

# A Framework for Discrete Integral Transformations I – the Pseudo-polar Fourier transform

A. Averbuch,<sup>\*</sup> R.R. Coifman,<sup>†</sup> D.L. Donoho,<sup>‡</sup>  
M. Israeli, and Y. Shkolnisky<sup>§</sup>

June 26, 2007

This paper is dedicated to the memory of Professor Moshe Israeli 1940-2007,  
who passed away on February 18.

## Abstract

Computing the Fourier transform of a function in polar coordinates is an important building block in many scientific disciplines and numerical schemes. In this paper we present the pseudo-polar Fourier transform that samples the Fourier transform on the pseudo-polar grid, also known as the concentric squares grid. The pseudo-polar grid consists of equally spaced samples along rays, where different rays are equally spaced and not equally angled. The pseudo-polar Fourier transform is shown to be fast (the same complexity as the FFT), stable, invertible, requires only

---

<sup>\*</sup>School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel

<sup>†</sup>Department of Mathematics, Yale University, New Haven, Connecticut 06520

<sup>‡</sup>Statistics Department, Stanford University, Stanford, CA 94305

<sup>§</sup>Department of Mathematics, Yale University, New Haven, Connecticut 06520

1D operations, and uses no interpolations. We prove that the pseudo-polar Fourier transform is invertible and develop two algorithms for its inversion: iterative and direct, both with complexity  $O(n^2 \log n)$ , where  $n \times n$  is the size of the reconstructed image. The iterative algorithm is based on the application of the conjugate-gradient method to the Gram operator of the pseudo-polar Fourier transform. Since the transform is ill-conditioned, we introduce a preconditioner, which significantly accelerates the convergence. The direct inversion algorithm utilizes the special frequency domain structure of the transform in two steps. First, it resamples the pseudo-polar grid to a Cartesian frequency grid, and then, recovers the image from the Cartesian frequency grid.

## 1 Introduction

Given a function  $f(x, y)$ , its 2D Fourier transform, denoted  $\hat{f}(\omega_x, \omega_y)$  is given by

$$\hat{f}(\omega_x, \omega_y) = \int_{\mathbb{R}^2} f(x, y) e^{-2\pi i(x\omega_x + y\omega_y)}, \quad \omega_x, \omega_y \in \mathbb{R}. \quad (1)$$

For discrete images  $I(u, v)$  Eq. 1 becomes

$$\hat{I}(\omega_x, \omega_y) = \sum_{u, v=-n/2}^{n/2-1} I(u, v) e^{-\frac{2\pi i}{m}(u\omega_x + v\omega_y)}, \quad \omega_x, \omega_y \in \mathbb{R}, \quad (2)$$

where  $m \geq n$  is an arbitrary integer. We assume for simplicity that the image  $I$  has equal dimensions in the  $x$  and  $y$  directions and that  $n$  is even. For practical applications, we need to evaluate  $\hat{I}$  for  $(\omega_x, \omega_y)$  in some discrete set.

The algorithm presented in this paper produces non-uniform polar samples of  $\hat{I}$ . It has the same complexity as the 2D FFT, uses no interpolations, and requires only 1D operations. Specifically, it uses only 1D FFTs. The algorithm does not require an accuracy parameter and the resulting samples have machine accuracy. This is in contrast to approaches like [8] and [12] that require an accuracy parameter, and whose relative accuracy, as well as their complexity,

depend on the accuracy parameter. In this sense, the proposed algorithm is the 2D counterpart of [3], [2], and [22] for the geometry of the pseudo-polar grid. These algorithms utilize the algebraic properties of the trigonometric polynomial

$$\hat{I}_1(\omega) = \sum_{u=-n/2}^{n/2-1} I_1(u)e^{-2\pi i u \omega/m}$$

to efficiently sample it on a given 1D grid: [3] samples  $\hat{I}_1$  on points that are equally spaced on the unit circle from 0 to  $2\pi$ ; [2] samples  $\hat{I}_1$  on points that are equally spaced on an arbitrary arc of the unit circle; and, [22] samples  $\hat{I}_1$  on spirals of the form  $AW^k$ , where  $A, W \in \mathbb{C}$ . The algorithm we propose in this paper uses the algebraic properties of Eq. 2 to efficiently compute its samples on the pseudo-polar grid. The suggested algorithm is different from other algorithms that compute the non-uniform FFT, as other algorithms utilize the analytic properties of Eq. 2 to efficiently approximate the values of  $\hat{I}$  within a prescribed accuracy. For a survey of non-uniform FFT algorithms that employ the latter approach see [23].

The paper is organized as follows. In Section 2 we present the 2D pseudo-polar grid and the 2D pseudo-polar Fourier transform. Relation to previous works and the contribution of the current paper are considered in Section 3. In Section 4 we present a fast  $O(n^2 \log n)$  algorithm for the computation of the pseudo-polar Fourier transform. In Section 5 we prove that it is invertible, and in Sections 6 and 7 we present two efficient algorithms that compute the inverse transform. Section 6 presents an iterative inversion algorithm that is based on the conjugate gradient method, while Section 7 presents a direct algorithm that is based on fast resampling of the frequency domain.

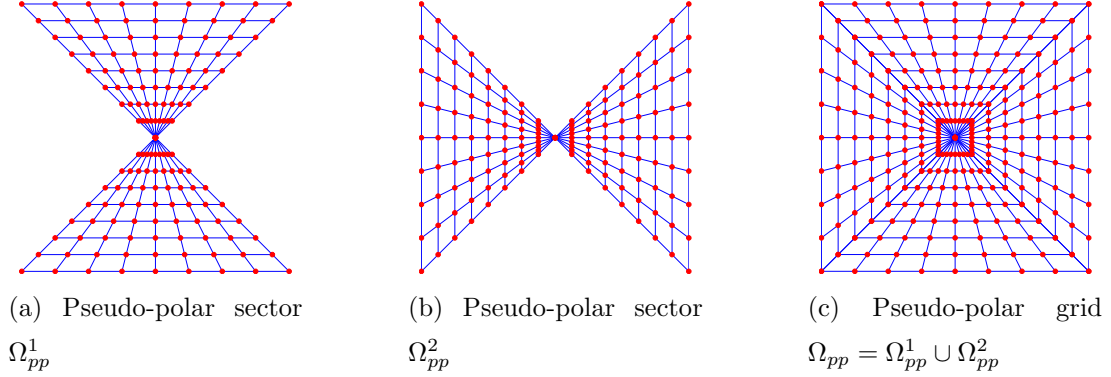


Figure 1: Pseudo-polar grid

## 2 Pseudo-polar grid

The 2D pseudo-polar grid, denoted  $\Omega_{pp}$ , is given by

$$\Omega_{pp} \triangleq \Omega_{pp}^1 \cup \Omega_{pp}^2, \quad (3)$$

where

$$\Omega_{pp}^1 \triangleq \left\{ \left( -\frac{2l}{n}k, k \right) \mid -n/2 \leq l \leq n/2, -n \leq k \leq n \right\} \quad (4)$$

$$\Omega_{pp}^2 \triangleq \left\{ \left( k, -\frac{2l}{n}k \right) \mid -n/2 \leq l \leq n/2, -n \leq k \leq n \right\}. \quad (5)$$

See Figs. 1a, 1b and 1c for an illustration of  $\Omega_{pp}^1$ ,  $\Omega_{pp}^2$ , and  $\Omega_{pp}$ , respectively. As can be seen from the figures,  $k$  serves as a “pseudo-radius” and  $l$  serves as a “pseudo-angle”. We denote a specific point in  $\Omega_{pp}^1$  and  $\Omega_{pp}^2$  by  $\Omega_{pp}^1(k, l)$  and  $\Omega_{pp}^2(k, l)$ , respectively.

The resolution of the pseudo-polar grid, given by Eqs. 3–5, is  $n + 1$  in the angular direction and  $m = 2n + 1$  in the radial direction. The presented construction uses these angular and radial resolutions to support the future derivation of the discrete Radon transform [1]. However, the construction can be repeated with arbitrary resolutions in the angular and radial directions.

