Direct Inversion of the 3D Pseudo-polar Fourier Transform

Amir Averbuch\textsuperscript{1} Gil Shabat\textsuperscript{1*} Yoel Shkolnisky\textsuperscript{2}
\textsuperscript{1}School of Computer Science, Tel Aviv University, Israel
\textsuperscript{2}School of Applied Mathematics, Tel Aviv University, Israel

Abstract

The pseudo-polar Fourier transform is a specialized non-equally 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 non-uniform 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 non-equally spaced Fourier transform can be inverted, the only known algorithms are iterative. The convergence speed of these algorithms as well as their accuracy are difficult to control, as they depend both on the sampling geometry as well as on the unknown reconstructed object. In this paper, we present a direct inversion algorithm for the three-dimensional pseudo-polar Fourier transform. The algorithm is based only on one-dimensional resampling operations, and is shown to be significantly faster than existing iterative inversion algorithms.

Keywords: 3D pseudo polar Fourier transform, Radon transform, Unequally spaced FFT, Polar Fourier transform, Toeplitz matrices

AMS subject classification: 65T50, 44A12, 92C55

1 Introduction

The fast Fourier transform (FFT) is one of the most commonly used algorithms that has far reaching implications in science and technology [6]. 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. This can be implemented using one of the available non-equally spaced FFTs (NUFFTs) [7, 9, 14, 13, 35, 19]. 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, 5, 40, 10, 21, 29, 28, 26, 4, 24].

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. 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, which is self-adjoint and thus can be easily inverted, the NUFFT, except for very special cases, is not self-adjoint, and thus, more elaborated inversion schemes are required. In particular, although the transform that evaluates the Fourier

*Corresponding author: gil@eng.tau.ac.il
transform on the polar grid is formally invertible \[3\], the condition number of this transformation excludes such inversion in practice.

Since the polar sampling geometry of Fourier space is natural for various applications such as computerized tomography \[8\] or electron microscopy \[27, 3\], it is sometimes possible to overcome the aforementioned difficulties by replacing the polar frequency sampling grid by 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 its samples lie on equally-spaced rays. We refer to the transformation, which samples the Fourier transform on the pseudo-polar grid, as the pseudo-polar Fourier transform (PPFT). Both the forward and inverse transformations admit efficient and stable algorithms. Iterative algorithms, which are used to invert the pseudo-polar Fourier transforms, can be found in \[2, 20, 11, 36\]. Since the pseudo-polar Fourier transform is ill-conditioned, inverting using these algorithms requires preconditioning. Several preconditioning approaches have been suggested such as using Voronoi weights \[12\] or a perconditioner derived from the Jacobian \[38\]. In particular, it is shown in \[38\] that under the appropriate preconditioning, the condition number of the pseudo-polar Fourier transform is small. Therefore, it can be inverted using iterative inversion schemes. Nevertheless, such an iterative inversion requires to compute the pseudo-polar Fourier transform and its adjoint in each iteration, which as mentioned above, is too expensive for large problems. Even if the pseudo-polar Fourier transform 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 dozens, which may be too slow for large scale problems. Moreover, the accuracy of the reconstructed object in 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.

In this paper, we present a direct algorithm for inverting the three-dimensional pseudo polar Fourier transform. It generalizes the two-dimensional 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 error of the inversion algorithm is guaranteed to be of the order of machine precision, and since most of steps in the algorithm are implemented in-place, its memory consumption is sufficiently low so it can be applied 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 2.1, we provide the required background on the pseudo-polar grid and the pseudo-polar Fourier transform. In Section 3, we give a detailed description of the direct inversion algorithm for the three-dimensional pseudo-polar transform. Numerical results and comparison 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 required mathematical background. In Section 2.1 we revisit the pseudo-polar Fourier transform. In Section 2.2 we describe a fast algorithm for solving Toeplitz systems of equations. This algorithm is used in Section 2.3 as a fast algorithm for resampling univariate trigonometric polynomials.

### 2.1 Pseudo-polar Fourier transform

The 3D pseudo-polar grid, denoted $\Omega_{pp}$, is defined by

$$
\Omega_{pp} \triangleq \Omega_{ppx} \cup \Omega_{ppy} \cup \Omega_{ppz},
$$

(1)
Figure 1: The three sectors $Ω_{ppx}$, $Ω_{ppy}$ and $Ω_{ppz}$ that form the 3D pseudo polar grid.

where

\[
Ω_{ppx} \triangleq \{ (k, -\frac{2l}{n} k, -\frac{2j}{n} k) : l, j = -n/2, \ldots, n/2, k = -\frac{m}{2}, \ldots, \frac{m}{2} \}, \\
Ω_{ppy} \triangleq \{ (-\frac{2l}{n} k, k, -\frac{2j}{n} k) : l, j = -n/2, \ldots, n/2, k = -\frac{m}{2}, \ldots, \frac{m}{2} \}, \\
Ω_{ppz} \triangleq \{ (-\frac{2l}{n} k, -\frac{2j}{n} k, k) : l, j = -n/2, \ldots, n/2, k = -\frac{m}{2}, \ldots, \frac{m}{2} \},
\]

with $m = qn + 1$ for an even $n$ and some positive integer $q$. We denote a specific point in $Ω_{ppx}$, $Ω_{ppy}$ and $Ω_{ppz}$ by $Ω_{ppx}(k,l,j)$, $Ω_{ppy}(k,l,j)$ and $Ω_{ppz}(k,l,j)$, respectively. The psuedo-polar grid is illustrated in Fig. 1 for $n = 4$ and $q = 1$. As can be seen from Fig. 1 the pseudo-polar grid consists of equally spaced samples along rays, where different rays are equally spaced and not equally angled. 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

\[
\hat{I}(ω_x, ω_y, ω_z) = \sum_{u,v,w=-n/2}^{n/2-1} I(u,v,w) e^{2πi(ω_x u + ω_y v + ω_z w)/m},
\]

were as before $m = qn + 1$ for an even $n$ and a positive integer $q$. The three-dimensional pseudo-polar Fourier transform (PPFT) is defined as samples of the discrete time Fourier transform of Eq. 3 on the pseudo-polar grid $Ω_{pp}$. Specifically,

\[
\hat{I}_{Ω_{ppx}} = \hat{I} \left( k, -\frac{2l}{n} k, -\frac{2j}{n} k \right), \quad \hat{I}_{Ω_{ppy}} = \hat{I} \left( -\frac{2l}{n} k, k, -\frac{2j}{n} k \right), \quad \hat{I}_{Ω_{ppz}} = \hat{I} \left( -\frac{2l}{n} k, -\frac{2j}{n} k, k \right).
\]

