $\mathbf{H} = \mathbf{B}^\dagger \mathbf{A}$ gives

$$\mathbf{H} = \mathbf{Q} + \mathbf{B}^\dagger \mathbf{E}. \quad (3.6)$$

Moreover, by multiplying \mathbf{B} on the left and rearranging the equation, we have

$$\mathbf{A} = \mathbf{B}\mathbf{H} + (\mathbf{I} - \mathbf{B}\mathbf{B}^\dagger)\mathbf{E}. \quad (3.7)$$

The two equations (3.6) and (3.7) serve as the foundation of our algorithm. From the discussion in Section 3.1, \mathbf{Q} can be well approximated by an HSS matrix. Consequently, the matrix \mathbf{H} in (3.6), viewed as a perturbation of \mathbf{Q} , can also be empirically compressed into an HSS matrix. On the other hand, from (3.7), $\mathbf{B}\mathbf{H}$ can be regarded as an approximation of \mathbf{A} . Noting that $\mathbf{I} - \mathbf{B}\mathbf{B}^\dagger$ is a projection matrix, the following results are direct consequences based on the discussion in Section 3.1:

$$\mathbb{E}\|(\mathbf{I} - \mathbf{B}\mathbf{B}^\dagger)\mathbf{E}\|_F^2 \leq \mathbb{E}\|\mathbf{E}\|_F^2 \leq 2/(\pi^2(R - 3/2))\|\mathbf{A}\|_F^2. \quad (3.8)$$

It is worth mentioning that the upper bound in (3.8) is quite pessimistic. We will see in Section 4.1 that in practice, the approximation behaves much better numerically.

Therefore, we approximate the NUDFT matrix \mathbf{A} by the product of \mathbf{B} and \mathbf{H} , where \mathbf{B} is a type-II NUDFT matrix admitting a fast approximation structure given by (2.12), and \mathbf{H} can be approximated by

an HSS matrix. The construction of the fast structure associated with the type-II NUDFT matrix has been discussed in Section 2.4. Thus, the main task here is to find an efficient way to compress \mathbf{H} into its HSS form. This can be accomplished based on the following two observations:

- The matrix-vector multiplication $\mathbf{A}\mathbf{u}$ and $\mathbf{A}^*\mathbf{u}$ corresponds to the forward NUDFT computation, which can be performed very efficiently.
- \mathbf{B} is a type-II NUDFT matrix, thus, its pseudoinverse (as well as the conjugate transpose of the pseudoinverse) can be applied rapidly by the fast solver introduced in Section 2.4.

Therefore, by applying $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$ on random test matrices, the block-box algorithm in Section 2.2 is used to obtain an HSS approximation of \mathbf{H} . After construction, the fast structure in the type-III NUDFT problem can be expressed as

$$\mathbf{A}_{\text{fast}} = \mathbf{B}_{\text{fast}} \mathbf{H}_{\text{HSS}}. \quad (3.9)$$

3.2.2 Factorization and Solution

Once the HSS matrix \mathbf{H}_{HSS} has been prepared, we use the URV factorization to give an inversion of it. As a result, we obtain a direct solver

$$\mathbf{A}_{\text{fast}}^\dagger = \mathbf{H}_{\text{HSS}}^\dagger \mathbf{B}_{\text{fast}}^\dagger. \quad (3.10)$$

Using the fast structure (3.10), the solution of the least-squares problem is quite straightforward. We directly compute $\mathbf{u} = \mathbf{H}_{\text{HSS}}^{-1} \mathbf{B}_{\text{fast}}^\dagger \mathbf{f}$, which involves the solution processes for the type-II NUDFT matrix \mathbf{B} and the HSS matrix \mathbf{H} .

The fast structure (3.10) could also serve as a preconditioner of \mathbf{A}^{-1} . When \mathbf{A} is not too ill-conditioned, one can obtain a solution by applying PCG to the following normal equation:

$$\mathbf{A}^* \mathbf{A} \mathbf{u} = \mathbf{A}^* \mathbf{f}. \quad (3.11)$$

Because of the fast matrix-vector multiplication, each iteration of CG can be computed efficiently. In this circumstance, one can use $\mathbf{A}_{\text{fast}}^\dagger \mathbf{A}_{\text{fast}}^{\dagger*}$ as a preconditioner of system (3.11).

3.2.3 Complexity Estimate

We analyze the complexity of Algorithm 2 under the assumption that the HSS rank of both \mathbf{H}_{HSS} and $\tilde{\mathbf{H}}_{\text{HSS}}$ is k . The construction stage includes 3 steps: The first step involves constructing and factorizing \mathbf{B}_{fast} , which has a complexity of $\mathcal{O}(k^2 M)$. The second step applies \mathbf{A} and \mathbf{B}_{fast} on random test matrices with $\mathcal{O}(k)$ columns, which costs $\mathcal{O}(k^2 M + kN \log N)$. The final step recovers \mathbf{H}_{HSS} using the test matrices and sample matrices, with a complexity of $\mathcal{O}(k^2 N)$. Therefore, the total complexity of the construction stage is $\mathcal{O}(k^2 M + kN \log N)$. The factorization stage involves computing the URV factorization of \mathbf{H}_{HSS} , which has a cost of $\mathcal{O}(k^2 N)$. The solution stage includes the applications of $\mathbf{B}_{\text{fast}}^\dagger$ and $\mathbf{H}_{\text{HSS}}^\dagger$, with complexities of $\mathcal{O}(kM + N \log N)$ and $\mathcal{O}(kN)$ respectively. Hence, the total complexity for solution is $\mathcal{O}(kM + N \log N)$. If $k = \mathcal{O}(\log N)$, then the complexities of construction, factorization and solution of the proposed type-III INUDFT direct solver are $\mathcal{O}(M \log^2 N)$, $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(M \log N)$ respectively.