In polar coordinates, the pseudo-polar grid is given by

$$\Omega_{pp}^1(k, l) = (r_k^1, \theta_l^1), \quad \Omega_{pp}^2(k, l) = (r_k^2, \theta_l^2), \quad (6)$$

where

$$r_k^1 = k\sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \quad r_k^2 = k\sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \quad (7)$$

$$\theta_l^1 = \pi/2 - \arctan\left(\frac{2l}{n}\right), \quad \theta_l^2 = \arctan\left(\frac{2l}{n}\right), \quad (8)$$

$k = -n, \dots, n$  and  $l = -n/2, \dots, n/2$ . As we can see in Fig. 1c, for each fixed angle  $l$ , the samples of the pseudo-polar grid are equally spaced in the radial direction. However, this spacing is different for different angles. Also, the grid is not equally spaced in the angular direction, but has equally spaced slopes. Formally,

$$\Delta r_k^1 \triangleq r_{k+1}^1 - r_k^1 = \sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \quad \Delta r_k^2 \triangleq r_{k+1}^2 - r_k^2 = \sqrt{4\left(\frac{l}{n}\right)^2 + 1} \quad (9)$$

and

$$\Delta\theta_{pp}^1(l) \triangleq \cot\theta_{l+1}^1 - \cot\theta_l^1 = \frac{2}{n}, \quad \Delta\theta_{pp}^2(l) \triangleq \tan\theta_{l+1}^2 - \tan\theta_l^2 = \frac{2}{n}, \quad (10)$$

where  $r_k^1, r_k^2, \theta_l^1$  and  $\theta_l^2$  are given by Eqs. 7 and 8.

The pseudo-polar Fourier transform is defined as the samples of  $\hat{I}$ , given by Eq. 2, on the pseudo-polar grid  $\Omega_{pp}$ , given by Eqs. 3–5. Formally, the pseudo-polar Fourier transform  $\hat{I}_{\Omega_{pp}^j}$  ( $j = 1, 2$ ) is a linear transformation that is defined for  $k = -n, \dots, n$  and  $l = -n/2, \dots, n/2$  as

$$\hat{I}_{\Omega_{pp}^1}(k, l) = \hat{I}\left(-\frac{2l}{n}k, k\right), \quad (11)$$

$$\hat{I}_{\Omega_{pp}^2}(k, l) = \hat{I}\left(k, -\frac{2l}{n}k\right), \quad (12)$$

where  $\hat{I}$  is given by Eq. 2. Using operator notation we denote the pseudo-polar Fourier transform by  $\mathcal{F}_{pp}I$ , where

$$(\mathcal{F}_{pp}I)(s, k, l) \triangleq \hat{I}_{\Omega_{pp}^s}(k, l), \quad (13)$$

$s = 0, 1, k = -n, \dots, n, l = -n/2, \dots, n/2$ .

### 3 Relation to previous work

The pseudo-polar grid was rediscovered several times in the literature under various names. It seems that this grid was first introduced by Mersereau and Oppenheim [18] under the name “concentric squares grid”. Mersereau and Oppenheim worked from the viewpoint of computerized tomography. They assumed that data on a continuum object were gathered in unequally spaced projections chosen so that the 1D Fourier transform corresponded to the concentric squares grid. They considered the problem of reconstructing a discrete array of  $n^2$  pixels from such Fourier domain data, and developed an algorithm based on interpolating from the data given in the concentric squares grid to the Cartesian grid. Mersereau and Oppenheim used simple 1D interpolation based on linear interpolation in rows/columns.

The difference between [18] and our work is three-fold: (1) Mersereau and Oppenheim’s definition samples half as frequently in the radial direction. This can be shown to be exactly the grid which would arise if we use  $m = n$  in our definition. Although this seems a minor difference, using  $m = 2n + 1$  is of crucial importance if one wishes to derive a discrete Radon transform that is based on the pseudo-polar grid. This is proved in [1]. As shown in [1], the original concentric squares grid does not preserve the straight-lines geometry of the continuous Radon transform, as it involves wrap-around of the underlying lines; (2) Mersereau and Oppenheim’s methodology is about reconstruction from data given about a continuum object; they do not attempt to define a forward transform, or establish the invertibility and fast inversion algorithms; and (3) their methodology is approximate - they do not obtain an exact conversion between concentric-squares and Cartesian grids.

We next consider an important set of papers in the literature of computerized tomography - both medical tomography [20, 5, 6] and synthetic aperture radar imaging [16]. Like Mersereau and Oppenheim, these authors are concerned with

image reconstruction; effectively they assume that one is given data in the Fourier domain on a concentric squares grid.

Pasciak's unpublished work [20], which is known among tomography experts through a citation in Natterer's book [19], showed in 1980 that, given data on a pseudo-polar grid in Fourier space, one could calculate a collection of  $n^2$  sums which, using the notation of this paper we can write as

$$\sum c_{k,l}^s e^{i\xi_{k,l}^s(u,v)}, \quad -n/2 \leq u, v < n/2, \quad (14)$$

where the  $\xi_{k,l}^s$  are points in the concentric squares grid. (Pasciak makes no reference to Mersereau and Oppenheim.) Pasciak studies this calculation, which is essentially what we would call the calculation of  $\text{adj } \mathcal{F}_{pp}$  for a variant of  $\mathcal{F}_{pp}$  (Eq. 13) that is based on  $m = n$  rather than  $m = 2n + 1$ , and shows it may be done in  $O(n^2 \log n)$  time. His key insight is to use the chirp-Z transform to calculate Fourier-like sums with exponents different from the usual  $2\pi kt/n$  by a factor  $\alpha$ .

Edholm and Herman [5, 6] develop the linogram, with a very similar point of view. They assume that data on a continuum object have been gathered at a set of projections which are equispaced in  $\tan \theta$  rather than  $\theta$ . By digitally sampling each constant  $\theta$  projection and taking a 1D discrete Fourier transform of the resulting samples, they argue that they are essentially given data on a concentric squares grid in Fourier space (making no reference to Mersereau and Oppenheim or Pasciak). They are concerned with reconstruction, consider the sums of Eq. 14, and derive a fast algorithm which is the same as Pasciak's, using again the chirp-Z transform.

Contemporaneously with Edholm and Herman, Lawton [16] develops a so called Polar Fourier transform for Synthetic Aperture Radar (SAR) imagery. He introduces a concentric squares grid, assumes that SAR data are essentially given on such a concentric squares grid in Fourier space, and considers the problem of rapidly reconstructing an image from such data. He considers the sums of Eq. 14 and derives a fast algorithm using again the chirp-Z transform. He refers to

Mersereau and Oppenheim.

In comparison to our work: (1) these works are about reconstruction only, assuming that data are gathered about a continuum object by a physical device, and (2) the algorithmic problem they consider is equivalent to rapidly computing Eq. 14. On the other hand, our paper defines discrete transforms that map digital images to their values on the concentric squares grid and vice versa. We explicitly consider reconstruction from the concentric squares grid, as opposed to previous works that compute the sums of Eq. 14, and actually only compute adj  $\mathcal{F}_{pp}$  and not the inverse. More differences, which are related to the discrete Radon transform, are discussed in [1].

## 4 Fast forward transform

In this section we present a fast algorithm that efficiently computes the pseudo-polar Fourier transform of an image  $I$ . The idea behind the algorithm is to evaluate  $\hat{I}$ , given by Eq. 2, on a Cartesian grid, by using the 2D FFT algorithm, and then, to resample the Cartesian frequency grid to the pseudo-polar grid. The operator behind this algorithm is the *fractional Fourier transform*.