Where $Ω_{ppx}$, $Ω_{ppy}$ and $Ω_{ppz}$ are defined in Eq. 2 The parameter $q$ in Eq. 1 and Eq. 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 three-dimensional discrete Radon transform based on the pseudo-polar Fourier transform, one must use $q \geq 3$.

The pseudo-polar grid appeared in the literature several times under different names. It was originally introduced by [34] under the name “Concentric Squared 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] that use equally-sloped tomography for radiation dose reduction. Other image processing applications that use the pseudo-polar grid include synthetic aperture radar (SAR) imaging [27], Shearlets [26, 25], registration [31] and denoising [39], to name a few.

Recently, [38] proposed fast iterative inversion algorithms for the 2D and 3D pseudo-polar Fourier transforms, which are based on the well-known convolution structure of their Gram operators, combined with a preconditioned conjugate gradients [17] solver. The convolution structure of the transform allows 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 on the number of iterations and on the size of the data. The algorithm presented in this paper, which has a low memory requirements, achieves high accuracy and it 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, and it is based on the well-known circulant embedding approach \[16, 23\]. We denote by \(T_n(c, r)\) an \(n \times n\) Toeplitz matrix whose first column and row are \(c\) and \(r\), respectively. However, since as we see below, we are interested only in symmetric matrices, then we have \(c \equiv r\).

Circulant matrices are diagonalized by the Fourier matrix. Hence, a circulant matrix \(C_n\) can be written \(C_n = W_n^* D_n W_n\), 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 \(W_n(j, k) = \frac{1}{\sqrt{n}} e^{2\pi i j k/n}\). 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 for 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 result vector.

Next, assume that \(A_n\) is invertible. The Golberg–Semencul formula \[10, 15\] provides an explicit representation by \(A_n^{-1}\) as

\[
A_n^{-1} = \frac{1}{x_0} (M_1 M_2 - M_3 M_4),
\]

where

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

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

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

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

The vector \(x = [x_0, \ldots, x_{n-1}]\) is the solution of \(A_n x = e_0\), and the vector \(y = [y_0, \ldots, y_{n-1}]\) is the solution of \(A_n y = e_{n-1}\), with \(e_0 = [1, 0, \ldots, 0]^T\) and \(e_{n-1} = [0, \ldots, 0, 1]^T\). The matrices \(M_1, M_2, M_3,\) and \(M_4\) have a 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 2n terms. 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 $M_4$ are 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 a Toeplitz matrix that uses FFT. Algorithm 8 lists the function ToeplitzInvMul, which multiplies efficiently an inverse Toeplitz matrix, given in its diagonal form, by a vector. The latter function uses ToeplitzMul, listed in Algorithm 9, which multiplies efficiently 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 $\{f(y_j)\}$. We want to estimate the values of $f$ at points $\{x_j\}_{j=1}^M, x_j \in [-\pi, \pi]$. This can be formulated by first solving

$$
\min_{\alpha} \sum_{j=1}^N \left| f(y_j) - \sum_{k=-n/2}^{n/2-1} \alpha_k e^{iky_j} \right|^2 \tag{10}
$$

for the coefficients $\alpha$ followed by evaluating

$$
f(x_i) = \sum_{k=-n/2}^{n/2-1} \alpha_k e^{ikx_i}. \tag{11}
$$

In matrix notation Eq. (10) is written

$$
\min_{\alpha} \| f - A\alpha \|_2, \tag{12}
$$

where the entries of the matrix $A$ are given by $a_{jk} = e^{iky_j}$ and the coordinates of the vector $f$ are $f_j = f(y_j)$. Direct solution of the coefficient vector $\alpha$ requires $O(n^3)$ operations (assuming $N$ and $M$ are of size $O(n)$). Obviously, solving for $\alpha$ also depends on the condition number of $A$.

We next present a faster algorithm for computing $f(x_i)$ which exploits the Toeplitz structure of $A^*A$ together with the non-equally spaced FFT (NUFFT) \cite{7,9,13,35,19}. The resulting algorithm has complexity of $O(n \log n + n \log 1/\epsilon)$ operations, where $\epsilon$ is the accuracy of the computations, in addition to a preprocessing step that takes $O(n^2)$ operations.

NUFFT type-I \cite{13} is defined by

$$
f_k = \frac{1}{n_j} \sum_{j=1}^{n_j} c_j e^{\pm ikx_j}, \quad k = -n/2, \ldots, n/2 - 1. \tag{13}
$$

The NUFFT type-II is defined by

$$
c_j = \sum_{k=-n/2}^{n/2-1} f_k e^{\pm ikx_j}, \quad j = 1, \ldots, n_j. \tag{14}
$$

Both types can be approximated to a relative accuracy $\epsilon$ in $O(n \log n + n \log 1/\epsilon)$ 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 Eq. (12) 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 $O(n \log n + n \log 1/\epsilon)$ operations.
To solve the least-squares problem of Eq. \((12)\) we form the normal equations
\[
A^*A\alpha = A^*f. 
\] (15)

The right-hand side in Eq. \((15)\) 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 Toeplitz structure that encodes all its entries, is computed efficiently by applying NUFFT type-I to the vector \(w\) whose entries are \(w_j = e^{-i\pi y_j^2/2}\). Computation of \((A^*A)^{-1}\) takes \(O(n^2)\) operations using the Durbin-Levinson algorithm \([30]\) and its application to a vector takes \(O(n\log n)\) operations using the Gohberg-Semencul formula \([22]\), as was described in Section 2.2. This procedure is described in details in \([2]\). Since \((A^*A)^{-1}\) depends only on \(n\), it can be precomputed. Therefore, solving for \(\alpha\) in Eq. \((15)\) takes \(O(n\log n + n\log 1/\epsilon)\) operations and computing \(f(x_i)\) in Eq. \((11)\) takes additional \(O(n\log n + n\log 1/\epsilon)\) operations using NUFFT Type-II. The entire resampling algorithm is described in Algorithm \(5\).