4 Numerical Results

In this section, we conduct several tests on our proposed algorithm. We begin with the discussion on the approximation (3.9) and then use examples to illustrate the efficiency of our solver. Throughout all experiments, two cases of type-III NUDFT problem are considered:

- Perturbation-Perturbation: Sample points perturbed from uniform points in $[0, 1]$ and frequencies are perturbed from integers. To be more specific, they are expressed as $x_j = (j + \beta\phi_j)/M$ and $\omega_k = k + \alpha\psi_k$ where each ϕ_j and ψ_k is drawn from a uniform distribution on $[-1, 1]$ and $0 \leq \beta, \alpha < 1/2$ control the non-uniformity.

- Random-Perturbation: Sample points are i.i.d. uniform random variables $[0, 1]$ and the frequencies are perturbed integers given by $\omega_k = k + \alpha\psi_k$.

All algorithms are implemented in MATLAB R2023b where MATLAB's FFT library and the FINUFFT library [2] are utilized for all corresponding fast matrix-vector multiplications. All experiments are carried out on a server with an Intel Gold 6226R CPU at 2.90 GHz and 1000.6 GB of RAM.

We briefly explain our experimental settings. The number of sample points and frequencies is denoted by M and N , respectively, with M consistently set to $4N$ in all examples. Unless specified, the number of additional terms R is always set to be 0. The leaf size of each HSS matrix is fixed at 128, and the HSS rank of \mathbf{H}_{HSS} in (3.9), is set to be approximately $5 \log N$ for its black-box construction. We use ρ to denote the accuracy parameter in the fADI construction of $\tilde{\mathbf{B}}_{\text{HSS}}$, which is set to 10^{-12} when using (3.10) as a direct solver or 10^{-7} when it is applied as a preconditioner. Additionally, β and α represent the perturbation size on sample points and frequencies, respectively.

The following values are taken into consideration when evaluating the algorithm: We use t_c , t_f and t_s to denote the time costs in the construction, factorization and solution stage of the type-III solver. For each solution, its relative residual, defined as $\|\mathbf{A}\mathbf{u} - \mathbf{f}\|/\|\mathbf{f}\|$, is denoted by r_s . For iterative methods, we use t_{pre} and t_{iter} to denote the time cost in the preparation stage and iteration stage. More specifically, t_{pre} includes the time for both construction and factorization, while t_{iter} only includes the iteration time of CG or PCG. Additionally, n_{iter} is the number of iterations to solve (3.11) using CG or PCG to a tolerance of 10^{-12} , with a maximum number of iterations set to be 500.

4.1 Exploring Properties of the Type-III NUDFT

We verify the approximation error (3.8) in Proposition 3.1, by approximating a 4096×1024 type-III NUDFT matrix \mathbf{A} under the Random-Perturbation case using the fast structure in (3.9). The approximation error is defined as

$$\frac{\|\mathbf{A} - \mathbf{A}_{\text{fast}}\|_{\text{F}}}{\|\mathbf{A}\|_{\text{F}}},$$

where the Frobenius norm is estimated by random sampling on 30 samples [28]. The parameter α is chosen to be 10^{-7} , 10^{-4} , 0.1 and 0.4, while R is set to be 0, 16, 32, 64 and 128. The relative error is computed through an average on 5 simulations. Figure 4.1 shows the error for different values of R and α . It can

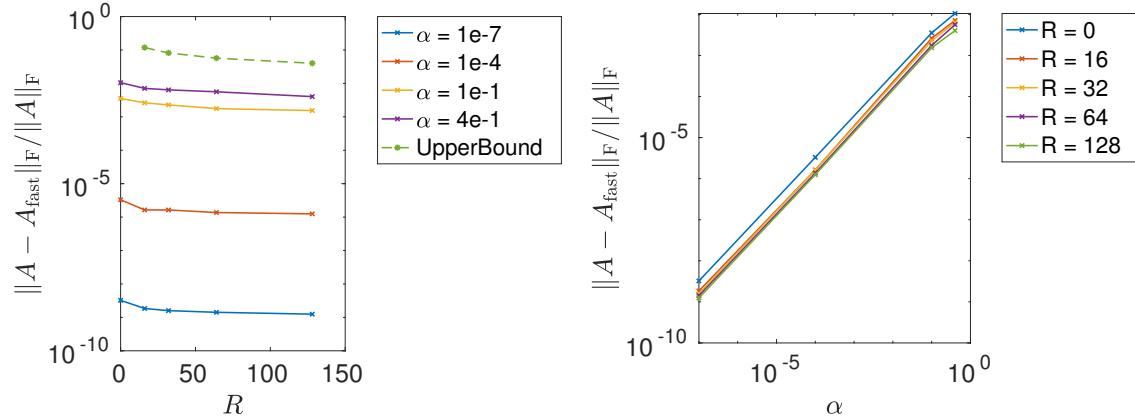


Figure 4.1: Approximation error of a type-III NUDFT matrix of size 4096×1024 for various R and α . Left: Approximation error with respect to R . Right: Approximation error with respect to α . The upper bound is given by $\sqrt{2}/(\pi\sqrt{R - 3/2})$ (see(3.8)).

be deduced that our theoretical upper bound is quite pessimistic since it lies beyond all other curves in the figure. Additionally, as the perturbation size α increases, the approximation error also increases, which aligns with our intuitions. Conversely, increasing R offers minimal benefits for the approximation, and simply setting $R = 0$ yields satisfactory results.

4.2 Perturbation-Perturbation Case

In the Perturbation-Perturbation case, both sample points and frequencies are perturbed from uniform points. In [36], the authors demonstrate that the condition number of a tall-and-skinny type-II NUDFT matrix with perturbed sample points can be bounded by a constant that depends only on the perturbation size β and the ratio M/N . Although there are no directly related generalizations for the type-III case, this finding provides useful insights. When the perturbation α of frequencies is small, we expect similar behaviors of the NUDFT matrix.

