SIAM J. SCI. COMPUT.
Vol. 38, No. 2, pp. A1100–A1120

DIRECT INVERSION OF THE THREE-DIMENSIONAL PSEUDO-POLAR FOURIER TRANSFORM*

AMIR AVERBUCH†, GIL SHABAT‡, AND YOEL SHKOLNISKY§

Abstract. The pseudo-polar Fourier transform is a specialized nonequally spaced Fourier transform, which evaluates the Fourier transform on a near-polar grid known as the pseudo-polar grid. The advantage of the pseudo-polar grid over other nonuniform sampling geometries is that the transformation, which samples the Fourier transform on the pseudo-polar grid, can be inverted using a fast and stable algorithm. For other sampling geometries, even if the nonequally spaced Fourier transform can be inverted, the only known algorithms are iterative. The convergence speed of these algorithms and their accuracy are difficult to control, as they depend both on the sampling geometry and on the unknown reconstructed object. In this paper, a direct inversion algorithm for the three-dimensional pseudo-polar Fourier transform is presented. The algorithm is based only on one-dimensional resampling operations and is shown to be significantly faster than existing iterative inversion algorithms.

Key words. 3D pseudo-polar Fourier transform, Radon transform, unequally spaced FFT, polar Fourier transform, Toeplitz matrices

AMS subject classifications. 65T50, 44A12, 92C55

DOI. 10.1137/15M1031916

1. Introduction.

The fast Fourier transform (FFT) is one of the most commonly used algorithms and has far reaching implications in science and technology [7]. While the FFT efficiently computes the discrete Fourier transform (DFT) on a Cartesian grid, in many applications it is required to compute the Fourier transform on non-Cartesian grids. Computing the FFT on a non-Cartesian grid can be implemented using one of the available nonequally spaced FFTs (NUFFTs) [8, 10, 16, 14, 35, 20]. Of particular importance in applications are the polar grids in two and three dimensions, which emerge naturally in a wide range of applications, from computerized tomography to nano-materials to image processing [37, 6, 43, 11, 22, 30, 29, 27, 4, 25].

Although the Fourier transform can be evaluated on a polar grid by using one of the available NUFFT algorithms, there are two factors that may limit their applicability to large problems. First, although all NUFFT algorithms have the same asymptotic complexity as the classical FFT, namely, $O(n \log n)$, where n is the number of input/output samples, their runtime complexity involves rather larger constants. This is due to local interpolations used by all these methods to resample from the Cartesian grid to the polar grid. Therefore, although these algorithms are asymptotically as efficient as the FFT, they are prohibitively slow for large input sizes. Second, as many real-life problems can be formulated as the recovery of image samples from frequency samples (e.g., medical imagery reconstruction algorithms), they actually require computing the inverse Fourier transform. However, unlike the FFT, whose inverse is equal to its adjoint and thus can be easily inverted, the NUFFT, except

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 21, 2015; accepted for publication (in revised form) February 4, 2016; published electronically April 14, 2016.

http://www.siam.org/journals/sisc/38-2/M103191.html

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (amir@math.tau.ac.il).

‡Corresponding author. School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (gil.shabat@cs.tau.ac.il).

§School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel (yoelsh@post.tau.ac.il). The research of this author was supported by Israel Science Foundation grant 578/14.

for very special cases, does not have this property, and therefore, more elaborated inversion schemes are required. In particular, although the transform that evaluates the Fourier transform on the polar grid is formally invertible, the condition number of this transformation excludes such inversion in practice.

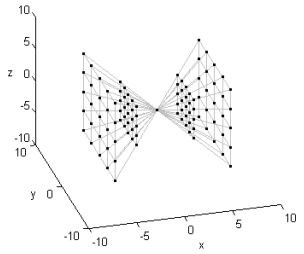
Although the polar sampling geometry of Fourier space is natural for various applications such as computerized tomography [9] or electron microscopy [37, 6], it is sometimes possible to overcome the aforementioned difficulties by replacing the polar frequency sampling grid with the pseudo-polar frequency sampling grid. The pseudo-polar grid [2] (to be described below) is nearly polar, but unlike the polar grid, it is not equally spaced in the angular direction but uses equally spaced slopes. We refer to the transformation which samples the Fourier transform on the pseudo-polar grid as the pseudo-polar Fourier transform (PPFT). Both the PPFT and its inverse admit efficient and stable algorithms. Iterative algorithms used to invert the PPFT can be found in [2, 21, 12, 36]. Since the PPFT is ill-conditioned, these inversion algorithms require preconditioning. Several preconditioning approaches have been suggested, such as using Voronoi weights [13] or a preconditioner derived from the Jacobian [38]. In particular, it is shown in [38] that under the appropriate preconditioning, the condition number of the PPFT is small. Therefore, it can be efficiently inverted using iterative inversion schemes. Nevertheless, such an iterative inversion requires one to compute the PPFT and its adjoint in each iteration, which, as mentioned above, is too expensive for large problems. Even if the PPFT and its adjoint are computed using specialized algorithms [3], the number of iterations required for the inversion may be of the order of a few dozen, which may be too slow for large-scale problems. Moreover, the accuracy of the object reconstructed by such iterative methods typically cannot be estimated from the convergence criterion and more calculations are needed to verify that the reconstructed object has the required accuracy.

The PPFT can also be expressed as a multilevel Toeplitz operator [36]. Thus, the inverse PPFT is equivalent to the inversion of this operator. Direct and iterative algorithms have been devised for inverting Toeplitz-related operators. Direct inversion algorithms, which are stable and fast, are available for Toeplitz matrices [41, 5]. On the other hand, these algorithms do not extend to multilevel Toeplitz operators. The only available exact algorithm for inverting two-level Toeplitz operators [42] is not applicable to our setting since it assumes that the two-level Toeplitz matrices are triangular in one or both their levels.

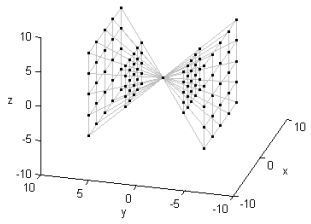
In this paper, we present a direct algorithm for inverting the three-dimensional (3D) PPFT, which generalizes the 2D direct inversion algorithm in [2]. Unlike the iterative approaches, the proposed algorithm terminates within a fixed number of operations, which is determined only by the size of the input. The inversion algorithm is shown numerically to have high accuracy, and since most of its steps are implemented in place, its memory requirements are sufficiently low to be applicable to very large inputs. Finally, the algorithm is numerically stable since it consists of a series of well-conditioned steps. In particular, no preconditioner is needed.

This paper is organized as follows. In section 2, we provide the required background on the pseudo-polar grid and the PPFT. In section 3, we give a detailed description of the direct inversion algorithm for the 3D PPFT. Numerical results and comparisons to other algorithms are given in section 4. Finally, some concluding remarks are given in section 5.

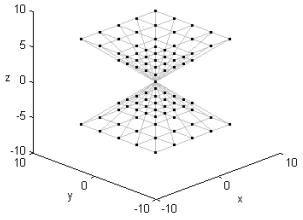
2. Mathematical preliminaries. In this section, we provide the mathematical background required to derive our inversion algorithm. In section 2.1 we revisit the



(a) The sector Ω_{ppx} for $n = 4$



(b) The sector Ω_{ppy} for $n = 4$



(c) The sector Ω_{ppz} for $n = 4$

FIG. 1. *The three sectors $\Omega_{ppx}, \Omega_{ppy}$, and Ω_{ppz} that form the 3D pseudo-polar grid.*

PPFT. In section 2.2, we describe a fast algorithm for solving Toeplitz systems of equations. This algorithm is used in section 2.3 for fast resampling of univariate trigonometric polynomials.

2.1. Pseudo-polar Fourier transform. The 3D pseudo-polar grid, denoted Ω_{pp}, is defined by

$$\Omega_{pp} = \Omega_{ppx} \cup \Omega_{ppy} \cup \Omega_{ppz}, \tag{1}$$

where

$$\begin{aligned}
\Omega_{ppx} &= \left\{ \left(k, -\tfrac{2l}{n}k, -\tfrac{2j}{n}k\right) : l, j = -n/2, \ldots, n/2,\ k = -\left\lfloor \tfrac{m}{2} \right\rfloor, \ldots, \left\lfloor \tfrac{m}{2} \right\rfloor \right\}, \\
\Omega_{ppy} &= \left\{ \left(-\tfrac{2l}{n}k, k, -\tfrac{2j}{n}k\right) : l, j = -n/2, \ldots, n/2,\ k = -\left\lfloor \tfrac{m}{2} \right\rfloor, \ldots, \left\lfloor \tfrac{m}{2} \right\rfloor \right\}, \\
\Omega_{ppz} &= \left\{ \left(-\tfrac{2l}{n}k, -\tfrac{2j}{n}k, k\right) : l, j = -n/2, \ldots, n/2,\ k = -\left\lfloor \tfrac{m}{2} \right\rfloor, \ldots, \left\lfloor \tfrac{m}{2} \right\rfloor \right\},
\end{aligned} \tag{2}$$

with $m = qn + 1$ for an even n and a positive integer q. We denote a specific point in Ω_{ppx}, Ω_{ppy}, and Ω_{ppz} by $\Omega_{ppx}(k, l, j)$, $\Omega_{ppy}(k, l, j)$, and $\Omega_{ppz}(k, l, j)$, respectively. The pseudo-polar grid is illustrated in Figure 1 for $n = 4$ and $q = 1$. As can be seen from Figure 1, the pseudo-polar grid consists of equally spaced samples along rays, where different rays have equally spaced slopes but the angles between adjacent rays are not equal. This is the key difference between the pseudo-polar grid and the polar grid [1]. Thus, we can refer to k as a "pseudo-radius" and to l and j as "pseudo-angles."