3 Direct inversion of the 3D PPFT

Given the PPFT \(\hat{I}_{pp}\) 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 into an intermediate Cartesian grid. Specifically, we resample the trigonometric polynomial \(\hat{I}\) of Eq. \((3)\) from the grid \(\Omega_{pp}\) in Eq. \((1)\) to the frequency grid
\[
\Omega_c = \{(qu, qv, qw) : u, v, w = -n/2, ..., n/2\}. 
\] (16)

The second step of our algorithm recovers \(I\) from the samples of \(\hat{I}\) on \(\Omega_c\). Note that this cannot be implemented by inverse FFT.

3.1 Resampling the pseudo-polar grid to the grid \(\Omega_c\)

We start by describing the procedure for resampling \(\hat{I}\) from \(\Omega_{pp}\) to \(\Omega_c\). It is based on an “onion peeling” approach, which resamples at iteration \(k, k = n/2, \ldots, 0\), points in \(\Omega_{pp}\) with “pseudo-radius” \(k\) to points in \(\Omega_c\) that lie on a plane. When the iteration corresponds to \(k = 0\) is completed, the values of \(\hat{I}\) are computed on all points in the grid \(\Omega_c\). The sets \(\Omega_c^{(k)} \subseteq \Omega_c, k = n/2, \ldots, 0\) consist of the frequencies on which \(\hat{I}\) has been resampled until (and including iteration \(k\)) are defined. 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
\[
\Theta^{(k)} = \Theta^{(k)}_{+z} \cup \Theta^{(k)}_{-z} \cup \Theta^{(k)}_{+y} \cup \Theta^{(k)}_{-y} \cup \Theta^{(k)}_{+z} \cup \Theta^{(k)}_{-z},
\] (17)
and for \(j, l = -k, \ldots, k\)
\[
\Theta^{(k)}_{+z} = \{(qj, qk, ql)\}, \quad \Theta^{(k)}_{-z} = \{(-qj, qk, ql)\},
\]
\[
\Theta^{(k)}_{+y} = \{(qj, qk, ql)\}, \quad \Theta^{(k)}_{-y} = \{(qj, -qk, ql)\},
\]
\[
\Theta^{(k)}_{+z} = \{(qj, ql, qk)\}, \quad \Theta^{(k)}_{-z} = \{(qj, ql, -qk)\}.
\] (18)

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

For subsequent iterations, we use an example to accompany and clarify the formal description. We use a grid
that corresponds 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 \( \Theta^{(k)}_{\pm x}, \Theta^{(k)}_{\pm y}, \text{ and } \Theta^{(k)}_{\pm z} \), we only explain the process for the grid \( \Theta^{(k)}_{\pm x} \). In Fig. 2a (green) solid squares correspond to points from \( \Omega^{(k)}_{\pm x} \) on which \( \hat{I} \) was already been evaluated (in previous iterations), (red) circles correspond to points from \( \Omega_{ppx}^{(k)} \) with a fixed \( k = n/2 - 1 \), and (blue) dots correspond to points of \( \Theta^{(k)}_{\pm x} \) on which we want to evaluate \( \hat{I} \). The frequencies in \( \Theta^{(k)}_{\pm x} \) are point in \( \mathbb{R}^3 \), however, as can be seen from Eq. 18, they all lie on the same plane, therefore, we can depict them as an image whose axes are \( i \) and \( j \) from Eq. 18.

The first step, depicted in Fig. 2b, consists of resampling the points of \( \Omega^{(k+1)}_{c} \) (solid green squares) to the same spacing as the pseudo-polar points. The result of this resampling is denoted by patterned (yellow) squares.

Next, as depicted in Fig. 3a for all columns, we use the latter resampled points together with the points in the pseudo-polar grid at the same column to resample \( \hat{I} \) to points of \( \Theta^{(k)}_{\pm x} \). Figures 3a and 3b depict the resampling of such one column. Note that the values of \( \hat{I} \) after the resampling step, which are depicted as patterned circles in 3b, are not yet the values of \( \hat{I} \) on the point in \( \Theta^{(k)}_{\pm x} \). Up to here, we only applied resampling to the columns. We repeat this for all the columns - see Fig. 4a for resampling of another column. This resulted in the grid of Fig. 4b.

Next, we apply one-dimensional resampling to each row, by using the resampled points from the previous steps together with the points of \( \Omega^{(k)}_{c} \) (Fig. 5a) to recover the samples of \( \Theta^{(k)}_{\pm x} \) (Fig. 5b).