In our experiments, β is fixed to be 0.4 and α ranges in $[10^{-7}, 10^{-4}, 0.1, 0.4]$. The number of frequencies is set to be $N = 2^n$ for $n = 14 : 20$ and the target values \mathbf{f} is generated by $\mathbf{A}\mathbf{u}$ where \mathbf{u} is some random complex vector.

Figure 4.2 demonstrates time costs of the construction, factorization and solution stage for different N , as well as the relative residuals. Since different perturbations only have a mild influence on our algorithm, the four solid curves exhibit similar performance trends. Besides, it is evident that our algorithm scales as predicted. The construction and factorization follow $\mathcal{O}(N \log^2 N)$ complexity and the solution is nearly $\mathcal{O}(N \log N)$ (recall $M = 4N$). In all cases, the construction time t_c is greater than that of any other components, while the solution time t_s occupies the smallest proportion. To be more specific, when $N = 2^{20} = 1048576$, the construction costs about 1500 seconds, nearly 20 times of the factorization. On the other hand, we observe that the relative residual r_s increases mildly as the problem size becomes large. This is because as N increases, the accuracy of the type-II INUDFT least-squares solver decreases.

We discuss the construction stage in more detail. It involves 3 steps: The construction of \mathbf{B}_{fast} , the URV factorization of $\tilde{\mathbf{B}}_{\text{HSS}}$ and the black-box compression of \mathbf{H}_{HSS} . The latter includes applying the URV factorization of $\tilde{\mathbf{B}}_{\text{HSS}}$ on $\mathcal{O}(\log N)$ vectors and other further compressions. Figure 4.3a shows the proportion of different components in the construction stage when $\alpha = 0.4$ and $\beta = 0.1$. It is quite straightforward that the black-box compression takes the most time, accounting for nearly 80% of the total time when N is moderately large. In contrast, the fADI-construction and URV factorization on the type-II NUDFT matrix \mathbf{B} contribute only 10%-20%.

We also explore the behavior our algorithm as a preconditioner of CG. Corresponding results are summarized in Table 4.1 and 4.2. First, we point out that the number of iterations in PCG is always less than 10 steps, indicating that (3.10) serves as a good preconditioner. In contrast, CG takes at most 20 steps to converge when $\alpha = 10^{-7}$, $\alpha = 10^{-4}$ and $\alpha = 0.1$, but nearly 100 steps when $\alpha = 0.4$. For both methods, more iterations are need to converge as the perturbation size α increases. Our PCG method is stable because it is almost not affected by the problem size. When $\alpha = 0.4$, it can be found that CG needs more iterations when N becomes large.

However, it is still recommended to use CG on the normal equation directly, without any preconditioner. There are several reasons for this: First, PCG needs a preparation stage, which can be very time-consuming. Second, in CG, each iteration only involves a NUFFT procedure, whereas PCG requires additional operations. Comparing the iteration stages when $N = 1048576$ and $\alpha = 0.4$, we found that one step of CG costs only 12 seconds, while one step of PCG cost 30 seconds. Thus, although PCG takes less steps, the overall time in the iteration stage is greater. The third reason is that, the NUDFT matrix under the Perturbation-Perturbation case is usually well-conditioned, especially when α is small. Consequently, CG converges quickly and does not require many iterations.