Next, we define the discrete time Fourier transform of an $n \times n \times n$ volume I by

$$\hat{I}(\omega_x, \omega_y, \omega_z) = \sum_{u,v,w=-n/2}^{n/2-1} I(u,v,w)\, e^{2\pi\imath(u\omega_x + v\omega_y + w\omega_z)/m}, \tag{3}$$

where $\omega_x, \omega_y, \omega_z \in [-m/2, m/2]$, and as before $m = qn + 1$ for an even n and a positive integer q. The 3D PPFT is defined as the samples of the discrete time Fourier transform of (3) on the pseudo-polar grid Ω_{pp}. Specifically, if we denote the concatenation of three arrays A_1, A_2, and A_3 by $A = [A_1, A_2, A_3]$, then the PPFT of a volume $I \in \mathbb{C}^{n\times n\times n}$ is an array $\hat{I}_{\Omega_{pp}} \in \mathbb{C}^{3\times(n+1)\times(n+1)\times(3n+1)}$ given by

$$\hat{I}_{\Omega_{pp}} = \left[\hat{I}_{\Omega_{ppx}}, \hat{I}_{\Omega_{ppy}}, \hat{I}_{\Omega_{ppz}}\right], \tag{4}$$

where

$$\begin{aligned} \hat{I}_{\Omega_{ppx}} &= \hat{I}\left(k, -\frac{2l}{n}k, -\frac{2j}{n}k\right), \quad \hat{I}_{\Omega_{ppy}} = \hat{I}\left(-\frac{2l}{n}k, k, -\frac{2j}{n}k\right), \\ \hat{I}_{\Omega_{ppz}} &= \hat{I}\left(-\frac{2l}{n}k, -\frac{2j}{n}k, k\right), \end{aligned} \tag{5}$$

and $\Omega_{ppx}, \Omega_{ppy}$, and Ω_{ppz} are defined in (2). The parameter q in (1) and (3) determines the frequency resolution of the transform, as well as its geometric properties. For example, as shown in [3], in order to derive a 3D discrete Radon transform based on the PPFT, q must satisfy $q \geq 3$.

The pseudo-polar grid appeared in the literature several times under different names. It was originally introduced by [34] under the name "concentric squares grid" in the context of computerized tomography. More recent works in the context of computerized tomography, which take advantage of the favorable numerical and computational properties of the grid, include [32, 33], where equally sloped tomography is used for radiation dose reduction. Other image processing applications that use the pseudo-polar grid include synthetic aperture radar imaging [28], Shearlets [27, 26], registration [31], and denoising [40], to name a few.

Recently, [38] proposed fast iterative inversion algorithms for the 2D and 3D PPFT, which are based on the well-known convolution structure of their Gram operators, combined with a preconditioned conjugate gradients [19] solver. The convolution structure of the transform allows one to invert it using the highly optimized FFT algorithm instead of the forward and adjoint transforms derived in [2]. As the conjugate gradients iterations require preconditioning, [38] proposes a preconditioner that leads to a very small condition number, thus making the inversion process fast and accurate. However, this approach has drawbacks such as high memory requirements and dependency of the number of iterations on the size of the data. The algorithm presented in the current paper, which has low memory requirements, achieves high accuracy and is faster than the iterative algorithm [38].

2.2. Solving Toeplitz systems. Let A_n be an $n \times n$ Toeplitz matrix and let y be an arbitrary vector of length n. We describe a fast algorithm for computing $A_n^{-1}y$. This algorithm is well-known and appears, for example, in [2], but we repeat it here for completeness of the description. The algorithm consists of a fast factorization of the inverse Toeplitz matrix followed by a fast algorithm that applies the inverse matrix to a vector [17, 24]. We denote by $T_n(c, r)$ an $n \times n$ Toeplitz matrix whose first column and row are c and r, respectively. For symmetric matrices, $c = r$.

Circulant matrices are diagonalized by the Fourier matrix. Hence, a circulant matrix C_n can be written as $C_n = W_n^* D_n W_n$, where D_n is a diagonal matrix containing the eigenvalues $\lambda_1, \ldots, \lambda_n$ of C_n, and W_n is the Fourier matrix given by $W_n(j,k) = \frac{1}{\sqrt{n}} e^{2\pi \imath jk/n}$. Moreover, if $c = [c_0, c_1, \ldots, c_{n-1}]^T$ is the first column of C_n, then $W_n c = [\lambda_1, \ldots, \lambda_n]^T$. Obviously, the matrices W_n and W_n^* can be applied in $O(n \log n)$ operations using the FFT. Thus, the multiplication of C_n with an arbitrary vector x of length n can be implemented in $O(n \log n)$ operations by applying FFT to x, multiplying the result by D_n, and then applying the inverse FFT.

To compute $A_n x$ for an arbitrary Toeplitz matrix $A_n = T_n(c, r)$ and an arbitrary vector x, we first embed A_n in a $2n \times 2n$ circulant matrix C_{2n}

$$C_{2n} = \begin{pmatrix} A_n & B_n \\ B_n & A_n \end{pmatrix},$$

where B_n is an $n \times n$ Toeplitz matrix given by

$$B_n = T_n([0, r_{n-1}, \ldots, r_2, r_1], [0, c_{n-1}, \ldots, c_2, c_1]).$$

Then, $A_n x$ is computed in $O(n \log n)$ operations by zero padding x to length $2n$, applying C_{2n} to the padded vector, and discarding the last n elements of the resulting vector.

Next, assume that A_n is invertible. The Gohberg–Semencul formula [17, 18] provides an explicit representation of A_n^{-1} as

$$A_n^{-1} = \frac{1}{x_0} \left(M_1 M_2 - M_3 M_4 \right), \tag{6}$$

where

$$M_1 = T_n([x_0, x_1, \ldots, x_{n-1}], [x_0, 0, \ldots, 0]), \tag{7}$$

$$M_2 = T_n([y_{n-1}, 0, \ldots, 0], [y_{n-1}, y_{n-2}, \ldots, y_0]), \tag{8}$$

$$M_3 = T_n([0, y_0, \ldots, y_{n-2}], [0, \ldots, 0]), \tag{9}$$

$$M_4 = T_n([0, \ldots, 0], [0, x_{n-1}, \ldots, x_1]). \tag{10}$$

The vectors $x = [x_0, \ldots, x_{n-1}]$ and $y = [y_0, \ldots, y_{n-1}]$ are given as the solutions to

$$\begin{aligned} A_n x &= e_0, & e_0 &= [1, 0, \ldots, 0]^T, \\ A_n y &= e_{n-1}, & e_{n-1} &= [0, \ldots, 0, 1]^T. \end{aligned} \tag{11}$$

The matrices M_1, M_2, M_3, and M_4 have Toeplitz structure and are represented implicitly using the vectors x and y. Hence, the total storage required to store M_1, M_2, M_3, and M_4 is that of $2n$ numbers. If the matrix A_n is fixed, then the vectors x and y can be precomputed. Once the triangular Toeplitz matrices M_1, M_2, M_3, and M_4 have been computed, the application of A_n^{-1} is reduced to the application of four Toeplitz matrices. Thus, the application of A_n^{-1} to a vector requires $O(n \log n)$ operations. The pseudo-code of applying A_n^{-1} to a vector is described in Algorithms 7, 8, and 9 in the appendix. Algorithm 7 lists the function `ToeplitzDiag`, which computes the diagonal form of the circulant embedding of a Toeplitz matrix. Algorithm 8 lists the function `ToeplitzInvMul`, which efficiently multiplies an inverse Toeplitz matrix, given by the diagonal forms of its Gohberg–Semencul factors (see (6)), by a vector. The latter function uses `ToeplitzMul`, listed in Algorithm 9, which efficiently multiplies a general Toeplitz matrix by a vector.

2.3. Resampling trigonometric polynomials. The main (and in fact the only) tool behind our algorithm is resampling of univariate trigonometric polynomials. Assume we are given a set of points $\{y_j\}_{j=1}^N$, $y_j \in [-\pi, \pi]$, and their values $\{f(y_j)\}_{j=1}^N$, where f is some unknown univariate trigonometric polynomial of degree $n \leq N$. We want to estimate the values of f at a new set of points $\{x_j\}_{j=1}^M$, $x_j \in [-\pi, \pi]$. This can be formulated by first solving

$$\min_{\alpha} \sum_{j=1}^{N} \left| f(y_j) - \sum_{k=-n/2}^{n/2-1} \alpha_k e^{\imath k y_j} \right|^2, \quad \alpha = \left(a_{-n/2}, \ldots, \alpha_{n/2-1}\right) \in \mathbb{C}^n, \tag{12}$$

for the coefficients vector α, followed by evaluating

$$f(x_i) = \sum_{k=-n/2}^{n/2-1} \alpha_k e^{\imath k x_i}. \tag{13}$$

In matrix notation, (12) is written as

$$\min_{\alpha} \|f - A\alpha\|_2, \tag{14}$$

where the entries of the matrix A are given by $a_{jk} = e^{\imath k y_j}$ and the coordinates of the vector f are $f_j = f(y_j)$ (we denote by f both the vector and the function, as the appropriate meaning is clear from the context). Direct solution of (14) for the coefficients vector α requires $\mathcal{O}(n^3)$ operations (assuming N and M are of size $O(n)$). Obviously, solving for α also depends on the condition number of A, which affects the accuracy of the estimate of $f(x_i)$.