The values of \( \hat{I} \) on a two-dimensional slice of \( \Theta^{(k)}_{x} \) are restored from the given \( \Omega^{(k)}_{ppx} \) by the application of Algorithm 2. The output is a two-dimensional data sampled on the grid \( \Omega_{c} \) of Eq. 16. As an example, consider an image \( I \) of size \( n \times n \times n \). Let \( j \) be the number of rows that were already resampled. Suppose that \( n = 8 \) and \( j = 1 \). In this case, an image of size \( 9 \times 9 \) has to be resampled. Figure 2 illustrates this case, where \( \alpha = \frac{n/2-j}{n/2} = 0.75 \). The green squares represent points which were already resampled (at the previous iteration, \( j = 0 \)), the red circles are the known points from the pseudo polar grid and the blue dots are the points in the set \( \Omega_{c} \) which \( \hat{I} \) should be resampled to. The first two calls to the resample function (lines 8, 9 in Algorithm 2) align the points that were already resampled (green squares) with the pseudo polar grid (red circles). In the case \( j = 1 \), it means that the first and the last rows were resampled. The resampled first and last rows are indicated by the patterned yellow squares. This is illustrated in Fig. 2b. The next two steps from line 13 to line 20 in Algorithm 2 resample the columns and the rows from the pseudo polar grid to the Cartesian grid. First, one dimensional resampling is applied to each column (lines 13 − 16), so that the points indicated by the red circles are resampled along the columns to be on the
Figure 3: The grid before and after the applications of lines 13 – 16 in Algorithm 2

Figure 4: The grid before and after additional further applications of lines 13 – 16 in Algorithm 2
**Algorithm 1:** Inversion of the 3D pseudo-polar Fourier transform based on onion-peeling procedure.

**Input:** Samples of the 3D pseudo-polar Fourier transform $\hat{I}_{Ω_{ppx}}, \hat{I}_{Ω_{ppy}}, \hat{I}_{Ω_{ppz}}$ each of size $(q_n + 1) \times (n + 1) \times (n + 1)$

**Output:** Image $I$ of size $n \times n \times n$

1. $\hat{I}_D \leftarrow \text{zeros}(n + 1, n + 1, n + 1)$
2. for $j = 1, \ldots, n/2$ do
   3. $\hat{I}_D(j, :, :) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppx}}(q(j - 1) + 1, :, :], \hat{I}_D(j, :, :) - 1)$ (Algorithm 2)
   4. $\hat{I}_D(n + 2 - j, :, :) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppx}}(q_n - q(j - 1) + 1, n + 1 : -1 : 1), \hat{I}_D(n + 2 - j, :, :) - 1)$
   5. $\hat{I}_D(:, j, :) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppy}}(q(j - 1) + 1, :, :], \hat{I}_D(:, j, :) - 1)$
   6. $\hat{I}_D(:, n + 2 - j, :) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppy}}(q_n - q(j - 1) + 1, n + 1 : -1 : 1), \hat{I}_D(:, n + 2 - j, :) - 1)$
   7. $\hat{I}_D(:, :, j) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppz}}(q(j - 1) + 1, :, :], \hat{I}_D(:, :, j) - 1)$
   8. $\hat{I}_D(:, :, n + 2 - j) \leftarrow \text{recover2d}(\hat{I}_{Ω_{ppz}}(q_n - q(j - 1) + 1, n + 1 : -1 : 1), \hat{I}_D(:, :, n + 2 - j) - 1)$
9. end for
10. $\hat{I}_D(n/2 + 1, n/2 + 1, n/2 + 1) \leftarrow \hat{I}_{Ω_{ppx}}(3n/2 + 1, n/2 + 1, n/2 + 1)$ (center point)
11. $I \leftarrow \text{InvDecimatedFreq} \hat{I}_D$ (recover $I$ from $\hat{I}_D$) (Algorithm 4)

---

**Figure 5:** The grid before and after the applications of lines 17 – 20 in Algorithm 2 (row resampling)
same rows as the points indicated by the blue dots. This is illustrated in Fig. 4 where the points indicated by the red circles have been resampled. Then, another one dimensional resampling is applied to the rows (lines 17–20), computing the values exactly on the blue dots.