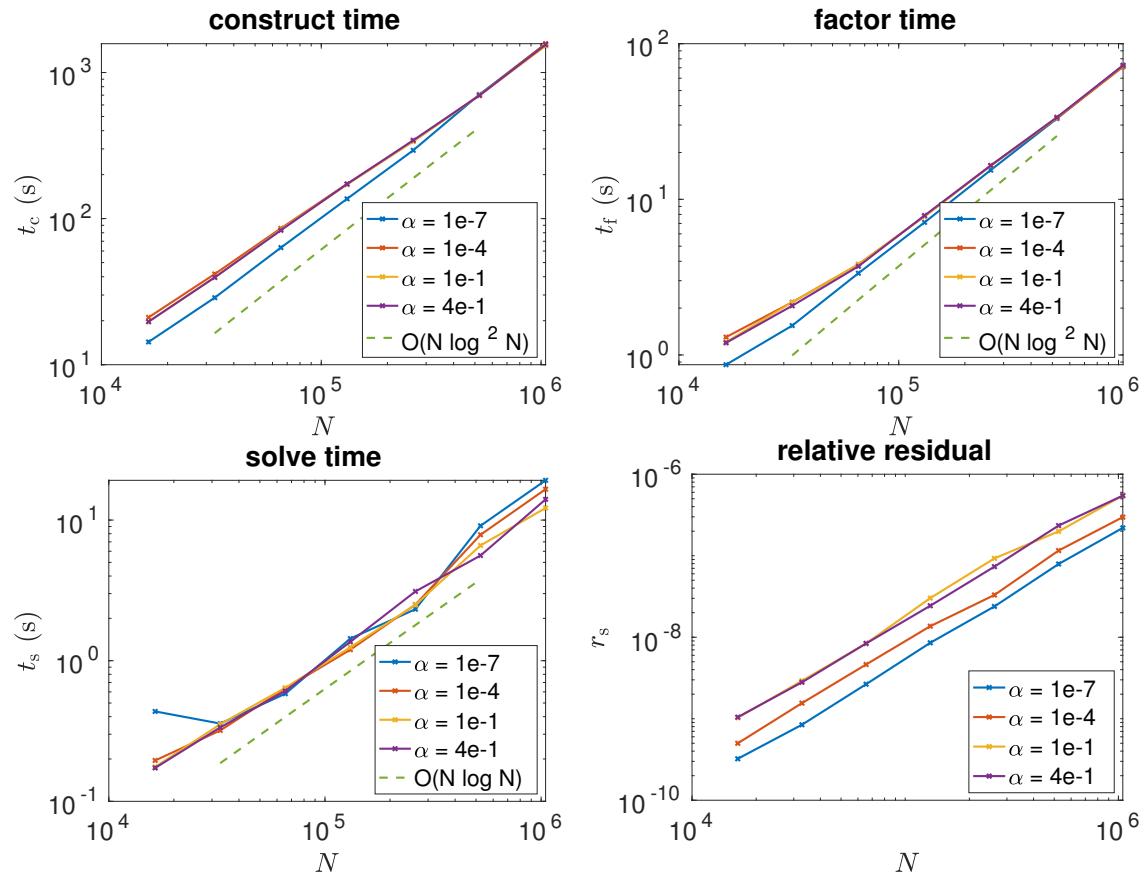


Figure 4.2: CPU times on different stages and relative residual of the proposed type-III INUDFT direct solver under the Perturbation-Perturbation case. The perturbation size is chosen to be 10^{-7} , 10^{-4} , 0.1 and 0.4. The green dotted lines represent reference complexity of corresponding stages.

N	method	$\alpha = 10^{-7}$				$\alpha = 10^{-4}$			
		t_{pre}	t_{iter}	n_{iter}	r_s	t_{pre}	t_{iter}	n_{iter}	r_s
16384	CG	-	3.8e-01	12	3.8e-13	-	1.7e-01	12	4.3e-13
	PCG	1.4e+01	1.3e+00	2	6.2e-15	1.9e+01	7.9e-01	2	1.2e-14
32768	CG	-	3.6e-01	12	3.9e-13	-	3.4e-01	12	5.3e-13
	PCG	2.9e+01	2.0e+00	2	1.4e-14	3.8e+01	1.4e+00	2	2.5e-14
65536	CG	-	7.2e-01	12	4.9e-13	-	7.8e-01	12	6.7e-13
	PCG	6.4e+01	4.0e+00	2	2.7e-14	7.9e+01	2.8e+00	2	5.9e-14
131072	CG	-	1.5e+00	12	6.5e-13	-	1.3e+00	12	6.4e-13
	PCG	1.4e+02	5.4e+00	2	6.9e-14	1.6e+02	5.7e+00	2	1.4e-13
262144	CG	-	2.8e+00	12	9.1e-13	-	2.6e+00	12	5.2e-13
	PCG	3.1e+02	1.6e+01	2	2.1e-13	3.3e+02	1.3e+01	2	3.4e-13
524288	CG	-	6.1e+00	12	6.5e-13	-	6.4e+00	12	5.8e-13
	PCG	6.8e+02	3.4e+01	2	9.8e-13	6.8e+02	4.0e+01	3	1.4e-15
1048576	CG	-	1.4e+01	12	8.2e-13	-	1.4e+01	12	8.0e-13
	PCG	1.5e+03	8.8e+01	3	1.4e-15	1.5e+03	7.1e+01	3	1.5e-15

Table 4.1: Time cost, iteration number and relative residual of CG and PCG under the Perturbation-Perturbation case, $\alpha = 10^{-7}$ and 10^{-4} .

N	method	$\alpha = 10^{-1}$				$\alpha = 4 \times 10^{-1}$			
		t_{pre}	t_{iter}	n_{iter}	r_s	t_{pre}	t_{iter}	n_{iter}	r_s
16384	CG	-	2.2e-01	17	4.4e-13	-	1.0e+00	90	8.7e-13
	PCG	1.8e+01	1.8e+00	5	2.1e-13	1.9e+01	3.2e+00	9	2.4e-15
32768	CG	-	4.2e-01	17	6.9e-13	-	2.2e+00	98	9.5e-13
	PCG	3.7e+01	3.3e+00	5	1.9e-13	3.8e+01	5.4e+00	8	1.8e-13
65536	CG	-	8.4e-01	17	5.6e-13	-	4.2e+00	95	8.5e-13
	PCG	7.8e+01	6.6e+00	5	2.1e-13	7.8e+01	1.1e+01	8	1.4e-13
131072	CG	-	1.7e+00	17	6.5e-13	-	8.0e+00	91	8.7e-13
	PCG	1.6e+02	1.3e+01	5	1.0e-13	1.7e+02	2.1e+01	8	4.6e-13
262144	CG	-	3.4e+00	17	7.6e-13	-	1.8e+01	100	9.1e-13
	PCG	3.3e+02	2.7e+01	5	1.8e-13	3.3e+02	4.4e+01	8	9.4e-13
524288	CG	-	8.5e+00	17	7.0e-13	-	3.4e+01	101	9.7e-13
	PCG	6.9e+02	5.4e+01	5	6.0e-14	6.9e+02	9.9e+01	9	2.8e-14
1048576	CG	-	1.9e+01	17	7.0e-13	-	1.1e+02	110	8.5e-13
	PCG	1.5e+03	1.3e+02	5	9.3e-14	1.5e+03	2.0e+02	8	8.1e-13

Table 4.2: Time cost, iteration number and relative residual of CG and PCG under Perturbation-Perturbation case, $\alpha = 0.1$ and 0.4 .