We next present a fast algorithm for computing $f(x_i)$ (faster than directly solving first for α), which exploits the Toeplitz structure of A^*A and uses the NUFFT [8, 10, 16, 14, 35, 20]. The resulting algorithm has complexity of $\mathcal{O}(n \log n + n \log 1/\epsilon)$ operations, where ϵ is the accuracy of the computations, in addition to a preprocessing step that takes $\mathcal{O}(n^2)$ operations.

Following [20], we define NUFFT type I by

$$f_k = \frac{1}{n_j} \sum_{j=1}^{n_j} c_j e^{\pm \imath k x_j}, \quad k = -n/2, \ldots, n/2 - 1, \tag{15}$$

and NUFFT type II by

$$c_j = \sum_{k=-n/2}^{n/2-1} f_k e^{\pm \imath k x_j}, \quad j = 1, \ldots, n_j. \tag{16}$$

The sums in (15) and (16) can be approximated to a relative accuracy ϵ in $\mathcal{O}(n \log n + n \log 1/\epsilon)$ operations by any of the aforementioned NUFFT algorithms. From the definitions of NUFFT of types I and II, we see that A^*f, where A is the matrix from (14) and f is an arbitrary vector, is equal to the application of NUFFT type I to f. Similarly, the application of A to an arbitrary vector c is equal to the application of NUFFT type II to c. Hence, the application of A or A^* to a vector can be implemented in $\mathcal{O}(n \log n + n \log 1/\epsilon)$ operations.

To solve the least-squares problem of (14) we form the normal equations

$$A^*A\alpha = A^*f. \tag{17}$$

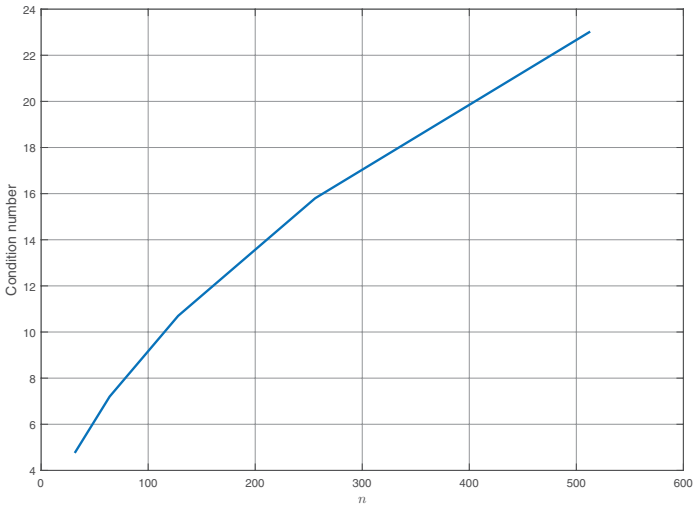


FIG. 2. *The maximal condition number of A (see (17)) obtained while applying Algorithm 5 to various sizes n.*

The right-hand side of (17) can be computed efficiently using NUFFT type I. The matrix A^*A is a symmetric Toeplitz matrix of size $n \times n$. The first column of A^*A, which due to the symmetric Toeplitz structure encodes all its entries, is computed efficiently by applying NUFFT type I to the vector w whose entries are $w_j = e^{-\imath n y_j/2}$. Computing $(A^*A)^{-1}$ takes $\mathcal{O}(n^2)$ operations using the Durbin–Levinson algorithm [15] and applying it to a vector takes $\mathcal{O}(n \log n)$ operations using the Gohberg–Semencul formula [23], as was described in section 2.2. This procedure is described in detail in [2]. Since $(A^*A)^{-1}$ depends only on n, it can be precomputed. Therefore, solving for α in (17) takes $\mathcal{O}(n \log n + n \log 1/\epsilon)$ operations and computing $f(x_i)$ in (13) takes additional $\mathcal{O}(n \log n + n \log 1/\epsilon)$ operations using NUFFT type II. The entire resampling algorithm is described in Algorithm 5. Accurately solving for α requires A to have a small condition number. The maximal condition number of A obtained in Algorithm 5 (solving (17)) while being called from Algorithm 2 for various inputs and different values of n is illustrated in Figure 2. Note that the operator A depends also on the values x_j, $j = 1, \ldots, n_j$, and not only on n. As can be seen, the condition number is less than 25 even for very large volumes.

3. Direct inversion of the 3D PPFT. Given the PPFT $\hat{I}_{\Omega_{pp}}$ defined in (4) of an unknown $n \times n \times n$ volume I, the proposed direct inversion algorithm recovers I in two steps. The first step resamples the PPFT to an intermediate Cartesian grid. Specifically, we resample the trigonometric polynomial $\hat{I}$ of (3) from the grid Ω_{pp} in (1) to the frequency grid

$$\Omega_c = \{(qu, qv, qw) : u, v, w = -n/2, \ldots, n/2\}. \tag{18}$$

The second step of our algorithm recovers I from the samples of $\hat{I}$ on Ω_c. Note that this second step cannot be directly implemented by inverse FFT. The two steps of the algorithm are described in sections 3.1 and 3.2, respectively, its pseudo-code is given in Algorithm 1, and its complexity is analyzed in section 3.3.

3.1. Resampling the pseudo-polar grid to the grid Ω_c. We start by describing the procedure for resampling $\hat{I}$ from Ω_{pp} to Ω_c. It is based on an "onion

peeling" approach, which resamples at iteration k, $k = n/2, \ldots, 0$, points in Ω_{pp} with "pseudo-radius" k to points in Ω_c that lie on a plane. When the iteration that corresponds to $k = 0$ is completed, the values of $\hat{I}$ have been computed on all points of the grid Ω_c. We define the sets $\Omega_c^{(k)} \subseteq \Omega_c$, $k = n/2, \ldots, 0$, consisting of the frequencies on which $\hat{I}$ has been resampled until (and including) iteration k. Note that for convenience, we index the iterations from $k = n/2$ to $k = 0$. Thus, we have that $\Omega_c^{(k)} \subseteq \Omega_c^{(k-1)}$ and $\Omega_c^{(0)} = \Omega_c$. Formally, we define $\Theta^{(k)}$ to be the frequencies on which we resample $\hat{I}$ at iteration k so that

$$\Omega_c^{(k)} = \Omega_c^{(k+1)} \cup \Theta^{(k)},$$

where $\Theta^{(k)}$ is given by

$$\Theta^{(k)} = \Theta_{+x}^{(k)} \cup \Theta_{-x}^{(k)} \cup \Theta_{+y}^{(k)} \cup \Theta_{-y}^{(k)} \cup \Theta_{+z}^{(k)} \cup \Theta_{-z}^{(k)}, \tag{19}$$

and for $j, l = -k, \ldots, k$

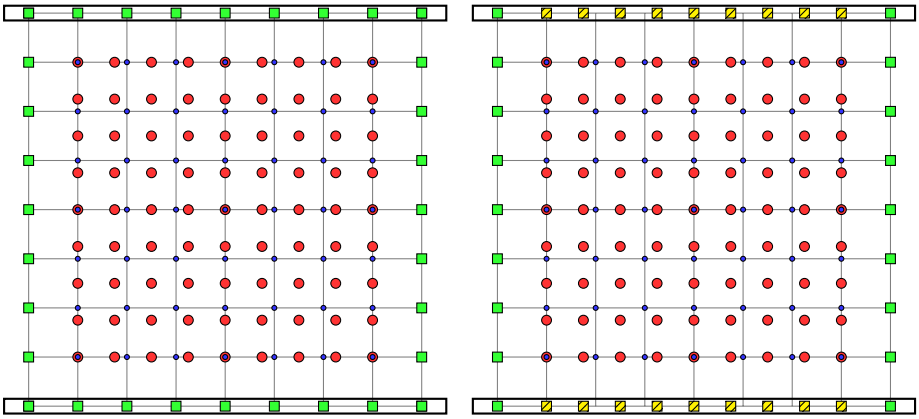
$$\begin{aligned} \Theta_{+x}^{(k)} &= \{(qk, qj, ql)\}, \quad \Theta_{-x}^{(k)} = \{(-qk, qj, ql)\}, \\ \Theta_{+y}^{(k)} &= \{(qj, qk, ql)\}, \quad \Theta_{-y}^{(k)} = \{(qj, -qk, ql)\}, \\ \Theta_{+z}^{(k)} &= \{(qj, ql, qk)\}, \quad \Theta_{-z}^{(k)} = \{(qj, ql, -qk)\}. \end{aligned} \tag{20}$$

For $k = n/2$ we set $\Omega_c^{(n/2)} = \Theta^{(n/2)}$. Moreover, the frequencies of $\Theta^{(n/2)}$ are the frequencies $\Omega_{ppx}(\pm n/2, j, l)$, $\Omega_{ppy}(\pm n/2, j, l)$, and $\Omega_{ppz}(\pm n/2, j, l)$, $l, j = -n/2, \ldots, n/2$, defined in (2). Thus, the values of $\hat{I}$ on $\Omega_c^{(n/2)}$ are given as a subset of the values of $\hat{I}$ on Ω_{pp}. Therefore, no resampling is needed at this iteration.