The fractional Fourier transform of a vector  $c \in \mathbb{C}^{n+1}$ , denoted  $F_{n+1}^\alpha c$ , is given by

$$(F_{n+1}^\alpha c)(k) = \sum_{u=-n/2}^{n/2} c(u) e^{-2\pi i \alpha k u / (n+1)}, \quad k = -n/2, \dots, n/2, \quad \alpha \in \mathbb{R}. \quad (15)$$

An important property of the fractional Fourier transform is that given a vector  $c$  of length  $n + 1$ , the sequence  $(F_{n+1}^\alpha c)(k)$ ,  $k = -n/2, \dots, n/2$ , can be computed using  $O(n \log n)$  operations for any  $\alpha \in \mathbb{R}$  (see [2]). Equation 15 is usually referred to as the unaliased fractional Fourier transform and it differs from the usual definition of the fractional Fourier transform given in [2]. The algorithm that computes the unaliased fractional Fourier transform (Eq. 15) is very similar to the algorithm in [2], and is therefore omitted.

The algorithm that computes the pseudo-polar Fourier transform uses the following notation:

$E$  Padding operator.  $E(I, m, n)$  symmetrically zero-pads an image  $I$  of size  $n \times n$  to size  $m \times n$ .

$F_1^{-1}$  1D inverse DFT.

$\tilde{F}_m^\alpha$  Fractional Fourier transform with factor  $\alpha$ . The operator takes a sequence of length  $n$ , symmetrically pads it to length  $m = 2n + 1$ , applies to it the fractional Fourier transform with factor  $\alpha$ , and returns the  $n + 1$  central elements.

$F_2$  2D DFT.

$G_{k,n}$  Resampling operator given by

$$G_{k,n} = \tilde{F}_m^\alpha \circ F_1^{-1}, \quad \alpha = 2k/n. \quad (16)$$

Using this notation, the algorithm that computes the pseudo-polar Fourier transform  $\hat{I}_{\Omega_{pp}^1}(k, l)$  (Eq. 11) is given by Algorithm 1. The algorithm that computes  $\hat{I}_{\Omega_{pp}^2}(k, l)$  (Eq. 12) is implemented by swapping between the  $x$  and  $y$  axes in Algorithm 1.

The next theorem proves that Algorithm 1 indeed computes the pseudo-polar Fourier transform.

**Theorem 4.1** (Correctness of Algorithm 1). *Upon termination of Algorithm 1 we have*

$$Res_1(k, l) = \hat{I}_{\Omega_{pp}^1}(k, l), \quad (17)$$

where  $k = -n, \dots, n$ ,  $l = -n/2, \dots, n/2$ , and  $\hat{I}_{\Omega_{pp}^1}$  is given by Eq. 11.

---

**Algorithm 1** Computing the pseudo-polar Fourier transform  $\hat{I}_{\Omega_{pp}^1}$  (Eq. 11)

---

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

**Output:** Array  $Res_1$  with  $n+1$  rows and  $m = 2n+1$  columns that contains the samples of  $\hat{I}_{\Omega_{pp}^1}$

- 1:  $m \leftarrow 2n + 1$
  - 2:  $\hat{I}_d \leftarrow F_2(E(I, m, n))$
  - 3: **for**  $k = -n, \dots, n$  **do**
  - 4:    $q \leftarrow \hat{I}_d(\cdot, k)$
  - 5:    $w_k \leftarrow G_{k,n}(q), \quad w_k \in \mathbb{C}^{n+1}$
  - 6:    $Res_1(k, l) \leftarrow w_k(-l)$
  - 7: **end for**
- 

*Proof.* After completion of step 2 in Algorithm 1, the  $(l, k)$  element of  $\hat{I}_d$  is given by

$$\hat{I}_d(l, k) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i l u/n} e^{-2\pi i k v/m}, \quad (18)$$

where  $l = -n/2, \dots, n/2$ ,  $k = -n, \dots, n$ , and  $m = 2n + 1$ . Next, we calculate the value of  $Res_1(k_0, j)$  for some fixed  $k_0$ . Take row  $k_0$  from  $\hat{I}_d$  and denote the resulting vector of length  $n$  by  $q$

$$q(l) = \hat{I}_d(l, k_0). \quad (19)$$

According to step 5 in Algorithm 1  $w_k(j) = (G_{k_0,n}(q))_j$ , where, according to Eq. 16,  $G_{k_0,n}(q) = \tilde{F}_m^{2k_0/n}(F_1^{-1}(q))$ . We begin by evaluating  $F_1^{-1}(q)$ . By expanding Eq. 19 using Eq. 18 we get

$$q(l) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i l u/n} e^{-2\pi i k_0 v/m} = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i l u/n}, \quad (20)$$

where

$$c_{k_0}(u) = \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i k_0 v/m}, \quad u = -n/2, \dots, n/2 - 1. \quad (21)$$

Equation 20 states that the vector  $q$  is the DFT of  $\{c_{k_0}(u)\}$ . Therefore,  $F_1^{-1}(q) = \{c_{k_0}(u)\}$ . Finally, taking the fractional Fourier transform  $\tilde{F}_m^{2k_0/n}$  of the array  $\{c_{k_0}(u)\}$  using Eqs. 15 and 21 gives  $w_{k_0}(j) = (\tilde{F}_m^{2k_0/n}(\{c_{k_0}(u)\}))_j = \hat{I}(2jk_0/n, k_0)$ , from which we conclude that after the completion of step 6 in Algorithm 1

$$Res_1(k_0, j) = w_{k_0}(-j) = \hat{I}(-2jk_0/n, k_0) = \hat{I}_{\Omega_{pp}^1}(k_0, j), \quad j = -n/2, \dots, n/2.$$

□

Next, we analyze the complexity of Algorithm 1. Step 2 can be implemented in  $O(n^2 \log n)$  operations by using successive applications of 1D FFT. Each call to  $G_{k,n}$  in step 5 involves the application of a 1D inverse Fourier transform ( $O(n \log n)$  operations) followed by the computation of a fractional Fourier transform ( $O(n \log n)$  operations), and thus, requires  $O(n \log n)$  operations. Step 5 computes  $G_{k,n}$  for each row  $k$  ( $2n+1$  rows), which requires a total of  $O(n^2 \log n)$  operations. Steps 6 involves flipping  $2n+1$  vectors of length  $n+1$ , which requires a total of  $O(n^2)$  operations. Thus, the total complexity of Algorithm 1 is  $O(n^2 \log n)$  operations.

With an optimized implementation, the complexity of computing the pseudo-polar Fourier transform of a  $n \times n$  image is  $100n^2 \log_2 n$  operations. Note that the number of frequency samples in the output array is roughly  $4n^2$ . Computing  $4n^2$  Cartesian frequency samples using the 2D FFT requires  $20n^2 \log_2 n$  operations. That is, computing the pseudo-polar Fourier transform is only 5 times slower than the 2D FFT. This complexity analysis assumes that the 1D FFT of a vector of length  $n$  requires  $5n \log_2 n$  operations, and that the fractional Fourier transform of a vector of length  $n$  can be computed in  $20n \log_2 n$  operations (independent of  $\alpha$ ) [2].

Algorithm 1 can be modified to compute the adjoint pseudo-polar Fourier transform in  $O(n^2 \log n)$  operations, by reversing the order of execution in Algorithm 1 and replacing each line by its adjoint. The resulting algorithm, given in Algorithm 2, uses the following notation

$U$  Truncation operator. The operator  $U(I, n)$  takes an image  $I$  and returns its  $n \times n$  central elements.

$F_2^{-1}$  2D inverse DFT

adj  $G_{k,n}$  Adjoint of the operator  $G_{k,n}$  (Eq. 16). It is computed as

$$\text{adj } G_{k,n} = \frac{1}{n} F_1 \circ \text{adj } F_m^\alpha, \quad \alpha = 2k/n, \quad (22)$$

where  $F_1$  is the 1D DFT, and adj  $F_m^\alpha$  is an operator that takes a vector of length  $n + 1$ , symmetrically zero pads it to length  $m = 2n + 1$ , applies on it the fractional Fourier transform with factor  $-\alpha$ , and returns the  $n$  central elements.