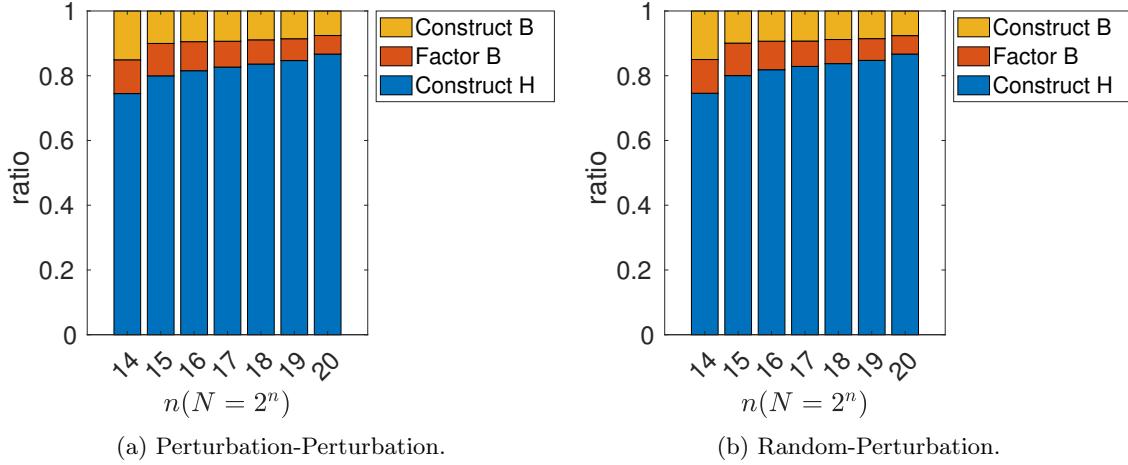


Figure 4.3: The proportion of different parts in the construction stage. The yellow, red and blue parts represent the fADI-construction of \mathbf{B}_{fast} , the URV factorization of $\tilde{\mathbf{B}}_{\text{HSS}}$ and the black-box compression of \mathbf{H}_{HSS} respectively. Left: Perturbation-Perturbation case. Right: Random-Perturbation case. Under both cases the perturbation of frequencies $\alpha = 0.4$ and $\beta = 0.4$. The column size $N = 2^n$.

4.3 Random-Perturbation Case

For the Random-Perturbation case, we did not find any literatures about the properties of the NUDFT matrix. For the type-II case when sample points are i.i.d. random variables, the paper [19] contains a study of the conditioning of the NUDFT matrix, showing that it is well-conditioned with high probability when $M \gg N \log N$. Although our choice of M and N does not meet the requirement they proposed, it still provides valuable insights for small perturbations. The settings of our test is similar: α takes values in $[10^{-7}, 10^{-4}, 0.1, 0.4]$, $N = 2^n$ for $n = 14 : 20$ and each \mathbf{f} is generated by the NUDFT of a random complex vector \mathbf{u} .

The time costs for the three stages and the relative residuals are shown in Figure 4.4, while Figure 4.3b displays the proportions of different components in the construction stage when $\alpha = 0.4$. Overall, since our algorithm is not highly sensitive to the distribution of the sample points, the runtime under the Perturbation-Perturbation case and the Random-Perturbation case remains nearly unchanged, aligning with the expected quasi-linear complexities. In the construction stage, the most expensive component is the black-box compression, accounting for about 80% of the total time, significantly outweighing the costs of the other two components. Additionally, the relative residual exhibits similar behavior as in the Perturbation-Perturbation case.

Considering the iterative methods, the situation differs significantly from the Perturbation-Perturbation case. Table 4.3 and 4.4 present the results for CG and PCG. It is clear that PCG consistently converges in 10 times. However, CG typically requires more than 200 iterations to converge, and the number of iterations increases dramatically when N becomes large. Even more concerning, in the case of moderately large N , CG may not converge at all. The minimum value of N for experiencing non-convergence in our tests is 262144 for $\alpha = 10^{-7}$ and $\alpha = 10^{-4}$, while it is 65536 for $\alpha = 0.4$ and $\alpha = 0.1$. These thresholds represent relatively small sizes for practical problems. Notably, the convergence and number of iterations for PCG are largely unaffected by the perturbation size and problem size, which underscores the efficiency of our preconditioner.