For subsequent iterations, we use an example to accompany and clarify the formal description. We use as an example a grid corresponding to $n = 8$ that demonstrates how the values of $\hat{I}$ on $\Theta^{(n/2-1)}$ (which is $\Theta^{(3)}$ in our particular example) are recovered. Since the same procedure recovers the values of $\hat{I}$ on $\Theta_{\pm x}^{(k)}$, $\Theta_{\pm y}^{(k)}$, and $\Theta_{\pm z}^{(k)}$, we only explain the process for the grid $\Theta_{+x}^{(k)}$. The pseudo-code for recovering the values of $\hat{I}$ on $\Theta^{(k)}$ is given in Algorithm 2, whose details are also explained below.

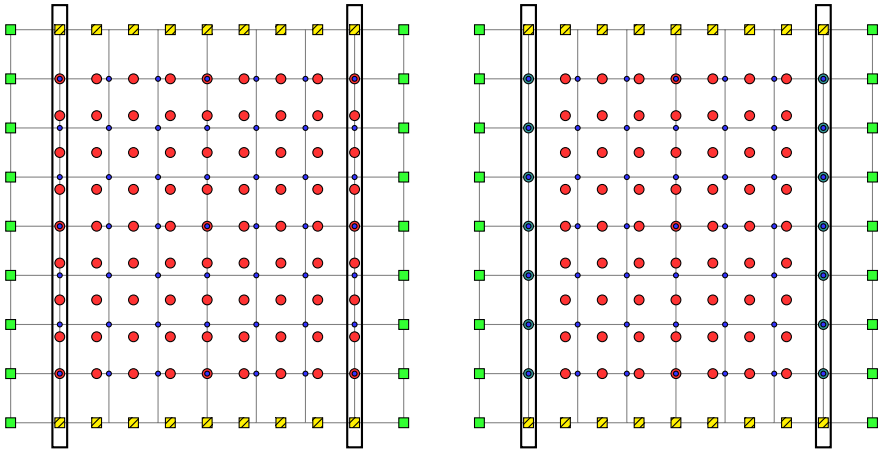
In Figure 3(a), green solid squares correspond to points from $\Omega_c^{(k+1)}$ on which $\hat{I}$ has already been evaluated (in previous iterations), red circles correspond to points from Ω_{ppx} with a fixed $k = n/2 - 1$, and blue dots correspond to points of $\Theta_{+x}^{(k)}$ on which we want to evaluate $\hat{I}$. The frequencies in $\Theta_{+x}^{(k)}$ are points in $\mathbb{R}^3$; however, as can be seen from (20), they all lie on the same plane, and therefore, we depict them as a 2D image whose axes are i and j from (20).

The first step, depicted in Figure 3(b), consists of resampling the points of $\Omega_c^{(k+1)}$ (solid green squares) to the same spacing as the pseudo-polar points. In the case of $k = n/2 - 1$, this means that the first and last rows are resampled, and the result of this resampling is denoted by patterned yellow squares. This step is implemented by lines 11−14 in Algorithm 2. Next, as depicted in Figure 4(a), for all columns, we use the latter resampled points (patterned yellow squares) together with the points of the pseudo-polar grid at the same column to resample $\hat{I}$ to intermediate sampling points. Figures 4(a) and 4(b) depict the resampling of one such column. Note that the resampled values of $\hat{I}$ after this step, which are depicted as filled teal circles in Figure 4(b), are not yet the values of $\hat{I}$ on the points of $\Theta_x^{(k)}$, as up to here, we only applied resampling to the columns (but not yet to rows). We repeat the latter resampling for



(a) Beginning of iteration $k = n/2 - 1$ (b) After line 14 in Algorithm 2

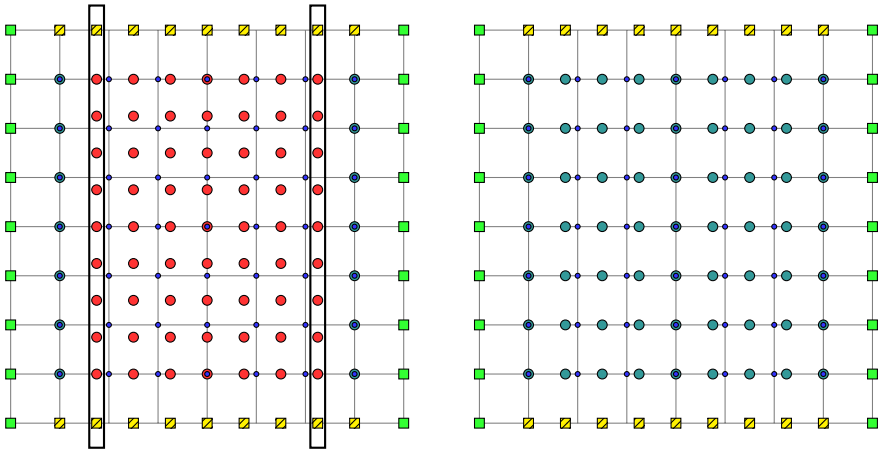
FIG. 3. *Illustration of lines* 11–14 *in Algorithm* 2. *Green solid squares correspond to points from* $\Omega_c^{(k+1)}$, *red circles correspond to points from* Ω_{ppx} *with a fixed* $k = n/2 - 1$, *blue dots correspond to points of* $\Theta_{+x}^{(k)}$, *and patterned yellow points correspond to point of* $\Omega_c^{(k+1)}$ *resampled to their intermediate position.*



(a) Before resampling (b) After resampling

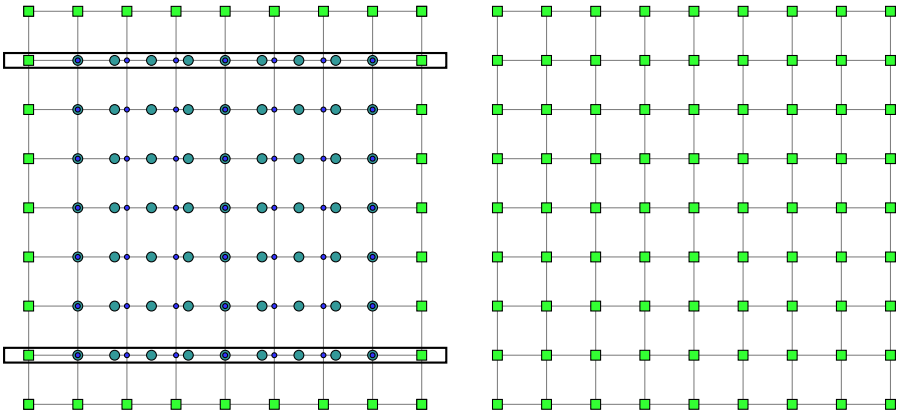
FIG. 4. *Illustration of one iteration of lines* 18–20 *in Algorithm* 2.

all the columns (see Figure 5(a) for resampling of another column), which results in the grid of Figure 5(b). This is implemented by lines $18-20$ of Algorithm 2, which using the conventions of Figure 5 resamples the red circles along the columns to the filled teal circles, which are now on the same rows as the points indicated by the blue dots (our target sampling points). Next, we apply 1D resampling to each row by using the resampled points from the previous steps together with the points of $\Omega_c^{(k+1)}$ (Figure 6(a)) to recover the samples of $\hat{I}$ on $\Theta_x^{(k)}$ (Figure 6(b)). This is implemented by lines $22-24$ of Algorithm 2. All 1D resampling operations in Algorithm 2 are implemented using the function `ToeplitzResample` in Algorithm 5.



(a) Before resampling (b) After resampling

FIG. 5. *Illustration of further iterations of lines* 18–20 *in Algorithm* 2.



(a) Before resampling (b) After resampling

FIG. 6. *Illustration of lines* 22–24 *in Algorithm* 2 *(row resampling).*

3.2. Recovering I from $\hat{I}$ on Ω_c. The second step in Algorithm 1 (line 11) recovers the volume I from the samples of $\hat{I}$ on Ω_c (see (18)). Define the operator $F_D : \mathbb{C}^n \to \mathbb{C}^{n+1}$ by

$$(F_D u)_k = \sum_{j=-n/2}^{n/2-1} u_j e^{-2\pi i j q k/m}, \quad k = -n/2, \ldots, n/2, \quad m = qn + 1. \tag{21}$$

For a volume I of size $n \times n \times n$, we denote by $F_D^{(x)}$, $F_D^{(y)}$, and $F_D^{(z)}$ the application of F_D to the x, y, and z dimensions of I, respectively. Furthermore, we define

$$\tilde{I} = F_D^{(z)} F_D^{(y)} F_D^{(x)} I. \tag{22}$$

Algorithm 1. Direct inversion of the 3D PPFT.