**Algorithm 2**: recover2d: This function recovers a 2D slice of the 3D pseudo polar grid and recovers it to Cartesian grid

```plaintext
Input: \( \hat{U}_{\Omega_{pp}} \) - Slice of the 3D pseudo-polar Fourier transform of size \((n + 1) \times (n + 1)\).
Input: \( \hat{U}_D \) - Already resampled slice of \( \hat{I}_D \) from Algorithm 1 of size \((n + 1) \times (n + 1)\).
Input: \( j \) - An integer that indicates the peeling procedure stage.

Output: result of size \((n + 1) \times (n + 1)\) (resampled part, \( I_D \) from Algorithm 1)

1. \( \alpha \leftarrow (n/2 - j)/(n/2) \)
2. \( m \leftarrow qn + 1 \)
3. if \( j \neq 0 \) then
4. \( y_1 \leftarrow [-n/2 : n/2] \times -2 \times q \times \pi/m \)
5. \( x_1 \leftarrow [-n/2 : n/2] \times -2 \times q \times \alpha \times \pi/m \)
6. \( C \leftarrow \text{zeros}(n + 1, n + 1) \)
7. for \( k = 1, \ldots, n + 1 \) do
8. \( C(k, :) \leftarrow \text{ToeplitzResample}(\hat{U}_{\Omega_{pp}}(k, 1 : n + 1), y_1, x_1) \) (Algorithm 5)
9. \( C(n + 2 - k, :) \leftarrow \text{ToeplitzResample}(\hat{U}_{\Omega_{pp}}(n + 2 - k, 1 : n + 1), y_1, x_1) \)
10. end for
11. \( x \leftarrow [-n/2 - j : (n/2 - j)] \times -2 \times q \times \pi/m \)
12. \( y \leftarrow \left([-n/2 : -n/2 + j - 1] \cup [-n/2 : n/2]\right) \times m \cup [n/2 - j + 1 : n/2] \times -2 \times q \times \pi/m \)
13. \( R_1 \leftarrow \text{zeros}(n + 1 - 2j, n + 1) \)
14. for \( k = 1, \ldots, n + 1 \) do
15. \( R_1 \leftarrow \text{ToeplitzResample}(\left[C(1 : j, k) \cup \hat{U}_{\Omega_{pp}}(:, k) \cup C(n - j + 2 : n + 1, k)\right], y, x) \)
16. end for
17. \( R_2 \leftarrow \text{zeros}(n + 1 - 2j, n + 1 - 2j) \)
18. for \( k = 1, \ldots, n + 1 - 2j \) do
19. \( R_2 \leftarrow \text{ToeplitzResample}(\left[\hat{U}_D(k + j, 1 : j) \cup R_1(k, :) \cup \hat{U}_D(k + j, n - j + 2 : n + 1)\right], y, x) \)
20. end for
21. \( \text{result} \leftarrow \hat{U}_D \)
22. \( \text{result}(1 + j : n + 1 - j, 1 + j : n + 1 - j) \leftarrow R_2 \)
23. else
24. \( \text{result} \leftarrow \hat{U}_{\Omega_{pp}} \)
25. end if

All one-dimensional resampling operations are implemented using the function ToeplitzResample in Algorithm 5

### 3.2 Recovering \( I \) from \( \hat{I} \) on \( \Omega_c \)

The second step in Algorithm 1 (line 11) recovers the volume \( I \) from the samples of \( \hat{I} \) on \( \Omega_c \) (Eq. 16). Define the operator \( F_D : \mathbb{C}^n \rightarrow \mathbb{C}^{n+1} \), by

\[
(F_D u)(k) = \sum_{j=-n/2}^{n/2-1} u(j) e^{-2\pi i j k/m}, \quad k = -n/2, \ldots, n/2, \quad m = qn + 1. \tag{19}
\]

For volume \( I \) of \( n \times n \times n \), 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

\[
\tilde{I} = F_D^{(x)} F_D^{(y)} F_D^{(z)} I. \tag{20}
\]
Figure 6: The maximal condition number of $F_D$ (Eq. 19) obtained from the application of Algorithm 4 to various sizes $n$.

From the definition of $F_D$ in Eq. 19, the definition of $\hat{I}$ in Eq. 3, and the definition of $\Omega_c$ in Eq. 16, we conclude that $\tilde{I}$ is equal to the samples of $\hat{I}$ on the grid $\Omega_c$. Thus, if we apply the inverse of $F_D$ to the $x, y, \text{and } z$ dimensions of $\tilde{I}$ (separately), we recover $I$ from the samples of $\hat{I}$ on $\Omega_c$. The inversion of the operator $F_D$ is described in [2]. Specifically, in our case, for $m = qn + 1$,

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

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

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

and its inverse is applied as described in Section 2.2. The image $I$ is recovered by the application of $(F_D^* F_D)^{-1} F_D^*$ to each dimension of $\tilde{I}$. The maximal condition number of $F_D$, which was obtained while applying Algorithm 2 to various sizes $n$, is illustrated in Fig. 6. $F_D^* w$ is applied efficiently by adding $q - 1$ zeros between every two samples of $w$, applying the inverse FFT, while keeping the $n$ central elements of the resulting vector as was described in Algorithm 3. The entire algorithm for recovering the volume from its samples on $\Omega_c$ is described in Algorithm 4.
Algorithm 3: AdjFDecimated. Application of $F^*_D$ to a vector

**Input:** $y$ - Input vector of odd length, $q$ - integer (decimation factor)

**Output:** $x = F^*_D y$

1. $n \leftarrow \text{length}(y) - 1$
2. $m \leftarrow qn + 1$
3. $z \leftarrow \text{zeros}(m, 1)$
4. $z(1 : q : m) \leftarrow y$
5. $l \leftarrow \text{length}(z)$
6. **if** $l$ is odd **then**
   7. $p_f \leftarrow \text{floor}(l/2)$
   8. $p_c \leftarrow \text{ceil}(l/2)$
   **else**
   10. $p_f \leftarrow l/2$
   11. $p_c \leftarrow p_f$
   **end if**
13. $z_i \leftarrow z([p_f + 1 : l : p_f])$
14. $z_i,\text{ifft} \leftarrow \text{IFFT}(z_i)$
15. $x \leftarrow m \cdot z_i,\text{ifft}([p_c + 1 : l \ 1 : p_c])$
16. $x \leftarrow x(n + 1 : 2n)$

3.3 Complexity Analysis

The inversion Algorithm 1 consists of two steps: Resampling (lines 1-10) and recovering $I$ from $\hat{I}$ on $\Omega_c$ (line 11). The first step, resampling, which utilizes Toeplitz matrices (Algorithm 3), takes $O(n \log n)$ operations when preprocessing took place. This function (Toeplitz-Resampling) is called $O(n)$ times in the recover2d function (Algorithm 2), which in its turn is called also $O(n)$ times by Algorithm 1. A total of $O(n^3 \log n)$ operations for the resampling step.