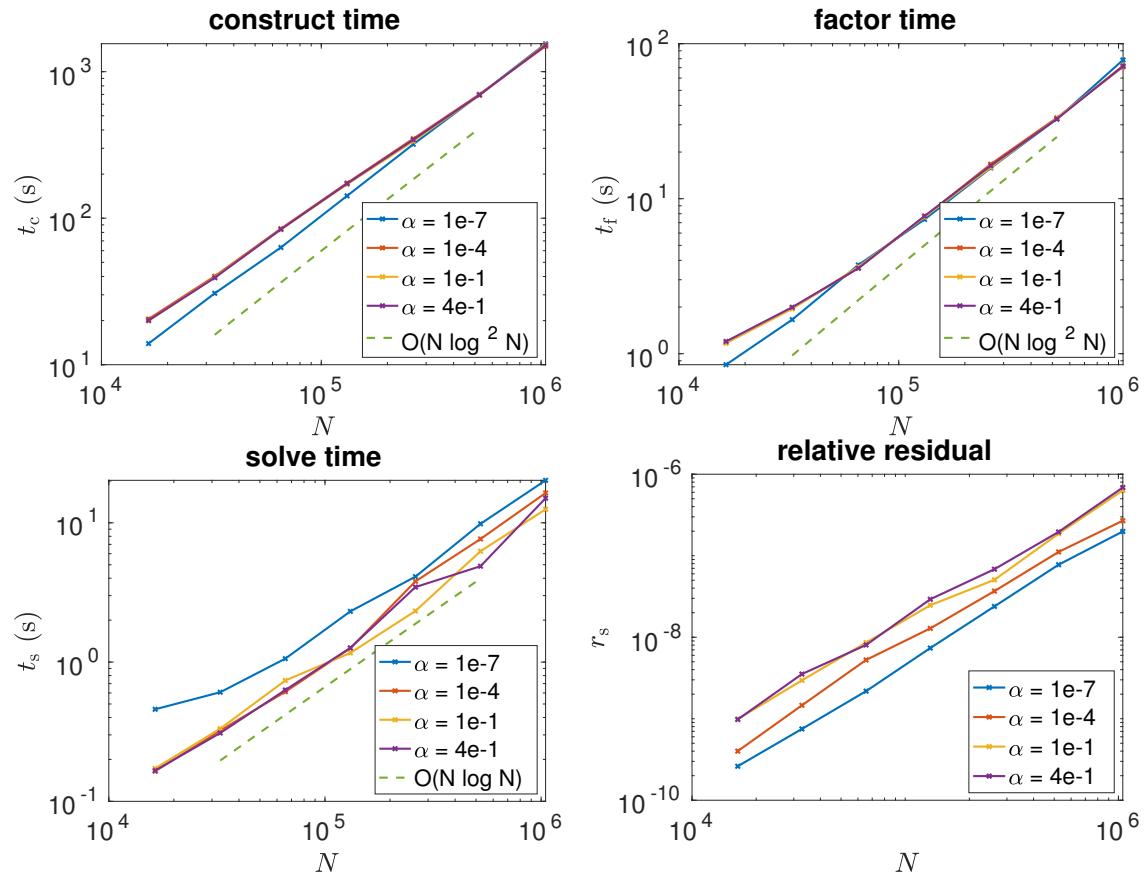


Figure 4.4: CPU times on different stages and relative residual of the proposed type-III INUDFT direct solver under Random-Perturbation case. The perturbation size is chosen to be $10^{-7}, 10^{-4}, 0.1$ and 0.4 .

N	method	t_{pre}	$\alpha = 10^{-7}$				$\alpha = 10^{-4}$			
			t_{iter}	n_{iter}	r_s	t_{pre}	t_{iter}	n_{iter}	r_s	
16384	CG	-	3.9e+00	326	9.9e-13	-	2.5e+00	228	9.4e-13	
	PCG	1.4e+01	1.7e+00	2	3.6e-14	1.9e+01	8.0e-01	2	6.3e-14	
32768	CG	-	7.4e+00	330	9.9e-13	-	6.9e+00	330	9.9e-13	
	PCG	2.8e+01	2.0e+00	2	7.5e-14	3.9e+01	1.6e+00	2	1.1e-13	
65536	CG	-	1.5e+01	354	9.6e-13	-	1.9e+01	441	9.8e-13	
	PCG	6.6e+01	3.3e+00	2	1.3e-13	8.2e+01	2.9e+00	2	3.4e-13	
131072	CG	-	4.1e+01	492	9.9e-13	-	\times	\times	\times	
	PCG	1.4e+02	5.7e+00	2	2.7e-13	1.7e+02	5.7e+00	2	7.7e-13	
262144	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	3.1e+02	1.4e+01	2	9.2e-13	3.4e+02	1.8e+01	3	1.4e-15	
524288	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	6.9e+02	4.4e+01	3	1.4e-15	7.2e+02	3.6e+01	3	1.4e-15	
1048576	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	1.6e+03	9.7e+01	3	2.6e-15	1.5e+03	9.4e+01	3	1.6e-15	

Table 4.3: Time cost, iteration number and relative residual of CG and PCG under the Random-Perturbation case, $\alpha = 10^{-7}$ and 10^{-4} . The marker “ \times ” means iterative method does not converge in 500 steps.

N	method	t_{pre}	$\alpha = 10^{-1}$				$\alpha = 4 \times 10^{-1}$			
			t_{iter}	n_{iter}	r_s	t_{pre}	t_{iter}	n_{iter}	r_s	
16384	CG	-	2.8e+00	248	9.3e-13	-	\times	\times	\times	
	PCG	1.9e+01	1.9e+00	5	2.0e-13	2.0e+01	3.3e+00	9	2.4e-15	
32768	CG	-	7.0e+00	324	9.4e-13	-	1.0e+01	483	9.9e-13	
	PCG	3.8e+01	3.5e+00	5	6.1e-13	3.9e+01	5.6e+00	8	1.1e-13	
65536	CG	-	\times	\times	\times	-	1.9e+01	442	9.7e-13	
	PCG	8.0e+01	6.9e+00	5	1.2e-14	8.1e+01	1.1e+01	8	5.0e-15	
131072	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	1.7e+02	1.4e+01	5	4.6e-13	1.7e+02	2.4e+01	9	1.0e-14	
262144	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	3.4e+02	3.0e+01	5	7.4e-14	3.4e+02	4.6e+01	8	7.1e-15	
524288	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	7.0e+02	5.6e+01	5	1.5e-13	7.0e+02	1.0e+02	9	4.1e-14	
1048576	CG	-	\times	\times	\times	-	\times	\times	\times	
	PCG	1.5e+03	1.4e+02	5	5.6e-14	1.5e+03	2.1e+02	8	7.7e-13	

Table 4.4: Time cost, iteration number and relative residual of CG and PCG under the Random-Perturbation case, $\alpha = 0.1$ and 0.4 . The marker “ \times ” means iterative method does not converge in 500 steps.

5 Conclusions and Future Work

In this paper, we introduce a superfast direct solver for the least-squares problem of type-III INUDFT problem. Our method combines the efficient type-II INUDFT direct solver [31] and fast algorithms of HSS matrices [22, 33]. By approximating the type-III NUDFT matrix by a product of a NUDFT-II matrix and an HSS matrix, we develop a fast inversion algorithm, which can serve as a direct solver or an efficient preconditioner for iterative methods. We give some theoretical results under suitable assumptions of the sample points and frequencies. Numerical experiments are carried out to explore the properties of our fast structure and to demonstrate the efficiency of the proposed method.