1: **Input:** Samples of the 3D PPFT $\hat{I}_{\Omega_{ppx}}, \hat{I}_{\Omega_{ppy}}, \hat{I}_{\Omega_{ppz}}$, given by (5), each of size $(qn+1)\times(n+1)\times(n+1)$.
2: **Output:** Volume I of size $n\times n\times n$.
3: $\hat{I}_D \leftarrow \text{zeros}(n+1, n+1, n+1)$
4: **for** $j = 1, \ldots, n/2$ **do**
5: $\quad \hat{I}_D(j,:,:) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppx}}(q(j-1)+1,:,:), \hat{I}_D(j,:,:), j-1)$ ▷ Algorithm 2
6: $\quad \hat{I}_D(n+2-j,:,:) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppx}}(qn-q(j-1)+1, n+1:-1:1, n+1:-1:1), \hat{I}_D(n+2-j,:,:), j-1)$
7: $\quad \hat{I}_D(:,j,:) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppy}}(q(j-1)+1,:,:), \hat{I}_D(:,j,:), j-1)$
8: $\quad \hat{I}_D(:,n+2-j,:) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppy}}(qn-q(j-1)+1, n+1:-1:1, n+1:-1:1), \hat{I}_D(:,n+2-j,:), j-1)$
9: $\quad \hat{I}_D(:,:,j) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppz}}(q(j-1)+1,:,:), \hat{I}_D(:,:,j), j-1)$
10: $\quad \hat{I}_D(:,:,n+2-j) \leftarrow \texttt{recover2d}(\hat{I}_{\Omega_{ppz}}(qn-q(j-1)+1, n+1:-1:1, n+1:-1:1), \hat{I}_D(:,:,n+2-j), j-1)$
11: **end for**
12: $\hat{I}_D(n/2+1, n/2+1, n/2+1) \leftarrow \hat{I}_{\Omega_{ppx}}(3n/2+1, n/2+1, n/2+1)$ (center point)
13: $I \leftarrow \texttt{InvDecimatedFreq}(\hat{I}_D)$ ▷ (recover I from $\hat{I}_D$ by Algorithm 4)

From the definitions of F_D, $\hat{I}$, and Ω_c in (21), (3), and (18), respectively, we conclude that $\tilde{I}$ is equal to the samples of $\hat{I}$ on the grid Ω_c. Thus, if we apply the inverse of F_D to the x, y, and z dimensions of $\tilde{I}$ (in a separable way), we recover I from the samples of $\hat{I}$ on Ω_c. The inversion of the operator F_D is described in [2]. Specifically, in our case, for $m = qn+1$, the adjoint of F_D is given by

$$(F_D^* w)_j = \sum_{k=-n/2}^{n/2} w_k e^{2\pi \imath jqk/m}, \quad j = -n/2, \ldots, n/2-1. \tag{23}$$

The matrix $F_D^* F_D$ is a Toeplitz matrix, whose entries are given by

$$(F_D^* F_D)_{k,l} = \sum_{j=-n/2}^{n/2} e^{2\pi \imath qj(k-l)/m}, \quad k,l = -n/2, \ldots, n/2-1,$$

and its inverse is applied as described in (6). The volume I is recovered by the application of $(F_D^* F_D)^{-1} F_D^*$ to each dimension of $\widetilde{I}$. The operator F_D^* is applied efficiently to a vector w by adding $q-1$ zeros between every two samples of w, applying the inverse FFT to the resulting vector, and keeping the n central elements of the FFT-transformed vector (see Algorithm 3 for a detailed description). The pseudo-code of the algorithm for recovering a volume I from the samples of $\hat{I}$ on Ω_c is given in Algorithm 4.

3.3. Complexity analysis. Algorithm 1 consists of two steps: resampling (lines 1–12) and recovering I from $\hat{I}$ on Ω_c (line 13). The first step, resampling, utilizes Algorithm 5, which takes $\mathcal{O}(n \log n)$ operations when preprocessing is excluded. Algorithm 5 is called $\mathcal{O}(n)$ times in the `recover2d` function (Algorithm 2), which in turn is called also $\mathcal{O}(n)$ times by Algorithm 1. Thus, a total of $\mathcal{O}(n^3 \log n)$ operations are needed for the resampling step.

Algorithm 2. `recover2d`: Recover $\hat{I}$ on one of the subsets comprising $\Theta^{(k)}$ (see 19).

Input:

$\hat{U}_{\Omega_{pp}}$	Subset of the samples of the 3D PPFT of I of size $(n+1) \times (n+1)$.
$\hat{U}_D$	Already resampled values of size $(n+1) \times (n+1)$ of $\hat{I}_D$ from Algorithm 1.
j	An integer indicating the iteration within the peeling procedure.

Output:

result	Values of $\hat{U}_{\Omega_{pp}}$ resampled to Cartesian coordinates. Array of size $(n+1) \times (n+1)$.

$\alpha \leftarrow (n/2 - j)/(n/2)$
$m \leftarrow qn + 1$
if $j == 0$ **then**
  $result \leftarrow \hat{U}_{\Omega_{pp}}$
  return
end if
$y_1 \leftarrow [-n/2 : n/2] \times (-2) \times q \times \pi/m$
$x_1 \leftarrow [-n/2 : n/2] \times (-2) \times q \times \alpha \times \pi/m$
$C \leftarrow \text{zeros}(n+1, n+1)$
for $k = 1, \ldots, j$ **do**
  $C(k,:) \leftarrow \texttt{ToeplitzResample}\left(\hat{U}_{\Omega_{pp}}(k, 1:n+1), y_1, x_1\right)$ ▷ Algorithm 5
  $C(n+2-k,:) \leftarrow \texttt{ToeplitzResample}\left(\hat{U}_{\Omega_{pp}}(n+2-k, 1:n+1), y_1, x_1\right)$
end for
$x \leftarrow [-(n/2 - j) : (n/2 - j)] \times (-2) \times q \times \pi/m$
$y \leftarrow [[-n/2 : -n/2 + j - 1], [-n/2 : n/2] \times \alpha, [n/2 - j + 1 : n/2]] \times (-2) \times q \times \pi/m$
$R_1 \leftarrow \text{zeros}(n + 1 - 2j, n + 1)$
for $k = 1, \ldots, n + 1$ **do**
  $R_1 \leftarrow \texttt{ToeplitzResample}\left(\left[C(1:j,k), \hat{U}_{\Omega_{pp}}(:,k), C(n-j+2:n+1,k)\right], y, x\right)$
end for
$R_2 \leftarrow \text{zeros}(n + 1 - 2j, n + 1 - 2j)$
for $k = 1, \ldots, n + 1 - 2j$ **do**
  $R_2 \leftarrow \texttt{ToeplitzResample}\left(\left[\hat{U}_D(k+j, 1:j), R_1(k,:), \hat{U}_D(k+j, n-j+2:n+1)\right], y, x\right)$
end for
$result \leftarrow \hat{U}_D$
$result(1 + j : n + 1 - j, 1 + j : n + 1 - j) \leftarrow R_2$

The second step, recovering I from $\hat{I}$ on Ω_c, takes $\mathcal{O}(n^3 \log n)$ operations as well. This yields a total computational complexity of $\mathcal{O}(n^3 \log n)$ operations. If the preprocessing is done in runtime, the total complexity becomes $\mathcal{O}(n^4)$ operations, due to the Durbin–Levinson algorithm used for inverting Toeplitz matrices. The computational complexity can be reduced from $\mathcal{O}(n^4)$ operations to $\mathcal{O}(n^3 \log^2 n)$ by solving (11) as described in [23].

Algorithm 3. `AdjFDecimated`: Applying F_D^* of (23) to a vector.

Input:
y Input vector of odd length.
Output: $x = F_D^* y$
$n \leftarrow$length$(y) - 1$
$m \leftarrow qn + 1$
$z \leftarrow$ zeros$(m, 1)$
$z(1 : q : m) \leftarrow y$
$l \leftarrow$length(z)
$p_f \leftarrow$ floor$(l/2)$
$p_c \leftarrow$ ceil$(l/2)$
$x \leftarrow z([p_f + 1 : l,\ 1 : p_f])$
$x_{IFFT} \leftarrow$IFFT(x)
$x \leftarrow m \cdot x_{IFFT}([p_c + 1 : l,\ 1 : p_c])$
$x \leftarrow x(n + 1 : 2n)$

The total storage requirements for n applications of Algorithm 2 is $4n^2$. This storage is used to store the diagonal Toeplitz matrices $D_1, \ldots, D_4$.

4. Numerical results. We implemented Algorithm 1 in MATLAB and applied it to several volumes of sizes $n \times n \times n$, where $n = 32, 64, \ldots, 512$. All experiments were executed on a Linux machine with two Intel Xeon processors (CPU X5560) running at 2.8 GHz, with 8 cores in total and 96 GB of RAM. All 3D experiments were performed with $q = 3$ (see (2)).