---

**Algorithm 2** Computing the adjoint pseudo-polar Fourier transform

---

```

1: for  $k = -n, \dots, n$  do
2:    $q(l) \leftarrow \hat{I}_{\Omega_{pp}^1}(k, -l), \quad l = -n/2, \dots, n/2$ 
3:    $\tilde{I}_1(k, \cdot) \leftarrow \text{adj } G_{k,n}(q)$ 
4: end for
5:  $\tilde{I}_1 \leftarrow U(mnF_2^{-1}(\tilde{I}_1), n)$ 
6: for  $k = -n, \dots, n$  do
7:    $q(l) \leftarrow \hat{I}_{\Omega_{pp}^2}(k, -l), \quad l = -n/2, \dots, n/2$ 
8:    $\tilde{I}_2(\cdot, k) \leftarrow \text{adj } G_{k,n}(q)$ 
9: end for
10:  $\tilde{I}_2 \leftarrow U(mnF_2^{-1}(\tilde{I}_2), n)$ 
11:  $\tilde{I} \leftarrow \tilde{I}_1 + \tilde{I}_2$ 

```

---

## 5 Invertibility

Suppose we are given the values of the pseudo-polar Fourier transform  $\hat{I}_{\Omega_{pp}}$ . We want to show that it is possible to recover  $I$  from  $\hat{I}_{\Omega_{pp}}$ . Consider a vector of

samples from  $\hat{I}_{\Omega_{pp}^1}$  that corresponds to some  $k_0 \neq 0$ . From Eq. 11

$$\hat{I}_{\Omega_{pp}^1}(k_0, j) = \hat{I}(-2jk_0/n, k_0), \quad j = -n/2, \dots, n/2.$$

From Eq. 2

$$\hat{I}(-2jk_0/n, k_0) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i(-2k_0/n)ju/m} e^{-2\pi i k_0 v/m}, \quad (23)$$

which we write as

$$\hat{I}(-2jk_0/n, k_0) = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i(-2jk_0/n)u/m}, \quad (24)$$

where

$$c_{k_0}(u) = \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i k_0 v/m}, \quad u = -n/2, \dots, n/2 - 1.$$

Denote  $T_{k_0}(-2jk_0/n) \triangleq \hat{I}(-2jk_0/n, k_0)$ .  $T_{k_0}(-2jk_0/n)$  are the values of the trigonometric polynomial

$$T_{k_0}(x) = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i x u/m} \quad (25)$$

at the points  $\{-2jk_0/n\}$ ,  $j = -n/2, \dots, n/2$ . Since  $k_0 \neq 0$  we have the values of  $T_{k_0}(x)$  at  $n+1$  distinct points  $\{-2jk_0/n\}$ . Therefore, we can uniquely determine  $\{c_{k_0}(u)\}$  and  $T_{k_0}(x)$ . By evaluating  $T_{k_0}(x)$  at integer points using Eqs. 25, 24, and 2, we get  $T_{k_0}(j) = \hat{I}(j, k_0)$ , which means that we can recover the DFT of  $I$  for all  $k_0 \neq 0$ . Therefore, remains to recover  $\hat{I}(j, 0)$ ,  $j = -n/2, \dots, n/2$ .

By taking a sequence of samples from  $\hat{I}_{\Omega_{pp}^2}$  (Eq. 12), which corresponds to some  $k_0 \neq 0$ , we get  $\hat{I}_{\Omega_{pp}^2}(k_0, j) = \hat{I}(k_0, -2jk_0/n)$ , and using Eq. 2 we write

$$T'_{k_0}(-2jk_0/n) \triangleq \hat{I}(k_0, -2jk_0/n) = \sum_{v=-n/2}^{n/2-1} c'_{k_0}(v) e^{-2\pi i(-2jk_0/n)v/m},$$

where

$$c'_{k_0}(v) = \sum_{u=-n/2}^{n/2-1} I(u, v) e^{-2\pi i k_0 u/m}, \quad v = -n/2, \dots, n/2 - 1. \quad (26)$$

$\{T'_{k_0}(-2jk_0/n)\}$  are the values of the trigonometric polynomial

$$T'_{k_0}(x) = \sum_{v=-n/2}^{n/2-1} c'_{k_0}(v) e^{-2\pi i x v/m}$$

at  $n+1$  distinct points  $\{-2jk_0/n\}$ ,  $j = -n/2, \dots, n/2$ , and thus, uniquely determine  $\{c'_{k_0}(v)\}$  and  $T'_{k_0}(x)$ . By evaluating  $T'_{k_0}(x)$  at integer points and using Eqs. 26 and 2 we get  $T'_{k_0}(j) = \hat{I}(k_0, j)$ , for  $j = -n/2, \dots, n/2$ . Specifically, we can evaluate  $\hat{I}(k_0, 0)$  for  $k_0 \neq 0$ , which means that we can recover  $\hat{I}$  at all Cartesian grid points, except the origin. Since at the origin  $\hat{I}_{\Omega_{pp}^1}(0, 0) = \hat{I}(0, 0)$ , we have the values of  $\hat{I}(\xi_1, \xi_2)$  on the entire discrete Cartesian grid, that is, we can recover the DFT of  $I$  from  $\hat{I}_{\Omega_{pp}}$ . Finally, we can recover  $I$  by using the 2D inverse DFT. Hence, the 2D pseudo-polar Fourier transform is invertible.

## 6 Iterative inverse algorithm

We want to solve  $\mathcal{F}_{pp}x = y$ , where  $\mathcal{F}_{pp}$  is the 2D pseudo-polar Fourier transform, given by Eq. 13. Since  $y$  is not necessarily in the range of the pseudo-polar Fourier transform, due to, for example, noise or measurement errors, we would like to solve

$$\min_{x \in \mathcal{D}(\mathcal{F}_{pp})} \|\mathcal{F}_{pp}x - y\|_2 \quad (27)$$

instead, where  $\mathcal{D}(\mathcal{F}_{pp})$  is the domain of the pseudo-polar Fourier transform. Solving Eq. 27 is equivalent to solving the normal equations

$$\mathcal{F}_{pp}^* \mathcal{F}_{pp} x = \mathcal{F}_{pp}^* y, \quad (28)$$

where  $\mathcal{F}_{pp}^*$  is the adjoint pseudo-polar Fourier transform. Since  $\mathcal{F}_{pp}^* \mathcal{F}_{pp}$  is symmetric and positive definite, we can use the conjugate-gradient method [11] to

solve Eq. 28. When using the conjugate-gradient method, we never explicitly form the matrices that correspond to  $\mathcal{F}_{pp}$  and  $\mathcal{F}_{pp}^*$  (which are huge), since only their applications to a vector are required. As shown in Section 4, both the pseudo-polar Fourier transform and its adjoint can be applied in  $O(n^2 \log n)$  operations. Moreover, very little extra storage is required by the conjugate-gradient algorithm (as opposed to other iterative schemes like, for example, GMRES) so that the method can be used to recover very large images. The initial guess used for the conjugate-gradient method is zero. As we see below, we get excellent convergence even with this trivial initial guess.

The number of iterations required by the conjugate-gradient method depends on the condition number of the transform. To accelerate the convergence, we construct a preconditioner  $M$  and solve

$$\mathcal{F}_{pp}^* M \mathcal{F}_{pp} x = \mathcal{F}_{pp}^* M y. \quad (29)$$

$M$  is usually designed such that the condition number of the operator  $\mathcal{F}_{pp}^* M \mathcal{F}_{pp}$  is much smaller than the condition number of  $\mathcal{F}_{pp}^* \mathcal{F}_{pp}$ , or, such that the eigenvalues of  $\mathcal{F}_{pp}^* M \mathcal{F}_{pp}$  are well clustered.

We define the preconditioner  $M$  to be

$$M(k, l) = \begin{cases} \frac{1}{m^2} & k = 0 \\ \frac{2(n+1)|k|}{nm} & \text{otherwise,} \end{cases} \quad (30)$$

where  $k = -n, \dots, n$ ,  $l = -n/2, \dots, n/2$  and  $m = 2n + 1$ . The preconditioner is applied to each of the pseudo-polar sectors, where each pseudo-polar sector is of size  $m \times (n + 1)$ . We mention that the paper [7] proposes a method for designing effective preconditioners (weights) in the 1D case. The performance of these preconditioners can be guaranteed in terms of the properties of the sampling points. Unfortunately, the arguments in [7] apply only to the 1D case, and the construction therein cannot be applied to our case.