The second step, recovering $I$ from $\hat{I}$ on $\Omega_c$, takes $O(n^3 \log n)$ operations. This leads to a total computational complexity of $O(n^3 \log n)$ operations. If the preprocessing is done in run time, the total complexity becomes $O(n^4)$ when the Durbin-Levinson algorithm is used for the inversion of a Toeplitz matrix. The computational complexity can be reduced from $O(n^4)$ operations to $O(n^3 \log n)$ by solving $T_n(c)x = e_0$ as described in [22].

The total storage requirement 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 volumes of size $n \times n \times n$ where $n = 32, 64, \ldots, 512$. All experiments were executed on a Linux machine with two Intel Xeon processors (CPU X5560) running at 2.8GHz that has 8 cores and 96GB of RAM.

For testing the resampling procedure, we compared it to a straightforward least-squares solution described in Eq. 12 for finding the coefficients $\alpha$ of the trigonometric polynomial, followed by direct evaluation of the trigonometric polynomial at new points. This approach has computational complexity of $O(n^4)$, but its implementation in MATLAB is highly optimized. The second tested resampling Algorithm 3 is the fast Toeplitz-NUFFT algorithm described in Section 2.3, which has computational complexity of $O(n^3 \log n + n^3 \log 1/\epsilon)$. Both algorithms use a preprocessing step which is excluded from the reported running time. Both achieve machine precision accuracy, where the error between the original volume $I[k, m, n]$ and the reconstructed volume $\hat{I}[k, m, n]$ is measured by

$$E(I, \hat{I}) = \sqrt{\frac{\sum_{k,m,n} |I[k, m, n] - \hat{I}[k, m, n]|^2}{\sum_{k,m,n} |I[k, m, n]|^2}}.$$
Algorithm 4: InvDecimatedFreq: This procedure inverts the decimated frequencies and restores $I$ from $\hat{I}$.

**Input:** $\hat{I}$ - Matrix of size $(n+1) \times (n+1) \times (n+1)$.

**Output:** $I$ - Image of size $n \times n \times n$.

1: $c \leftarrow \text{zeros}(n, 1)$
2: $m \leftarrow qn + 1$
3: for $k=-n/2, \ldots, n/2$ do
4:   for $l=-n/2, \ldots, n/2$ do
5:     $c(k+n/2+1) \leftarrow c(k+n/2+1) + \exp(q\pi il(-n/2-k)/m)$
6:   end for
7: end for
8: $(M_1, M_2, M_3, M_4) \leftarrow \text{ToeplitzInv}(c)$ (Algorithm 6)
9: $D_i \leftarrow \text{ToeplitzDiag}(M_i), i = 1, \ldots, 4$ (Algorithm 7)
10: $\text{FirstInv} \leftarrow \text{zeros}(n+1, n+1, n)$
11: $\text{SecondInv} \leftarrow \text{zeros}(n+1, n, n)$
12: $I \leftarrow \text{zeros}(n, n, n)$
13: for $k=1, \ldots, n+1$ do
14:   for $l=1, \ldots, n+1$ do
15:     $v \leftarrow I(k,l,:)$
16:     $v \leftarrow \text{AdjFDecimated}(v)$ (Algorithm 3)
17:     $u \leftarrow \text{ToeplitzInvMul}(D_1, D_2, D_3, D_4, v)$ (Algorithm 8)
18:     $\text{FirstInv}(k,l,:) \leftarrow u$
19:   end for
20: end for
21: for $k=1, \ldots, n+1$ do
22:   for $l=1, \ldots, n$ do
23:     $v \leftarrow \text{FirstInv}(k,:,l)$
24:     $v \leftarrow \text{AdjFDecimated}(v)$
25:     $u \leftarrow \text{ToeplitzInvMul}(D_1, D_2, D_3, D_4, v)$
26:     $\text{SecondInv}(k,l,:) \leftarrow u$
27:   end for
28: end for
29: for $k=1, \ldots, n$ do
30:   for $l=1, \ldots, n$ do
31:     $v \leftarrow \text{SecondInv}(:,k,l)$
32:     $v \leftarrow \text{AdjFDecimated}(v)$
33:     $u \leftarrow \text{ToeplitzInvMul}(D_1, D_2, D_3, D_4, v)$
34:     $I(:,k,l) \leftarrow u$
35: end for
36: end for
Running times for both methods for volumes of sizes $n \times n \times n$, where $n = 32, 64, ..., 512$ are shown in Fig. 7. Figure 7a corresponds to volumes consisting of random independent samples from a normal distribution with zero mean and unit variance. Figure 7b corresponds to volumes generated by sampling the indicator function of a ball of radius 1 at $n$ equally spaced points along each dimension on the grid $[-1, 1] \times [-1, 1] \times [-1, 1]$.

![Computation time vs image size](image1)

(a) Random volume

![Computation time vs image size](image2)

(b) Ball volume

Figure 7: Runtime comparison between least squares resampling and Toeplitz+NUFFT resampling (Algorithm 5) for volumes of size $n \times n \times n$. (a) Random volumes with normally distributed samples of zero mean and unit variance. (b) Indicator function of a ball of radius 1 sampled at equally spaced points on the grid $[-1, 1] \times [-1, 1] \times [-1, 1]$.

Next, we compare between the direct inversion Algorithm 1, the iterative algorithm described in [38] and a single iteration of a similar algorithm that computes the inversion operator directly in 3D ([12]). The direct implementation was implemented by 3D NUFFT. For high precision, several iterations are needed. The results are illustrated in Fig. 8. It shows the total running time of the direct onion peeling inversion algorithm is faster than the iterative algorithm [38] and faster than a single iteration of the 3D NUFFT based algorithm that typically requires a few dozens of iterations to converge. Two results for running $n = 512$ are missing (Fig. 8) because it was impractical to process using these methods (Iterative and 3D NUFFT).
Figure 8: Comparison between the iterative algorithm ([38]), direct 3D resampling algorithm ([12], also referred as 3D NUFFT) and the direct algorithm (Algorithm 1).

5 Conclusions