Algorithm 2 is based on a series of 1D resampling operations. We compare two methods for implementing this 1D resampling–least-squares-based (LS-based) approach and the Toeplitz-NUFFT-based approach, both described in section 2.3. The LS-based approach, described in (12), consists of finding the coefficients vector α of a trigonometric polynomial, followed by direct evaluation of the polynomial at the resampling points. Using this resampling approach in Algorithm 2 results in computational complexity of $\mathcal{O}(n^3)$ operations (excluding a preprocessing step). Nevertheless, its implementation in MATLAB is highly optimized. The Toeplitz-NUFFT-based approach, described in section 2.3 and Algorithm 5, results in computational complexity of Algorithm 2 of $\mathcal{O}(n^2 \log n + n^2 \log 1/\epsilon)$ operations. The two resampling methods are compared by using them as the underlying 1D resampling in Algorithm 2. Both algorithms use a preprocessing step which is excluded from the reported running times. The error incurred by the two algorithms is measured by

$$E(I, I_r) = \sqrt{\frac{\sum_{k,l,m} |I[k,l,m] - I_r[k,l,m]|^2}{\sum_{k,l,m} |I[k,l,m]|^2}}, \tag{24}$$

where $I[k, m, n]$ is the original volume and $I_r[k, m, n]$ is the reconstructed volume. Running times for both methods for volumes of sizes $n \times n \times n$, where $n = 32, 64, \ldots, 512$, are shown in Figure 7. The volumes in this experiment consist of PPFT transformed random independent samples from a normal distribution with zero mean and unit variance. Figure 8 compares between the reconstruction errors of Algorithm 5 and those of LS-based resampling. The results show that both methods are comparable in terms of accuracy.

Since our hardware currently cannot process 3D volumes of sizes larger than $n = 512$, we compare the LS-based resampling and the Toeplitz-based resampling

Algorithm 4. `InvDecimatedFreq`: Recover I from $\tilde{I}$ of (22).

Input: Matrix $\tilde{I}$ of size $(n+1)\times(n+1)\times(n+1)$.
Output: Volume I of size $n\times n\times n$ such that (22) holds.
$c \leftarrow \text{zeros}(n,1)$
$m \leftarrow qn+1$
for $k=-n/2,\ldots,n/2-1$ **do**
  for $l=-n/2,\ldots,n/2$ **do**
    $c(k+n/2+1) \leftarrow c(k+n/2+1) + e^{\pi\imath ql(-n/2-k)/m}$
  end for
end for
$(M_1,M_2,M_3,M_4) \leftarrow$ `ToeplitzInv`(c) ▷ Algorithm 6
$D_i \leftarrow$ `ToeplitzDiag`(M_i), $i=1,\ldots,4$ ▷ Algorithm 7
$I_1 \leftarrow \text{zeros}(n+1,n+1,n)$
$I_2 \leftarrow \text{zeros}(n+1,n,n)$
$I \leftarrow \text{zeros}(n,n,n)$
for $k=1,\ldots,n+1$ **do**
  for $l=1,\ldots,n+1$ **do**
    $v \leftarrow \widetilde{I}(k,l,:)$
    $v \leftarrow$ `AdjFDecimated`(v) ▷ Algorithm 3
    $u \leftarrow$ `ToeplitzInvMul`(D_1,D_2,D_3,D_4,v) ▷ Algorithm 8
    $I_1(k,l,:) \leftarrow u$
  end for
end for
for $k=1,\ldots,n+1$ **do**
  for $l=1,\ldots,n$ **do**
    $v \leftarrow I_1(k,:,l)$
    $v \leftarrow$ `AdjFDecimated`(v)
    $u \leftarrow$ `ToeplitzInvMul`(D_1,D_2,D_3,D_4,v)
    $I_2(k,l,:) \leftarrow u$
  end for
end for
for $k=1,\ldots,n$ **do**
  for $l=1,\ldots,n$ **do**
    $v \leftarrow I_2(:,k,l)$
    $v \leftarrow$ `AdjFDecimated`(v)
    $u \leftarrow$ `ToeplitzInvMul`(D_1,D_2,D_3,D_4,v)
    $I(:,k,l) \leftarrow u$
  end for
end for

for larger values of n by applying both 1D resampling methods to 1D chirp signals defined by $\cos(10k_j^2)$, $k_j = -\pi + \frac{2\pi}{n}j$, $j=0,\ldots,n$, $n=512,\ 1024,\ 2048,\ 4096$. Each signal's samples given at k_j were resampled to $0.3k_j$ using LS-based resampling and Toeplitz-based resampling. The 1D chirp signal is displayed in Figure 9. The running times of the two methods, which do not include preprocessing timing, appear in Figure 10. The approximation errors of both methods are practically identical, as shown in Figure 11.

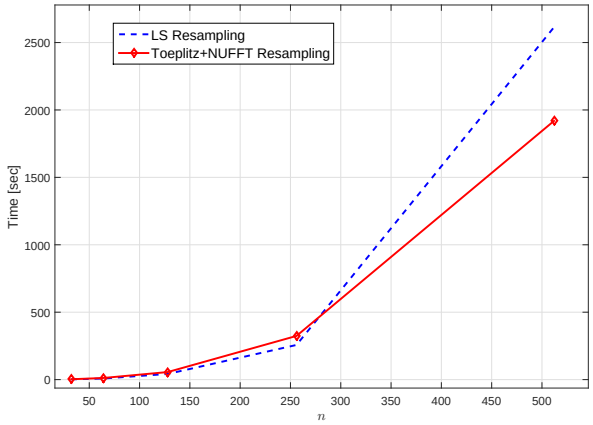


FIG. 7. *Runtime comparison between LS-based resampling and Toeplitz-NUFFT-based resampling (Algorithm 5) for inputs consisting of PPFT transformed volumes of sizes $n \times n \times n$ of independent and identically distributed (i.i.d.) normal random samples with zero mean and unit variance.*

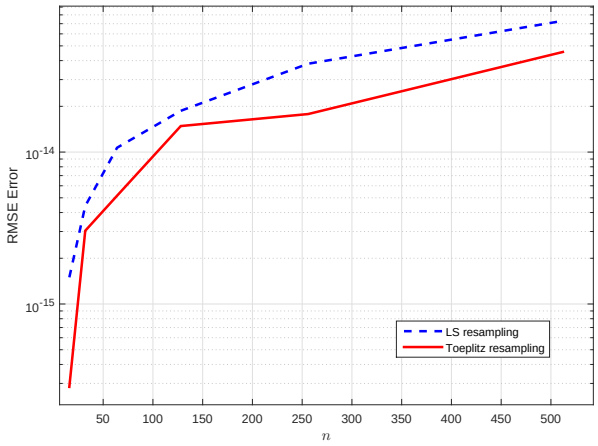


FIG. 8. *Error comparison (according to (24)) between LS-based resampling and Toeplitz-NUFFT-based resampling (Algorithm 5) for inputs consisting of PPFT transformed volumes of sizes $n \times n \times n$ of i.i.d. normal random samples with zero mean and unit variance.*

Next, we compare the direct inverse PPFT in Algorithm 1, the iterative algorithm described in [38], and a single iteration of an implementation that computes the Gram operator of the PPFT using the NUFFT as suggested by [13]. For the latter method, several iterations are required, whose number depends on the condition number of the PPFT operator. However, to keep all timings within the same scale, we compare the other two algorithms to only a single iteration of the latter. The results are illustrated in Figure 12, which shows that Algorithm 1 is faster than the iterative algorithm [38] as well as faster than a single iteration of the 3D NUFFT-based algorithm proposed by [13]. Results for $n = 512$ do not appear in Figure 12 (unlike Figure 7) as it was impossible to process volumes of that size using [38, 13].

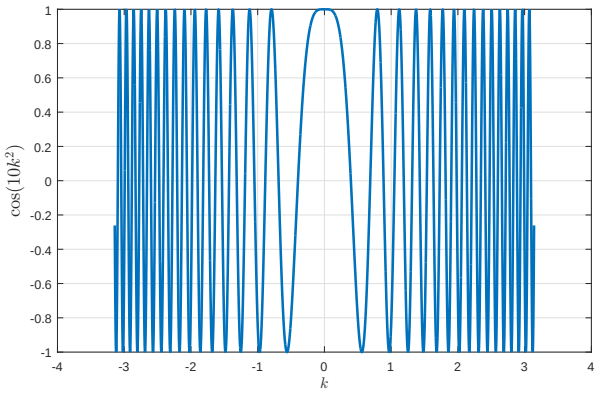


FIG. 9. 1D *chirp signal, used for comparing the LS-based resampling and the Toeplitz-NUFFT-based resampling.*

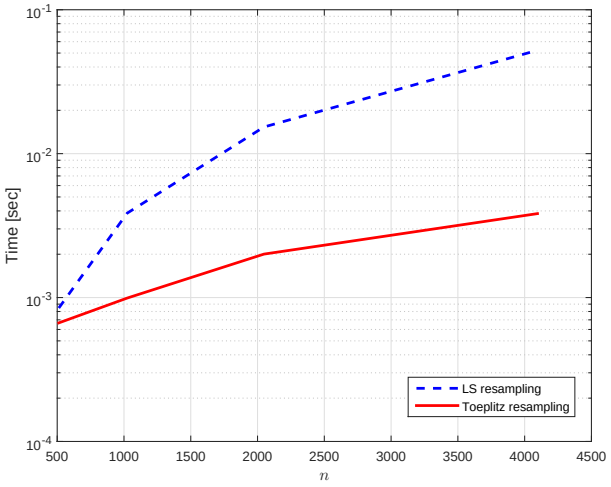


FIG. 10. *Runtime comparison between the LS-based resampling and the Toeplitz-NUFFT-based resampling (Algorithm 5) for* 1D *chirp signals of length* n.