The efficiency of the preconditioner  $M$  (Eq. 30) is demonstrated in Fig. 2. Each graph presents the residual error of the conjugate-gradient as a function

of the iteration number. In Fig. 2a, the original image is a 2D Gaussian of size  $512 \times 512$  with  $\mu_x = \mu_y = 0$  and  $\sigma_x = \sigma_y = \frac{512}{6}$ . In Fig. 2b, the original image is a random image of size  $512 \times 512$ , whose entries are uniformly distributed between 0 and 1. In Fig. 2d, the original image is Barbara of size  $512 \times 512$ , shown in Fig. 2c. As we can see from Figs. 2a–2d, the suggested preconditioner significantly accelerates the convergence. With the preconditioner, only a few iterations are required, and the number of iterations is nearly independent of the content of the reconstructed image.

Tables 1 and 2 demonstrate the performance of the iterative inversion algorithm for images of various sizes. Table 1 presents the inversion of the pseudo-polar Fourier transform of a Gaussian. Table 2 presents the inversion of the pseudo-polar Fourier transform of a random image, whose entries are uniformly distributed between 0 and 1. In both tables, the error tolerance of the conjugate-gradient method is set to  $\varepsilon = 10^{-12}$ . The tables were generated as follows. Given an image  $I$  (Gaussian or random), its pseudo-polar Fourier transform is computed. Then, the iterative inversion algorithm is applied to recover the image. We denote by  $\tilde{I}$  the reconstructed image. To evaluate the quality of the reconstruction we use the following error measures

$$E_2 = \frac{\sqrt{\sum_{u,v} |\tilde{I}(u,v) - I(u,v)|^2}}{\sqrt{\sum_{u,v} |I(u,v)|^2}}, \quad E_\infty = \frac{\max_{u,v} |\tilde{I}(u,v) - I(u,v)|}{\max_{u,v} |I(u,v)|}, \quad (31)$$

where  $I$  is the original image.

As we can see from Tables 1 and 2, very few iterations are required to invert the pseudo-polar Fourier transform with high accuracy. The total complexity of the iterative inversion of the pseudo-polar Fourier transform is  $O(\rho(\varepsilon)n^2 \log n)$ , where  $\rho(\varepsilon)$  is the number of iterations required to achieve accuracy  $\varepsilon$ . As we can see from Table 2, the value of  $\rho(\varepsilon)$  depends very weakly on the size of the reconstructed image, and in any case  $\rho(10^{-12}) \leq 10$ .

$n$	$r$	$E_2$	$E_\infty$	iter	$t$ (sec)
8	2.80682e-014	2.47277e-007	1.60617e-007	9	0.359
16	1.14334e-013	4.92517e-007	3.86542e-007	8	0.563
32	5.99921e-014	3.44244e-007	2.92515e-007	8	1.202
64	1.05319e-013	4.67737e-007	5.92969e-007	7	2.625
128	6.05610e-013	1.16930e-006	2.56236e-006	6	7.295
256	1.09915e-013	4.94793e-007	1.60205e-006	6	26.789
512	4.30207e-013	9.87174e-007	5.05849e-006	5	83.525
1024	7.02642e-014	4.16717e-007	3.00086e-006	5	373.172

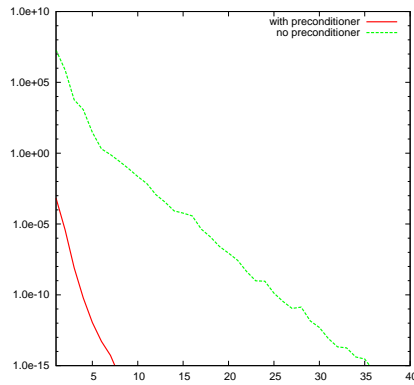
Table 1: Iterative inversion of the pseudo-polar Fourier transform of a Gaussian. The first column corresponds to the size of the original image ( $n \times n$ ). The second column corresponds to the residual error of the conjugate-gradient algorithm upon convergence. The third and fourth columns correspond to the  $E_2$  and  $E_\infty$  reconstruction errors, respectively. The fifth column corresponds to the iteration number at which the conjugate-gradient algorithm converged. The sixth column presents the running time in seconds.

$n$	$r$	$E_2$	$E_\infty$	iter	$t$ (sec)
8	4.56049e-014	3.33796e-007	5.21815e-007	9	0.391
16	2.21072e-013	7.13164e-007	1.06025e-006	9	0.672
32	6.30229e-013	1.27807e-006	3.81621e-006	9	1.359
64	3.72178e-013	9.30674e-007	4.31200e-006	9	3.250
128	1.40414e-013	5.43102e-007	2.27508e-006	10	11.562
256	1.69596e-013	5.82115e-007	1.95609e-006	10	43.484
512	1.25562e-013	5.05263e-007	2.47555e-006	10	158.641
1024	8.84166e-014	4.49097e-007	3.73745e-006	10	688.937

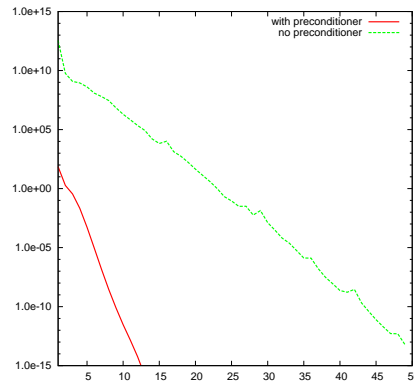
Table 2: Iterative inversion of the pseudo-polar Fourier transform of a random image. See Table 1 for details.

## 7 Direct inverse algorithm

The advantage of the iterative inversion algorithm is its simplicity. On the other hand, the iterative algorithm has several drawbacks. First, since the iterative inversion is based on a generic linear algebra approach, it does not utilize the special frequency domain structure of the transform. Second, while the iterative inversion algorithm is shown to have an acceptable empirical convergence rate, the exact number of iterations, and thus its running time, depends on the specific image to invert. See for example the different number of iterations required for a



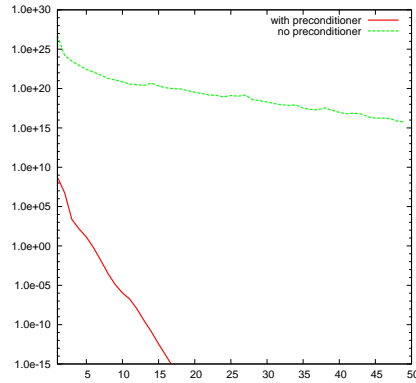
(a) Reconstruction of a Gaussian image  $512 \times 512$



(b) Reconstruction of a random image  $512 \times 512$



(c) Barbara  $512 \times 512$



(d) Reconstruction of Barbara  $512 \times 512$

Figure 2: The effect of using the preconditioner given in Eq. 30

Gaussian and a random image in Tables 1 and 2. Third, the conjugate-gradient method enables to estimate the reconstruction error in terms of the residual error. This error is related to the actual reconstruction error through the norm of the operator, which is difficult to estimate.

The direct inversion algorithm, presented in this section, overcomes all these limitations. It is tailored to the specific structure of the given transform. Its running time is independent of the specific image to invert. And, its accuracy can be estimated in terms of the actual reconstruction error.

The direct inversion algorithm consists of two phases. The first phase resamples the pseudo-polar Fourier transform into a Cartesian frequency grid; the second phase recovers the image from these Cartesian frequency samples. Resampling from the pseudo-polar to a Cartesian frequency grid is based on an “onion-peeling” procedure, which recovers a single row/column of the Cartesian grid in each iteration, from the outermost row/column to the origin. Recovering each row/column is based on a fast algorithm that resamples trigonometric polynomials from one set of frequencies to another set of frequencies.

For a different approach to the inversion of the pseudo-polar grid, which is based on 1D non-equally spaced FFTs, see [21].