In this paper, we described a new algorithm for inverting the 3D PPFT. The algorithm processes each 2D slice separately, where each 2D is processed by the application of 1D operations. The main component of the algorithm is 1D resampling of a trigonometric polynomial. The resampling is done by the application of a non-uniform Fourier transform and by fast operations involving Toeplitz matrices. The algorithm operates directly. It does not operate iteratively in order to converge, but rather runs in a fixed time with no dependency on the actual data. Moreover, the algorithm has low memory consumption, allowing to process large 3D datasets in a reasonable time. The performance of the algorithm is demonstrated on $512 \times 512 \times 512$ image in double precision.
A Toeplitz solvers algorithms

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

**Input:** \( y \) - vector of locations of known samples.

\( f \) - values of \( \hat{I} \) on the set \( y \).

\( x \) - vector of locations of unknown samples.

**Output:** \( g \) - values of \( \hat{I} \) on the set \( x \).

1: \( k \leftarrow [-n/2 : n/2 - 1] \) (row vector)
2: \( c \leftarrow \text{length}(y) \times \nuFFT_1(y, \exp(\imath y k(1)), -1, n) \) (\( k(1) \) - The first element of \( k \))
3: \( (M_1, M_2, M_3, M_4) \leftarrow \text{ToeplitzInv}(c, c) \) (Can be done in the preprocessing step)
4: \( D_i \leftarrow \text{ToeplitzDiag}(M_i(:, 1), M_i(1, :)), i = 1, 2, 3, 4 \) (Can be done in the preprocessing step)
5: \( v \leftarrow \text{length}(y) \times \nuFFT_1(y, f, -1, n) \)
6: \( \alpha \leftarrow \text{ToeplitzInvMul}(D_1, D_2, D_3, D_4, v) \)
7: \( g \leftarrow \nuFFT_2(x, 1, n, \alpha) \)

\( \nuFFT_k \) (\( k = 1, 2 \)) refers to the type of the NUFFT - see Eq. 13 and Eq. 14. The \(-1, 1\) in the parameters (lines 2, 5, 7 in Algorithm 5), refer to the sign of \( \imath \) in the complex exponent. The function ToeplitzInv is described...
in Algorithm 6 and performs the Gohberg-Semencul factorization of a symmetric Toeplitz matrix.

**Algorithm 6: ToeplitzInv.** Factorize a symmetric Toeplitz matrix using the Gohberg-Semencul factorization.

**Input:** \( c \) - First column (row) of a Toeplitz matrix.

**Output:** \( M_1, M_2, M_3, M_4 \) - Gohberg-Semencul factorization

1: \( n \leftarrow \text{length}(c) \)
2: \( e_0 \leftarrow \text{zeros}(n, 1) \)
3: \( e_0(1) \leftarrow 1 \)
4: Solve \( T_n(c)x = e_0 \)
5: Compute \( M_1 \) using Eq. 6
6: Compute \( M_2 \) using Eq. 7
7: Compute \( M_3 \) using Eq. 8
8: Compute \( M_4 \) using Eq. 9

**Algorithm 7: ToeplitzDiag.** Diagonalize a Toeplitz matrix embedded in circulant matrix

**Input:** \( c, r \) - First column/row of the Toeplitz matrix

**Output:** \( D \) - Diagonal form of \( c \) embedded in circulant matrix

1: \( n \leftarrow \text{length}(c) \)
2: \( \text{CircMatrix} \leftarrow [c; 0; r(n : -1 : 2)] \)
3: \( D \leftarrow \text{FFT}(\text{CircMatrix}) \)

**Algorithm 8: ToeplitzInvMul.** Multiply the inverse toeplitz matrix, where the diagonal form of its Gohberg-Semencul factors are given by \( D_1, D_2, D_3, D_4 \), with the vector \( v \), in \( \mathcal{O}(n \log n) \) operations.

**Input:** \( D_1, D_2, D_3, D_4 \) - Gohberg-Semencul factors given in their diagonal form

**Output:** \( u \) - The product of the input matrix multiplied by \( v \).

1: \( u_1 \leftarrow \text{ToeplitzMul}(D_2, v) \)
2: \( u_1 \leftarrow \text{ToeplitzMul}(D_1, u_1) \)
3: \( u_2 \leftarrow \text{ToeplitzMul}(D_4, v) \)
4: \( u_2 \leftarrow \text{ToeplitzMul}(D_3, u_2) \)
5: \( u \leftarrow u_1 - u_2 \)

**Algorithm 9: ToeplitzMul.** Multiply a Toeplitz matrix, whose diagonal form is \( D \) with a vector \( v \) in \( \mathcal{O}(n \log n) \) operations.

**Input:** \( D \) - The matrix in its diagonal form, \( v \) - vector to multiply

**Output:** \( u \) - the product \( Dv \)

1: \( n \leftarrow \text{length}(v) \)
2: \( v \leftarrow [v; \text{zeros}(n, 1)] \)
3: \( fv \leftarrow \text{FFT}(v) \)
4: \( u \leftarrow \text{IFFT}(D \ast fv) \)
5: \( u \leftarrow u(1 : n) \)

References

[1] Amir Averbuch, Ronald R Coifman, David L Donoho, Michael Elad, and Moshe Israeli. Fast and accurate polar Fourier transform. *Applied and Computational Harmonic Analysis*, 21(2):145–167, 2006.
[2] Amir Averbuch, Ronald R Coifman, David L Donoho, Moshe Israeli, and Yoel Shkolnisky. A framework for discrete integral transformations I-The pseudopolar fourier transform. *SIAM Journal on Scientific Computing*, 30(2):764–784, 2008.

[3] Amir Averbuch and Yoel Shkolnisky. 3D Fourier based discrete Radon transform. *Applied and Computational Harmonic Analysis*, 15(1):33–69, 2003.

[4] Amit Bermanis, Amir Averbuch, and Yosi Keller. 3D symmetry detection and analysis using the pseudo-polar fourier transform. *International journal of computer vision*, 90(2):166–182, 2010.

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

[6] Barry A Cipra. The best of the 20th century: editors name top 10 algorithms. *SIAM news*, 33(4):1–2.

[7] Dutt A. and Rokhlin V. Fast fourier transforms for nonequispaced data. *SIAM J. Sci. Comput.*, 14(6):1368–1393, 1993.

[8] Natterer F. *The Mathematics of Computerized Tomography*. Classics in Applied Mathematics. SIAM, 2001.

[9] Ware A. F. Fast approximate Fourier transforms for irregularly spaced data. *SIAM Review*, 40(4):838–856, 1998.