Given the long running time of the 3D NUFFT-based algorithm, we executed above only a single iteration of this algorithm. However, to compare accuracy of our proposed method with that of the 3D NUFFT-based algorithm, we execute next the 3D NUFFT-based algorithm until the error becomes smaller than 10^{-12} or the number of iterations exceeds 100. Due to time constraints, we used only $n = 16,\ 32,\ 64$. The input for each n was the PPFT of a volume of size $n \times n \times n$ of random normally distributed i.i.d. samples with zero mean and unit variance. We implemented the 3D NUFFT-based algorithm with the same preconditioner as in [38]. For $n = 16$ the algorithm took 112 seconds and the resulting error was 7.25×10^{-13}, obtained after 40 iterations; for $n = 32$, it took $2,710$ seconds with an error of 7.5×10^{-13} after 100 iterations; for $n = 64$ it took $22,477$ seconds (more than 6 hours) with an error of 1.61×10^{-12} after 100 iterations. The iterative algorithm described in [38] gives

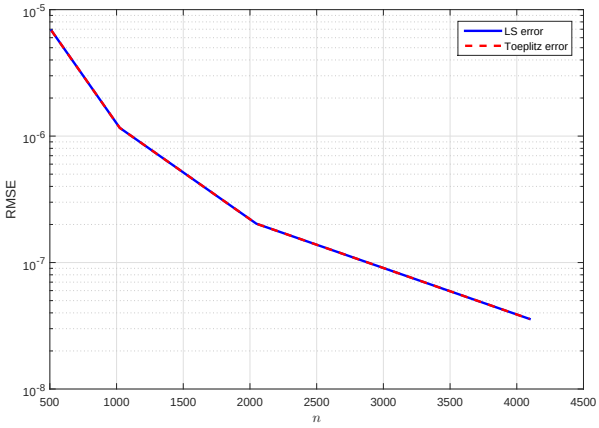


FIG. 11. *Error comparison (according to* (24)*) between the LS-based resampling and the Toeplitz-NUFFT-based resampling (Algorithm* 5*) for* 1D *chirp signals of length* n.

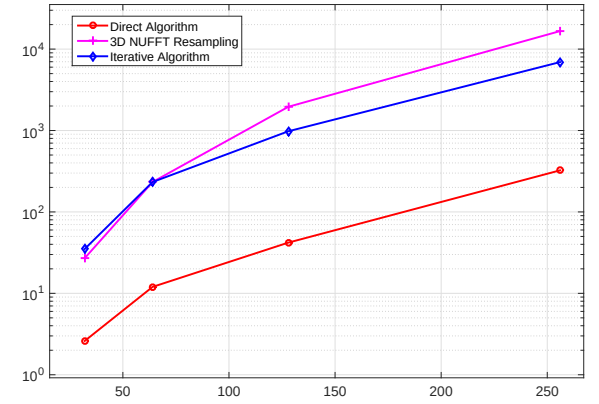
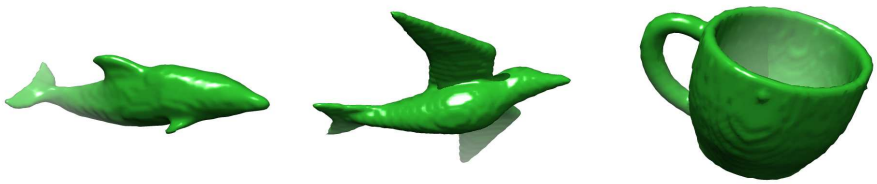


FIG. 12. *Comparison between the running time of Algorithm* 1*, the iterative algorithm* [38]*, and one iteration of the NUFFT-based algorithm* [13].

errors similar to the direct inversion algorithm (Algorithm 1) whose errors appear in Figure 8.

Finally, we tested the direct inversion algorithm on real volumes of different sizes: a dolphin of size $64 \times 64 \times 64$ (Figure 13(a)), a bird of size $128 \times 128 \times 128$ (Figure 13(b)), and a 3D cup of size $256 \times 256 \times 256$ (Figure 13(c)). The volumes were taken from the McGill 3D shape dataset [39]. The PPFT of each of them was used as an input for the inversion algorithm. The results of the inversion appear in Table 1.

5. Conclusions. In this paper, a new algorithm for inverting the 3D PPFT is described. The algorithm processes at each iteration a 2D slice of the input, where each such processing uses only 1D operations. The main component of the algorithm is fast resampling of univariate trigonometric polynomials. The resampling is implemented using a 1D nonuniform Fourier transform together with fast algorithms for



(a) Dolphin, $n = 64$ (b) Bird, $n = 128$ (c) Cup, $n = 256$

FIG. 13. *Test volumes.*

TABLE 1
Results for the volumes of Figure 13.

Name	Size	Time (sec)	Error (24)
Dolphin	$64 \times 64 \times 64$	7.43	1.69×10^{-15}
Bird	$128 \times 128 \times 128$	40.15	3.60×10^{-15}
Cup	$256 \times 256 \times 256$	270	1.25×10^{-14}

Toeplitz matrices. The algorithm is not iterative and requires a fixed amount of time that depends only on the size of the input. Moreover, the algorithm has low memory requirements, allowing to process large 3D datasets in a reasonable time. The performance of the algorithm is demonstrated on volumes as large as $512 \times 512 \times 512$ in double precision.

Appendix A. Algorithms for Toeplitz matrices. The algorithms for Toeplitz matrices used above are summarized in Algorithms 5–9. Algorithm 5 describes fast resampling of univariate trigonometric polynomials. In Algorithm 5, NUFFT_k $(k = 1, 2)$ refers to the type of the NUFFT—see (15) and (16). The parameters -1 and 1 in lines 6, 11, and 13 in Algorithm 5 refer to the sign of $\imath$ in the complex exponent.

Algorithm 5. `ToeplitzResample`: Fast resampling of a univariate trigonometric polynomial.

1: **Input:**
   y Vector of length n.
   f Values of the trigonometric polynomial $\hat{I}$ on the set y.
   x Vector of length n.
2: **Output:**
   g Values of $\hat{I}$ on the set x.
3: $k \leftarrow [-n/2 : n/2 - 1]$ ▷ k is a row vector
4: $c \leftarrow \text{length}(y) \times \text{NUFFT}_1(y, e^{\imath y k(1)}, -1, n)$ ▷ $k(1)$ is the first element of k
5: $(M_1, M_2, M_3, M_4) \leftarrow$ `ToeplitzInv`(c) ▷ Can be computed in the preprocessing step
6: **for** $i = 1, \ldots, 4$ **do**
7: $D_i \leftarrow$`ToeplitzDiag`$(M_i(:, 1), M_i(1, :))$ ▷ Can be computed in the preprocessing step
8: **end for**
9: $v \leftarrow \text{length}\ (y) \times \text{NUFFT}_1(y, f, -1, n)$
10: $\alpha \leftarrow$`ToeplitzInvMul`(D_1, D_2, D_3, D_4, v)
11: $g \leftarrow \text{NUFFT}_2\ (x, 1, n, \alpha)$

Algorithm 6. `ToeplitzInv`: Factorize a symmetric Toeplitz matrix using the Gohberg–Semencul formula.

Input:
c First column (row) of a symmetric Toeplitz matrix.
Output:
M_1, M_2, M_3, M_4 Gohberg–Semencul factorization of the matrix.
$n \leftarrow$length(c)
$e_0 \leftarrow$ zeros$(n, 1)$
$e_n \leftarrow$ zeros$(n, 1)$
$e_0(1) \leftarrow 1$
$e_n(n) \leftarrow 1$
Solve $T_n(c, c)x = e_0$
Solve $T_n(c, c)y = e_n$
Compute M_1 using (7)
Compute M_2 using (8)
Compute M_3 using (9)
Compute M_4 using (10)

Algorithm 7. `ToeplitzDiag`: Diagonalize the circular embedding of a Toeplitz matrix.

Input:
c, r First column/row of the Toeplitz matrix.
Output:
D Diagonal form of $T_n(c, r)$ embedded in a circulant matrix.
$n \leftarrow$ length(c)
$C \leftarrow [c; 0; r(n : -1 : 2)]$
$D \leftarrow$ FFT(C)

Algorithm 8. `ToeplitzInvMul`: Multiply an inverse Toeplitz matrix A^{-1} of size $n \times n$ by a vector v in $\mathcal{O}(n \log n)$ operations, given the diagonal forms of the Gohberg-Semencul factors of A^{-1}.

Input:
D_1, D_2, D_3, D_4 Diagonal forms of the Gohberg-Semencul factors of the inverse Toeplitz matrix.
v Vector of length n.
Output: $u = A^{-1}v$.
$u_1 \leftarrow$`ToeplitzMul`(D_2, v)
$u_1 \leftarrow$`ToeplitzMul`(D_1, u_1)
$u_2 \leftarrow$`ToeplitzMul`(D_4, v)
$u_2 \leftarrow$`ToeplitzMul`(D_3, u_2)
$u \leftarrow u_1 - u_2$

Algorithm 6 computes the Gohberg–Semencul factorization of a symmetric Toeplitz matrix. Algorithm 7 diagonalizes the circulant embedding of a Toeplitz matrix, as a preprocessing for fast multiplication of a Toeplitz matrix by a vector (see Algorithm 9). Algorithm 8 applies the inverse of a Toeplitz matrix to a vector. Finally, Algorithm 9 applies a Toeplitz matrix to a vector.