This section is organized as follows. In Section 7.1 we present the mathematical tools used by the algorithm. Specifically, we present a fast algorithm that for a given Toeplitz matrix  $T_n$  of size  $n \times n$ , applies  $T_n^{-1}$  to an arbitrary vector in  $O(n \log n)$  operations. In Section 7.2 we present the outline of the algorithm that inverts the pseudo-polar Fourier transform. Section 7.3 describes the operators that resample from the pseudo-polar to a Cartesian frequency grid. Section 7.4 describes the procedure that recovers the image from this Cartesian frequency grid. And finally, Section 7.5 provides numerical examples that demonstrate the accuracy and efficiency of the proposed algorithm.

## 7.1 Solving Toeplitz systems

Let  $T_n$  be a Toeplitz matrix of size  $n \times n$  and let  $y$  be an arbitrary vector of length  $n$ . We are looking for a fast algorithm that computes  $T_n^{-1}y$ . We present an algorithm that consists of a fast factorization of the inverse Toeplitz matrix, followed by a fast algorithm that applies the inverse matrix on a vector. These algorithms are well-known, e.g. [9, 13], and are presented for the sake of completeness. We denote by  $T_n(c, r)$  a  $n \times n$  Toeplitz matrix whose first column and row are  $c$  and  $r$ , respectively.

Let  $C_n$  be a circulant matrix. As is well known, circulant matrices are diagonalized by the Fourier matrix  $C_n = F_n^* D_n F_n$ , where  $D_n$  is a diagonal matrix whose diagonal contains the eigenvalues  $\lambda_1, \dots, \lambda_n$  of  $C_n$ , and  $F_n$  is the Fourier matrix, given by  $F_n(j, k) = \frac{1}{\sqrt{n}} e^{2\pi i j k / n}$ . Moreover, if  $c = [c_0, c_{n-1}, \dots, c_1]^T$  is the first column of  $C_n$ , then,  $F_n c = [\lambda_1, \dots, \lambda_n]^T$ . Obviously, the matrices  $F_n$  and  $F_n^*$  can be applied in  $O(n \log n)$  operations by using the FFT. The multiplication of  $C_n$  with an arbitrary vector  $x$  of length  $n$  can be implemented in  $O(n \log n)$  operations by applying an FFT to  $x$ , multiplying the result by  $D_n$  element-by-element, and taking the inverse FFT.

To compute  $T_n x$  for an arbitrary Toeplitz matrix  $T_n$  and an arbitrary vector  $x$ , we first embed  $T_n$  in a circulant matrix  $C_{2n}$  of size  $2n \times 2n$

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

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

$$B_n = T_n([0, t_{-n+1}, \dots, t_{-2}, t_{-1}], [0, t_{n-1}, \dots, t_2, t_1]).$$

Then,  $T_n x$  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  $T_n$  is invertible. The Gohberg-Semencul formula [9, 10] provides a representation of  $T_n^{-1}$  as

$$T_n^{-1} = \frac{1}{x_0} (M_1 M_2 - M_3 M_4) \quad (32)$$

where

$$\begin{aligned} M_1 &= T_n([x_0, x_1, \dots, x_{n-1}], [x_0, 0, \dots, 0]), \\ M_2 &= T_n([y_{n-1}, 0, \dots, 0], [y_{n-1}, y_{n-2}, \dots, y_0]), \\ M_3 &= T_n([0, y_0, \dots, y_{n-2}], [0, \dots, 0]), \\ M_4 &= T_n([0, \dots, 0], [0, x_{n-1}, \dots, x_1]), \end{aligned}$$

$x = [x_0, \dots, x_{n-1}]$  is the solution of  $T_n x = e_0$ ,  $y = [y_0, \dots, y_{n-1}]$  is the solution of  $T_n y = e_{n-1}$ ,  $e_0 = [1, 0, \dots, 0]^T$  and  $e_{n-1} = [0, \dots, 0, 1]^T$ . 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  $2n$  elements. If the matrix  $T_n$  is fixed then the vectors  $x$  and  $y$  can be precomputed. Once the triangular Toeplitz matrices  $M_1$ ,  $M_2$ ,  $M_3$ ,  $M_4$  are computed, the application of  $T_n^{-1}$  is reduced to the application of four Toeplitz matrices, and thus, the application of  $T_n^{-1}$  to a vector requires  $O(n \log n)$  operations.

## 7.2 Outline of the direct inversion algorithm

For an image  $I$  of size  $n \times n$ , we define the array  $\hat{I}_D$  to be

$$\hat{I}_D(k, l) = \sum_{u, v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i(2ku+2lv)/m}, \quad k, l = -n/2, \dots, n/2, \quad (33)$$

where  $m = 2n + 1$ .  $\hat{I}_D(k, l)$  is obtained from the image  $I$  by zero padding it to size  $(2n + 1) \times (2n + 1)$ , applying the 2D FFT on the padded image, and discarding every other sample along each dimension.

The algorithm for inverting the pseudo-polar Fourier transform has two steps. The first computes the array  $\hat{I}_D$  from the samples of the pseudo-polar Fourier

transform. The second recovers the image  $I$  from the array  $\hat{I}_D$ . The first step processes each row/column of the pseudo-polar grid, from the outermost rows/columns to the origin, where at iteration  $k$  it recovers rows/columns  $k$  and  $n + k + 1$  of  $\hat{I}_D$  from rows/columns  $2k$  and  $2(n + k) + 1$  of the pseudo-polar grid ( $k = -n/2, \dots, 0$ ). This is depicted in Fig. 3. Gray circles represent samples of the pseudo-polar grid. Red circles represent samples of  $\hat{I}_D$ . The outermost rows and columns of  $\hat{I}_D$  are simply the outermost rows/columns of  $\hat{I}_{\Omega_{pp}^1}$  and  $\hat{I}_{\Omega_{pp}^2}$  (Eqs. 11 and 12), respectively (Figs. 3a and 3b). Rows  $n/2 - 1$  and  $-n/2 + 1$  of  $\hat{I}_D$  are recovered from rows  $n - 2$  and  $-n + 2$  of  $\hat{I}_{\Omega_{pp}^1}$  and from the columns of  $\hat{I}_D$  recovered in iteration 1 (Fig. 3c). Similarly, columns  $n/2 - 1$  and  $-n/2 + 1$  of  $\hat{I}_D$  are recovered from columns  $n - 2$  and  $-n + 2$  of  $\hat{I}_{\Omega_{pp}^2}$  and from the rows of  $\hat{I}_D$  recovered in iteration 1 (Fig. 3d). Rows  $n/2 - 2$  and  $-n/2 + 2$  of  $\hat{I}_D$  are recovered from rows  $n - 4$  and  $-n + 4$  of  $\hat{I}_{\Omega_{pp}^1}$  and from the columns of  $\hat{I}_D$  recovered in iterations 1 and 2. Columns  $n/2 - 4$  and  $-n/2 + 4$  of  $\hat{I}_D$  are recovered from columns  $n - 4$  and  $-n + 4$  of  $\hat{I}_{\Omega_{pp}^2}$  and from the rows of  $\hat{I}_D$  recovered in iterations 1 and 2. These iterations continue until all the samples of  $\hat{I}_D$  are recovered. As we see from Fig. 3, the Cartesian grid  $\hat{I}_D$  is recovered from the outside to the origin row by row and column by column. For this reason we refer to this procedure as “onion-peeling” inversion. At each step, we recover the next row/column of  $\hat{I}_D$  by using the samples of the corresponding row/column in the pseudo-polar grid and the columns/rows of  $\hat{I}_D$  recovered in previous iterations. Since the resolution of the pseudo-polar grid in the radial direction is  $2n + 1$ , only half of the rows/columns of the pseudo-polar grid are used to recover  $\hat{I}_D$ .