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

[11] Hans G Feichtinger, Karlheinz Gr, Thomas Strohmer, et al. Efficient numerical methods in non-uniform sampling theory. *Numerische Mathematik*, 69(4):423–440, 1995.

[12] Markus Fenn, Stefan Kunis, and Daniel Potts. On the computation of the polar FFT. *Applied and Computational Harmonic Analysis*, 22(2):257–263, 2007.

[13] Fessler J. A. and Sutton B. P. Nonuniform fast Fourier transforms using min-max interpolation. *IEEE Transactions on Signal Processing*, 51(2):560 – 574, February 2003.

[14] Beylkin G. On the fast Fourier transform of functions with singularities. *Applied and Computational Harmonic Analysis*, 2:363–381, 1995.

[15] I Gohberg and A Semencul. On the inversion of finite Toeplitz matrices and their continuous analogs. *Mat. issled*, 2(1):201–233, 1972.

[16] Israel Gohberg and Vadim Olshevsky. Fast algorithms with preprocessing for matrix-vector multiplication problems. *Journal of Complexity*, 10(4):411–427, 1994.

[17] Gene H Golub and Charles F Van Loan. *Matrix computations*. John Hopkins University Press, 4 edition, 2012.

[18] Leslie Greengard and June-Yub Lee. Accelerating the nonuniform fast fourier transform. *SIAM Review*, 46(3):443–454, 2004.

[19] Greengard L. and Lee J.-Y. Accelerating the nonuniform fast fourier transform. *SIAM Review*, 46(3):443–454, 2004.

[20] Karlheinz Gröchenig. Reconstruction algorithms in irregular sampling. *Mathematics of Computation*, 59(199):181–194, 1992.
[21] Jiang H., Song C., Chen C. C., Xu R., Raines K. S., Fahimian B. P., Lu C. H., Lee T. K., Nakashima A.,
Urano J., Ishikawa T., Tamanoi F., and Miao J. Quantitative 3d imaging of whole, unstained cells by using
x-ray diffraction microscopy. Proceedings of the NAS, 107(25):11234–11239, 2010.

[22] Thomas Kailath and Joohwan Chun. Generalized Gohberg-Semencul formulas for matrix inversion. In The
Gohberg anniversary collection, pages 231–246. Springer, 1989.

[23] Thomas Kailath and Ali H Sayed. Fast reliable algorithms for matrices with structure. SIAM, 1999.

[24] Yosi Keller, Amir Averbuch, and Yoel Shkolnisky. Algebraically accurate volume registration using euler’s
theorem and the 3D pseudo-polar FFT. In Computer Vision and Pattern Recognition, 2005. CVPR 2005.
IEEE Computer Society Conference on, volume 2, pages 795–800. IEEE, 2005.

[25] Gitta Kutyniok, Wang-Q Lim, and Xiaosheng Zhuang. Digital shearlet transforms. In Shearlets, pages 239–282.
Springer, 2012.

[26] Gitta Kutyniok, Morteza Shahram, and Xiaosheng Zhuang. Shearlab: A rational design of a digital parabolic
scaling algorithm. SIAM Journal on Imaging Sciences, 5(4):1291–1332, 2012.

[27] Wayne Lawton. A new polar Fourier transform for computer-aided tomography and spotlight synthetic aperture
radar. IEEE Transactions on Acoustics Speech and Signal Processing, 36:931–933, 1988.

[28] Lee E., Fahimian B. P., Iancu C. V., Suloway C., Murphy G. E., Wright E. R., Castano Diez D., Jensen G. J.,
and Miao J. Radiation dose reduction and image enhancement in biological imaging through equally sloped
tomography. J. Struct. Biol., 164(2):221–227, 2008.

[29] Lee E., Fahimian B. P., Iancu C. V., Suloway C., Murphy G. E., Wright E. R., Jensen G. J., and Miao J.
Radiation dose reduction and image enhancement in biological imaging through equally sloped tomography.
J. Struct. Biol., 164:221–227, 2008.

[30] N Levinson. The Wiener RMS (root mean square) error criterion in filter design and prediction. 1947.

[31] Hanzhou Liu, Baolong Guo, and Zongzhe Feng. Pseudo-log-polar fourier transform for image registration.
Signal Processing Letters, IEEE, 13(1):17–20, 2006.

[32] Yu Mao, Benjamin P Fahimian, Stanley J Osher, and Jianwei Miao. Development and optimization of regularized
tomographic reconstruction algorithms utilizing equally-sloped tomography. Image Processing, IEEE
Transactions on, 19(5):1259–1268, 2010.

[33] Jianwei Miao, Friedrich Förster, and Ofer Levi. Equally sloped tomography with oversampling reconstruction.
Physical Review B, 72(5):052103, 2005.

[34] JE Pasciak. A note on the fourier algorithm for image reconstruction. Preprint AMD, 896, 1973.

[35] Potts D., Steidl G., and Tasche M. Fast Fourier transforms for nonequispaced data: A tutorial, chapter 12,
pages 249 – 274. Modern Sampling Theory: Mathematics and Applications. Springer, 2001.

[36] Michael Rauth and Thomas Strohmer. Smooth approximation of potential fields from noisy scattered data.
Geophysics, 63(1):85–94, 1998.

[37] Scott M. C., Chen C. C., Mecklenburg M., Zhu C., Xu R., Ercius P., Dahmen U., Regan B. C., and Miao J.
Electron tomography at 2.4-angstrom resolution. Nature, 483(7390):444–447, 2012.
[38] Yoel Shkolnisky and Shlomo Golubev. Fast convolution based inversion of the pseudo-polar Fourier transform. Submitted, 2014.

[39] J-L Starck, Emmanuel J Candès, and David L Donoho. The curvelet transform for image denoising. *Image Processing, IEEE Transactions on*, 11(6):670–684, 2002.

[40] Zhao Y., Brun E., Coan P., Huang Z., Sztroky A., Diemoz P. C., Liebhardt S., Mittone A., Gasilov S., Miao J., and Bravin A. High-resolution, low-dose phase contrast x-ray tomography for 3d diagnosis of human breast cancers. *Proceedings of the NAS*, 109(45):18290–18294, 2012.