Some important directions for future research include:

- Theoretical analysis of the type-III NUDFT problem such as the properties of the NUDFT matrix. This would enable a much deeper understanding of the approximation. Current works mostly focus on the type-II case and lack the discussion of type-III case.
- Investigate a method on the regularized least-squares problem. In many circumstance (such as when sample points and frequencies are both uniform random variables), the NUDFT matrix is ill-conditioned or even rank-deficient. Regularization techniques need to be added into the least-squares problem and in this case, our method cannot be applied directly. The question of whether the regularized problem can be efficiently solved remains an interesting topic.
- Extension to high dimensional cases. The challenges of type-III INUDFT problem in high dimensions are similar to type-II INUDFT. For 2D and higher dimension, the displacement structure (2.10) of \tilde{A} does not hold anymore. In this case one can still derive some low-rank properties by the kernel function expression (2.11). However, it can not be compressed into an HSS matrix since many blocks are not numerically low-rank. Designing efficient methods to solve high dimensional INUDFT problems remains an open field.

References

- [1] Chris Anderson and Marie Dillon Dahleh, *Rapid computation of the discrete Fourier transform*, SIAM Journal on Scientific Computing **17** (1996), no. 4, 913–919.
- [2] Alexander H. Barnett, Jeremy Magland, and Ludvig af Klinteberg, *A parallel nonuniform fast fourier transform library based on an “exponential of semicircle” kernel*, SIAM Journal on Scientific Computing **41** (2019), no. 5, C479–C504.
- [3] Bernhard Beckermann and Alex Townsend, *On the singular values of matrices with displacement structure*, SIAM Journal on Matrix Analysis and Applications **38** (2017), no. 4, 1227–1248.
- [4] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch, *Introduction to hierarchical matrices with applications*, Engineering Analysis with Boundary Elements **27** (2003), no. 5, 405–422.
- [5] Emmanuel Candès, Laurent Demanet, and Lexing Ying, *A fast butterfly algorithm for the computation of Fourier integral operators*, Multiscale Modeling & Simulation **7** (2009), no. 4, 1727–1750.
- [6] Shvikumar Chandrasekaran, Ming Gu, and Timothy Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM Journal on Matrix Analysis and Applications **28** (2006), no. 3, 603–622.
- [7] Hongwei Cheng, Zydrunas Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin, *On the compression of low rank matrices*, SIAM Journal on Scientific Computing **26** (2005), no. 4, 1389–1404.
- [8] James W. Cooley and John W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation **19** (1965), no. 90, 297–301.
- [9] Eduardo Corona, Per-Gunnar Martinsson, and Denis Zorin, *An $O(N)$ direct solver for integral equations on the plane*, Applied and Computational Harmonic Analysis **38** (2015), no. 2, 284–317.

- [10] Alok Dutt and Vladimir Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM Journal on Scientific Computing **14** (1993), no. 6, 1368–1393.
- [11] ———, *Fast Fourier transforms for nonequispaced data, II*, Applied and Computational Harmonic Analysis **2** (1995), no. 1, 85–100.
- [12] Hans G. Feichtinger, Karlheinz Gröchenig, and Thomas Strohmer, *Efficient numerical methods in non-uniform sampling theory*, Numerische Mathematik **69** (1995), no. 4, 423–440.
- [13] Leslie Greengard and June-Yub Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Review **46** (2004), no. 3, 443–454.
- [14] Ming Gu and Stanley C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM Journal on Scientific Computing **17** (1996), no. 4, 848–869.
- [15] Wolfgang Hackbusch, *A sparse matrix arithmetic based on \mathcal{H} -matrices. Part I: Introduction to \mathcal{H} -matrices*, Computing **62** (1999), no. 2, 89–108.
- [16] Wolfgang Hackbusch and Boris N. Khoromskij, *A sparse \mathcal{H} -matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing **64** (2000), no. 1, 21–47.
- [17] Wolfgang Hackbusch, Boris N. Khoromskij, and Stefan A. Sauter, *On \mathcal{H}^2 -matrices*, Lectures on Applied Mathematics, 2000, pp. 9–29.
- [18] Kenneth L. Ho and Leslie Greengard, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM Journal on Scientific Computing **34** (2012), no. 5, A2507–A2532.
- [19] Lutz Kämmerer, Tino Ullrich, and Toni Volkmer, *Worst-case recovery guarantees for least squares approximation using random samples*, Constructive Approximation **54** (2021), no. 2, 295–352.
- [20] Melanie Kircheis and Daniel Potts, *Direct inversion of the nonequispaced fast Fourier transform*, Linear Algebra and its Applications **575** (2019), 106–140.
- [21] June-Yub Lee and Leslie Greengard, *The type 3 nonuniform FFT and its applications*, Journal of Computational Physics **206** (2005), no. 1, 1–5.
- [22] James Levitt and Per-Gunnar Martinsson, *Linear-complexity black-box randomized compression of rank-structured matrices*, SIAM Journal on Scientific Computing **46** (2024), no. 3, A1747–A1763.
- [23] Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying, *Butterfly factorization*, Multiscale Modeling & Simulation **13** (2015), no. 2, 714–732.
- [24] Lin Lin, Jianfeng Lu, and Lexing Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, Journal of Computational Physics **230** (2011), no. 10, 4071–4087.
- [25] P. G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM J. Matrix Anal. Appl. **32** (2011), no. 4, 1251–1274.
- [26] Per-Gunnar Martinsson, *Fast direct solvers for elliptic pdes*, Fast direct solvers for elliptic partial differential equations, Society for Industrial Applied Mathematics, 2019.
- [27] Per-Gunnar Martinsson and Vladimir Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, Journal of Computational Physics **205** (2005), no. 1, 1–23.
- [28] Per-Gunnar Martinsson and Joel A. Tropp, *Randomized numerical linear algebra: Foundations and algorithms*, Acta Numerica **29** (2020), 403–572.
- [29] Daniel Potts, Gabriele Steidl, and Manfred Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, pp. 247–270, Birkhäuser Boston, Boston, MA, 2001.
- [30] Diego Ruiz-Antolín and Alex Townsend, *A nonuniform fast Fourier transform based on low rank approximation*, SIAM Journal on Scientific Computing **40** (2018), no. 1, A529–A547.