Let  $H_{n,k}^h$  be an operator that recovers row  $k$  of  $\hat{I}_D$  from row  $2k$  of  $\hat{I}_{\Omega_{pp}^1}$  and columns of  $\hat{I}_D$  with indices smaller than  $k$ , which were recovered in previous iterations. Similarly, let  $H_{n,k}^v$  be an operator that recovers column  $k$  of  $\hat{I}_D$  from column  $2k$  of  $\hat{I}_{\Omega_{pp}^2}$  and rows of  $\hat{I}_D$  with indices smaller than  $k$ , which were recovered in previous iterations. As depicted in Fig. 3,  $H_{n,k}^h$  and  $H_{n,k}^v$  operate on vectors of even length and return vectors of even length, as it is easier to implement them

for even lengths. Also,  $H_{n,k}^h$  and  $H_{n,k}^v$  always return vectors of length  $n$ , although some of the returned values were already computed in previous iterations. This simplifies the implementation while not affecting the complexity of the scheme. In Section 7.3 we give a formal description of  $H_{n,k}^h$  and  $H_{n,k}^v$  and describe a fast algorithm that implements them. Then, in Section 7.4 we present an algorithm that recovers  $I$  from  $\hat{I}_D$  (Eq. 33).

Algorithm 3 provides the pseudo-code of the direct inversion algorithm. Each application of the operators  $H_{n,k}^h$  and  $H_{n,k}^v$  requires  $O(n \log n + n \log(1/\varepsilon))$  operations, where  $\varepsilon$  is the prescribed accuracy of the reconstruction. Hence, the loop in lines 2–7 of Algorithm 3 recovers  $\hat{I}_D$  to accuracy  $\varepsilon$  using  $O(n^2 \log n + n^2 \log(1/\varepsilon))$

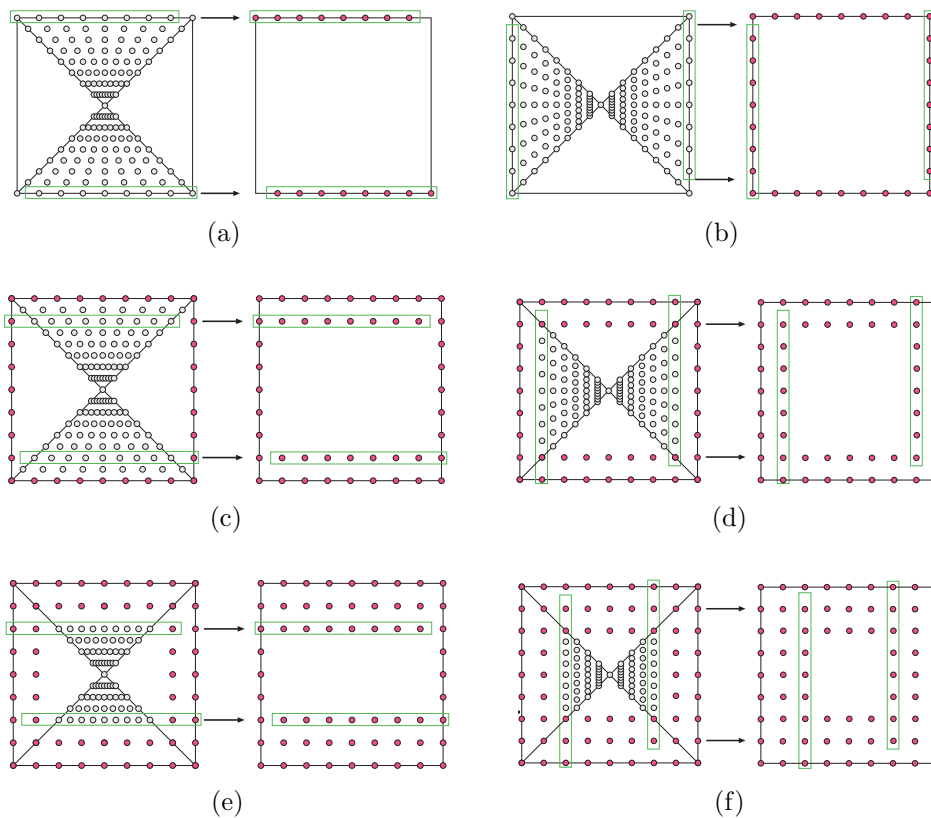


Figure 3: Outline of the “onion-peeling” algorithm for inverting the pseudo-polar Fourier transform

operations. Recovering  $I$  from  $\hat{I}_D$  requires  $O(n^2 \log n)$  operations. Hence, the total complexity of the direct inversion algorithm is  $O(n^2 \log n + n^2 \log(1/\varepsilon))$  operations, where  $\varepsilon$  is the required accuracy.

---

**Algorithm 3** Inversion of the pseudo-polar Fourier transform based on the “onion-peeling” approach

---

**Input:** Samples of the pseudo-polar Fourier transform  $\hat{I}_{\Omega_{pp}^1}$  and  $\hat{I}_{\Omega_{pp}^2}$  (Eqs. 11 and 12)

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

- 1:  $\hat{I}_D \leftarrow \text{zeros}(n+1, n+1)$
  - 2: **for**  $k = -n/2, \dots, 0$  **do**
  - 3:    $\hat{I}_D(k, :) \leftarrow H_{n,k}^h(\hat{I}_{\Omega_{pp}^1}(k, l), \hat{I}_D)$
  - 4:    $\hat{I}_D(-k, :) \leftarrow H_{n,-k}^h(\hat{I}_{\Omega_{pp}^1}(-k, l), \hat{I}_D)$
  - 5:    $\hat{I}_D(k, :) \leftarrow H_{n,k}^v(\hat{I}_{\Omega_{pp}^2}(k, l), \hat{I}_D)$
  - 6:    $\hat{I}_D(-k, :) \leftarrow H_{n,-k}^v(\hat{I}_{\Omega_{pp}^2}(-k, l), \hat{I}_D)$
  - 7: **end for**
  - 8:  $I \leftarrow \mathcal{F}_D^{-1} \hat{I}_D$  {recover  $I$  from  $\hat{I}_D$ }
- 

### 7.3 Operators $H_{n,k}^h$ and $H_{n,k}^v$

In this section we provide a detailed description of the operator  $H_{n,k}^h$ , as well as an efficient algorithm that computes it. The construction of  $H_{n,k}^v$  is similar.

Let  $\Omega_{n,k}$  denote the samples of the pseudo-polar grid  $\Omega_{pp}^1$  (Eq. 4) that correspond to row  $k$

$$\Omega_{n,k} \triangleq \left\{ \left( -\frac{2l}{n}k, k \right), l = -n/2, \dots, n/2 \right\}.$$

Let

$$\Lambda_{n,k} \triangleq \Omega_{n,2k} \cup C_{n,k}, \quad (34)$$

where

$$C_{n,k} \triangleq \begin{cases} C_{n,k}^1 & k = -n/2, \dots, 0, \\ C_{n,k}^2 & k = 1, \dots, n/2, \end{cases} \quad (35)$$

$$C_{n,k}^1 = \left\{ (2j, 2k) \mid j = -\frac{n}{2}, \dots, -|k| - 1, |k| + 1, \dots, \frac{n}{2} - 1 \right\}, \quad (36)$$

$$C_{n,k}^2 = \left\{ (2j, 2k) \mid j = -\frac{n}{2} + 1, \dots, -|k| - 1, |k| + 1, \dots, \frac{n}{2} \right\}. \quad (37)$$

For  $k = -n/2$  or  $k = n/2$  we have  $C_{n,k} = \emptyset$ . The set  $\Lambda_{n,k}$ , given by Eq. 34, contains samples of two densities:  $n - 2|k| - 1$  samples with spacing 2 and  $n + 1$  samples with spacing  $-2k/n$ .