Algorithm 9. `ToeplitzMul`: Multiply a Toeplitz matrix A of size $n \times n$, whose diagonal factor is D, by a vector v in $\mathcal{O}(n \log n)$ operations.

Input:
- D Diagonal form of A (computed using Algorithm 7).
- v vector of length n.

Output: $u = Av$.

$n \leftarrow \text{length}(v)$
$v \leftarrow [v; \text{zeros}(n, 1)]$
$w \leftarrow \text{FFT}(v)$
$u \leftarrow \text{IFFT}(D. * w)$
$u \leftarrow u(1 : n)$

REFERENCES

[1] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Elad, and M. Israeli, *Fast and accurate polar Fourier transform*, Appl. Comput. Harmoni. Anal., 21 (2006), pp. 145–167.

[2] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli, and Y. Shkolnisky, *A framework for discrete integral transformations* I—*The pseudopolar Fourier transform*, SIAM J. Sci. Comput., 30 (2008), pp. 764–784.

[3] A. Averbuch and Y. Shkolnisky, 3*D Fourier based discrete Radon transform*, Appl. and Comput. Harmoni. Anal., 15 (2003), pp. 33–69.

[4] A. Bermanis, A. Averbuch, and Y. Keller, 3*D symmetry detection and analysis using the pseudo-polar Fourier transform*, Int. J. Comput. Vis., 90 (2010), pp. 166–182.

[5] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, *A superfast algorithm for Toeplitz systems of linear equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1247–1266.

[6] C C Chen, C. Zhu, E. R. White, C. Y. Chiu, M. C. Scott, B. C. Regan, L. D. Marks, Y. Huang, and J. Miao, *Three dimensional imaging of dislocations in a nanoparticle at atomic resolution*, Nature, 496 (2013), pp. 74–77.

[7] B. A. Cipra, *The best of the* 20*th century: Editors name top* 10 *algorithms*, SIAM News, 33 (2000), pp. 1–2.

[8] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14 (1993), pp. 1368–1393.

[9] F. Natterer, *The Mathematics of Computerized Tomography*, Classics in Appl. Math., SIAM, Philadelphia, 2001.

[10] A. F. Ware, *Fast approximate Fourier transforms for irregularly spaced data*, SIAM Revi., 40 (1998), pp. 838–856.

[11] B. P. Fahimian, Y. Zhao, Z. Huang, R. Fung, Y. Mao, C. Zhu, M. Khatonabadi, J. J. DeMarco, S. J. Osher, M. F. McNitt-Gray, and J. Miao, *Radiation dose reduction in medical x-ray CT via Fourier-based iterative reconstruction*, Med. Phys., 40 (2013), 031914.

[12] H. G. Feichtinger, K. Gröchenig, T. Strohmer, et al., *Efficient numerical methods in non-uniform sampling theory*, Numer. Math., 69 (1995), pp. 423–440.

[13] M. Fenn, S. Kunis, and D. Potts, *On the computation of the polar FFT*, Appl. Comput. Harmonic Anal., 22 (2007), pp. 257–263.

[14] J. A. Fessler and B. P. Sutton, *Nonuniform fast Fourier transforms using min-max interpolation*, IEEE Trans. Signal Process, 51 (2003), pp. 560–574.

[15] B. P. Flannery, W. H. Press, S. A. Teukolsky, and W. Vetterling, *Numerical Recipes in C*, Press Syndicate of the University of Cambridge, New York, 1992.

[16] G. Beylkin, *On the fast Fourier transform of functions with singularities*, Appl. Comput. Harmoni. Anal., 2 (1995), pp. 363–381.

[17] I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, J. Complexity, 10 (1994), pp. 411–427.

[18] I. Gohberg and A. Semencul, *On the inversion of finite Toeplitz matrices and their continuous analogs*, Mat. Issled, 2 (1972), pp. 201–233.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., John Hopkins University Press, Baltimaren, MD, 2012.

[20] L. Greengard and J.-Y. Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Rev., 46 (2004), pp. 443–454.

[21] K. Gröchenig, *Reconstruction algorithms in irregular sampling*, Math. Comp., 59 (1992), pp. 181–194.
[22] H. Jiang, C. Song, C. C. Chen, R. Xu, K. S. Raines, B. P. Fahimian, C. H. Lu, T. K. Lee, A. Nakashima, J. Urano, T. Ishikawa, F. Tamanoi, and J. Miao, *Quantitative 3D imaging of whole, unstained cells by using x-ray diffraction microscopy*, Proc., Natl. Acad. Sci. USA, 107 (2010), pp. 11234–11239.
[23] T. Kailath and J. Chun, *Generalized Gohberg-Semencul Formulas for Matrix Inversion*, in The Gohberg Anniversary Collection, Springer, NewYork, 1989, pp. 231–246.
[24] T. Kailath and A. H. Sayed, *Fast Reliable Algorithms for Matrices with Structure*, SIAM, Philadelphia, 1999.
[25] Y. Keller, A. Averbuch, and Y. Shkolnisky, *Algebraically accurate volume registration using Euler's theorem and the 3D pseudo-polar FFT*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2, 2005, pp. 795–800.
[26] G. Kutyniok, W.-Q. Lim, and X. Zhuang, *Digital Shearlet Transforms*, in Shearlets, Springer, NewYork, 2012, pp. 239–282.
[27] G. Kutyniok, M. Shahram, and X. Zhuang, *SHEARLAB: A rational design of a digital parabolic scaling algorithm*, SIAM J. Imaging Sci., 5 (2012), pp. 1291–1332.
[28] W. Lawton, *A new polar Fourier transform for computer-aided tomography and spotlight synthetic aperture radar*, IEEE Trans. Acoustics Speech Signal Process., 36 (1988), pp. 931–933.
[29] E. Lee, B. P. Fahimian, C. V. Iancu, C. Suloway, G. E. Murphy, E. R. Wright, D. Castano Diez, G. J. Jensen, and J. Miao, *Radiation dose reduction and image enhancement in biological imaging through equally sloped tomography*, J. Struct. Biol., 164 (2008), pp. 221–227.
[30] E. Lee, B. P. Fahimian, C. V. Iancu, C. Suloway, G. E. Murphy, E. R. Wright, G. J. Jensen, and J. Miao, *Radiation dose reduction and image enhancement in biological imaging through equally sloped tomography*, J. Struct. Biol., 164 (2008), pp. 221–227.
[31] H. Liu, B. Guo, and Z. Feng, *Pseudo-log-polar Fourier transform for image registration*, IEEE Signal Process. Lett., 13 (2006), pp. 17–20.
[32] Y. Mao, B. P. Fahimian, S. J. Osher, and J. Miao, *Development and optimization of regularized tomographic reconstruction algorithms utilizing equally-sloped tomography*, IEEE Trans. Image Processing, 19 (2010), pp. 1259–1268.
[33] J. Miao, F. Förster, and O. Levi, *Equally sloped tomography with oversampling reconstruction*, Phys. Rev. B, 72 (2005), P. 052103.
[34] J. E. Pasciak, *A Note on the Fourier Algorithm for Image Reconstruction*, Preprint AMD 896, Applied Mathematics Department, Brookhaven National Laboratory, Upton, NY, 1981.
[35] D. Potts, G. Steidl, and M. Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, in Modern Sampling Theory: Mathematics and Applications, Springer, NewYork, 2001, pp. 249–274.
[36] M. Rauth and T. Strohmer, *Smooth approximation of potential fields from noisy scattered data*, Geophysics, 63 (1998), pp. 85–94.
[37] M. C. Scott, C-. C. Chen, M. Mecklenburg, C. Zhu, R. Xu, P. Ercius, U. Dahmen, B. C. Regan, and J. Miao, *Electron tomography at 2.4-angstrom resolution*, Nature, 483 (2012), pp. 444–447.
[38] Y. Shkolnisky and S. Golubev, *Fast Convolution Based Inversion of the Pseudo-Polar Fourier Transform*, submitted, 2014.
[39] K. Siddiqi, J. Zhang, D. Macrini, A. Shokoufandeh, S. Bouix, and S. Dickinson, *Retrieving articulated 3-D models using medial surfaces*, Machine Vision Appl., 19 (2008), pp. 261–275.
[40] J.-L. Starck, E. J. Candès, and D. L. Donoho, *The curvelet transform for image denoising*, IEEE Trans. Image Process., 11 (2002), pp. 670–684.
[41] M. Stewart, *A superfast Toeplitz solver with improved numerical stability*, SIAM J. Matrix Anal. Appl, 25 (2003), pp. 669–693.
[42] C. K. Turnes, D. Balcan, and J. Romberg, *Image deconvolution via superfast inversion of a class of two-level Toeplitz matrices*, in Proceedings of the 19th IEEE International Conference on Image Processing, IEEE, 2012, pp. 3073–3076.
[43] Y. Zhao, E. Brun, P. Coan, Z. Huang, A. Sztrokay, P. C. Diemoz, S. Liebhardt, A. Mittone, S. Gasilov, J. Miao, and A. Bravin, *High-resolution, low-dose phase contrast x-ray tomography for 3D diagnosis of human breast cancers*, Proc. Natl. Acad. Sci. USA, 109 (2012), pp. 18290–18294.