- [31] Heather Wilber, Ethan N. Epperly, and Alex H. Barnett, *Superfast direct inversion of the nonuniform discrete fourier transform via hierarchically semiseparable least squares*, SIAM J. Sci. Comput. **47** (2025), no. 3, A1702–A1732.
- [32] Heather Denise Wilber, *Computing numerically with rational functions*, Ph.d., Cornell University, 2021.
- [33] Yuanzhe Xi, Jianlin Xia, Stephen Cauley, and Venkataramanan Balakrishnan, *Superfast and stable structured solvers for Toeplitz least squares via randomized sampling*, SIAM Journal on Matrix Analysis and Applications **35** (2014), no. 1, 44–72.
- [34] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xianye S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications **17** (2010), no. 6, 953–976.
- [35] Robert M Young, *An introduction to nonharmonic fourier series*, vol. 93, Academic press, 1981.
- [36] Annan Yu and Alex Townsend, *On the stability of unevenly spaced samples for interpolation and quadrature*, BIT Numerical Mathematics **63** (2023), no. 2, 23.

A Properties of The Type-III NUDFT Approximation

We discuss the HSS rank of \mathbf{H}_{HSS} . To do this, we extract the $N/2 \times N/2$ top-right submatrix of the $N \times N$ matrix $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$ under the Random-Perturbation case and compute its ε -rank r_ε . The results in terms of $\varepsilon = 10^{-7}$ are plotted in Figure A.1. It is obvious that the numerical rank is significantly small compared to

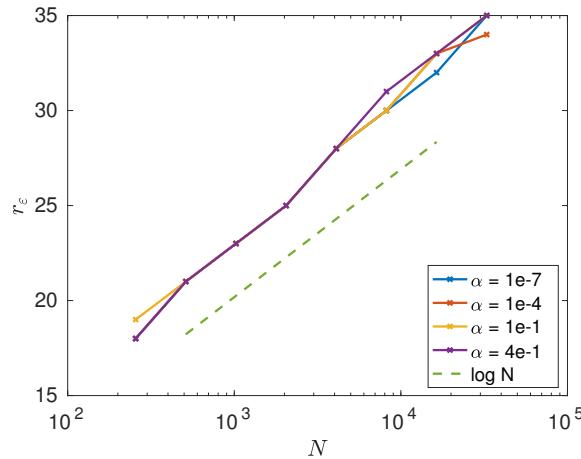


Figure A.1: ε -rank of the top-right submatrix in $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$. Here $\varepsilon = 10^{-7}$ and the size $N = 2^n$ for $n = 8 : 15$. The size of the submatrix is $N/2$.

the size of the submatrix, implying the low-rank structure of \mathbf{H} . Furthermore, given a fixed accuracy ε , the rank of the submatrix behaves approximately as $\mathcal{O}(\log N)$ relative to the matrix size, consistent with our intuition and discussion in Section 3.2.

The second experiment aims to figure out the source of errors in our fast structure (3.9). The following errors occurring in the computation are taken into consideration:

- e_B : The error using \mathbf{B}_{fast} to approximate the NUDFT-II matrix \mathbf{B} .
- e_H : The error using \mathbf{H}_{HSS} to approximate $\mathbf{B}_{\text{fast}}^\dagger \mathbf{A}$.
- e_A : The error using $\mathbf{A}_{\text{fast}} = \mathbf{B}_{\text{fast}} \mathbf{H}_{\text{HSS}}$ to approximate \mathbf{A} .

For each approximation, we estimate the relative error through applying them on 30 random sample points. Focusing on the Random-Perturbation case, we fix $N = 262144$ and let τ vary in $10^{-12}, 10^{-7}$ and 10^{-4} . Corresponding results are plotted in Figure A.2. Interestingly, when perturbation size is relatively small

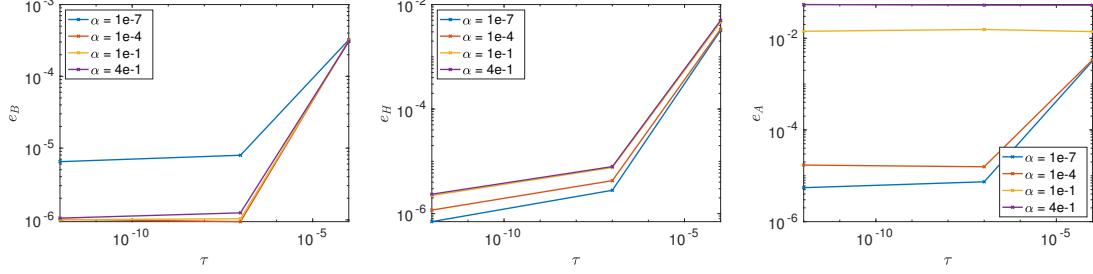


Figure A.2: Different approximate errors in fast structure (3.9) for different τ and α under the Random-Perturbation case. $N = 262144$ and $M = 4N$. Left: e_B . Mid: e_H . Right: e_A .

($\alpha = 10^{-7}$ or 10^{-4}), the dominate error is e_B and e_H since all 3 errors go up when we increases τ . However, when perturbation size is relatively large ($\alpha = 0.1$ or 0.4), e_A begins taking the leading place. In this case, no matter how small the τ is, e_A remains almost unchanged.