Define the set  $\tilde{C}_{n,k}$  as

$$\tilde{C}_{n,k} \triangleq \begin{cases} \tilde{C}_{n,k}^1 & k = -n/2, \dots, 0, \\ \tilde{C}_{n,k}^2 & k = 1, \dots, n/2, \end{cases} \quad (38)$$

where

$$\tilde{C}_{n,k}^1 = \left\{ (2j, 2k) \mid j = -\frac{n}{2}, \dots, \frac{n}{2} - 1 \right\}, \quad (39)$$

$$\tilde{C}_{n,k}^2 = \left\{ (2j, 2k) \mid j = -\frac{n}{2} + 1, \dots, \frac{n}{2} \right\}. \quad (40)$$

The operator  $H_{n,k}^h$  takes the values  $\hat{I}_{\Lambda_{n,k}}$ , which are the values of  $\hat{I}$  (Eq. 2) on the set  $\Lambda_{n,k}$  (Eq. 34), and evaluates the values of  $\hat{I}$  on the set  $\tilde{C}_{n,k}$ . In other words, the operator  $H_{n,k}^h$  resamples the trigonometric polynomial  $\hat{I}$  from the set  $\Lambda_{n,k}$  to the set  $\tilde{C}_{n,k}$ . For a fixed  $k$ , the samples of  $\hat{I}$  on the set  $\Lambda_{n,k}$  can be written as the samples of some univariate trigonometric polynomial. Hence, for a fixed  $k$ , if we consider  $\Lambda_{n,k}$  and  $\tilde{C}_{n,k}$  as 1D sets, then, the operator  $H_{n,k}^h$  resamples a univariate trigonometric polynomial from the set  $\Lambda_{n,k}$  to the set  $\tilde{C}_{n,k}$ . Thus, we implement the operator  $H_{n,k}^h$  as follows. Let  $k$  be a fixed integer in the range  $-n/2, \dots, 0$  (the construction for  $k = 1, \dots, n/2$  is similar). We choose for each point  $p_j \in \tilde{C}_{n,k}$ ,  $j = -n/2, \dots, n/2 - 1$ , its closest point in the set  $\Lambda_{n,k}$ . Denote this subset of points from  $\Lambda_{n,k}$  by  $\tilde{\Lambda}_{n,k}$ . Then, we use the algorithm

presented in [4] to resample a trigonometric polynomial from the set  $\tilde{\Lambda}_{n,k}$  to the set  $\tilde{C}_{n,k}$  with an arbitrary prescribed accuracy  $\varepsilon$ . An important property of the set  $\tilde{\Lambda}_{n,k}$  is that its points are “not too far” from the points of  $\tilde{C}_{n,k}$ . Specifically, if  $j = -\frac{n}{2}, \dots, -|k| - 1, |k| + 1, \dots, \frac{n}{2} - 1$  then  $p_j \in C_{n,k}$  and hence, the distance between  $p_j$  and its closest point in  $\tilde{\Lambda}_{n,k}$  is zero. If  $p_j \notin C_{n,k}$ , then, we can find a point in  $\Lambda_{n,k}$  whose distance to  $p_j$  is less than  $-2k/n$ , which goes to zero as  $k$  goes to zero (approaches the origin).

A simple induction shows that at the beginning of the  $k^{\text{th}}$  iteration of Algorithm 3, we have already recovered the values of  $\hat{I}$  on  $C_{n,k}$  by using the operators  $H_{n,q}^v$  with  $|q| < k$ . By combining the values of  $\hat{I}$  on  $C_{n,k}$  with the values of  $\hat{I}$  on  $\Omega_{n,k}$ , which are the values of the pseudo-polar Fourier transform that correspond to row  $k$ , we obtain the values of  $\hat{I}$  on the set  $\Lambda_{n,k}$ . Hence, at iteration  $k$  we apply  $H_{n,k}^h$  on the set  $\Lambda_{n,k}$  and recover the values of  $\hat{I}$  on the set  $\tilde{C}_{n,k}$ . In other words, we recover row  $k$  of  $\hat{I}_D$  (Eq. 33). The definition of the operator  $H_{n,k}^v$  is similar, and a similar argument shows that iteration  $k$  recovers the  $k^{\text{th}}$  column of  $\hat{I}_D$ . Thus, by using the operators  $H_{n,k}^h$  and  $H_{n,k}^v$  we recover  $\hat{I}_D$  to accuracy  $\varepsilon$  in  $O(n^2 \log n + n^2 \log(1/\varepsilon))$  operations.

## 7.4 Recovering $I$ from $\hat{I}_D$

Next, we present a fast algorithm that recovers the image  $I$  from the values of  $\hat{I}_D(k, l)$ ,  $k, l = -n/2, \dots, n/2$  (Eq. 33). We define  $\mathcal{F}_D : \mathbb{C}^n \rightarrow \mathbb{C}^{n+1}$  as

$$(\mathcal{F}_D x)(k) = \sum_{u=-n/2}^{n/2-1} x(u) e^{-2\pi i u(2k)/m}, \quad k = -n/2, \dots, n/2, \quad m = 2n + 1. \quad (41)$$

Given a vector  $x$  of length  $n$ , the operator  $\mathcal{F}_D$  is implemented by symmetrically zero padding  $x$  to length  $2n + 1$ , applying the FFT on the padded vector, and discarding every other sample. The complexity of applying  $\mathcal{F}_D$  on a vector of length  $n$  is  $O(n \log n)$  operations.

From Eq. 33 we see that  $\hat{I}_D$  can be computed by a separable application of the 1D operator  $\mathcal{F}_D$  along the rows and columns of  $I$ . Since each application of  $\mathcal{F}_D$  requires  $O(n \log n)$  operations, the total complexity of computing  $\hat{I}_D$  is  $O(n^2 \log n)$  operations. To recover  $I$  from  $\hat{I}_D$  we need to apply  $\mathcal{F}_D^{-1}$  along the rows and columns of  $\hat{I}_D$ . In the remaining of the section we show that each application of  $\mathcal{F}_D^{-1}$  to a row/column of  $\hat{I}_D$  requires  $O(n \log n)$  operations. Hence, recovering  $I$  from  $\hat{I}_D$ , that is, applying  $\mathcal{F}_D^{-1}$  to all rows and columns of  $\hat{I}_D$ , requires  $O(n^2 \log n)$  operations.

We start with the adjoint operator  $\mathcal{F}_D^*$ . Let  $y$  be a vector of length  $n + 1$ . The operator  $\mathcal{F}_D^*$  is defined by

$$(\mathcal{F}_D^* y)(u) = \sum_{k=-n/2}^{n/2} y(k) e^{2\pi i u(2k)/m}, \quad u = -n/2, \dots, n/2 - 1, \quad m = 2n + 1. \quad (42)$$

It is easy to verify that  $\mathcal{F}_D^*$  is indeed the adjoint of  $\mathcal{F}_D$ .

The application of  $\mathcal{F}_D^*$  on  $y$  is computed by inserting a zero between every two elements of  $y$ , which results in a vector of length  $2n + 1$ , applying the adjoint Fourier transform, which is the inverse FFT multiplied by  $2n + 1$ , and retaining the  $n$  central elements. Clearly, the application of  $\mathcal{F}_D^*$  to  $y$  requires  $O(n \log n)$  operations. Since the matrix of the operator  $\mathcal{F}_D$  is not unitary, the operator  $\mathcal{F}_D^*$  is not the inverse of  $\mathcal{F}_D$ .

Applying the operator  $\mathcal{F}_D^{-1}$  on a vector  $y$  is equivalent to solving the linear system  $\mathcal{F}_D x = y$ . We apply  $\mathcal{F}_D^*$  on both sides and obtain the normal equations  $\mathcal{F}_D^* \mathcal{F}_D x = \mathcal{F}_D^* y$ , or equivalently,  $x = (\mathcal{F}_D^* \mathcal{F}_D)^{-1} \mathcal{F}_D^* y$ . Solving the normal equations gives the solution to  $\min_x \|\mathcal{F}_D x - y\|_2$ . Hence, if  $y$  is not in the range of  $\mathcal{F}_D$ , the inversion algorithm finds the vector  $x$  such that  $\mathcal{F}_D x$  is closest to  $y$ .

Note that  $\mathcal{F}_D^* \mathcal{F}_D$  is invertible. To see this, first note that  $x(u)$ ,  $u = -n/2, \dots, n/2 - 1$ , in Eq. 41 is uniquely determined by the samples  $(\mathcal{F}_D x)(k)$ ,  $k = -n/2, \dots, n/2$ . Therefore,  $\text{Ker } \mathcal{F}_D = \{0\}$ . Next,  $\mathcal{F}_D^* \mathcal{F}_D$  is positive definite. Indeed, for an arbitrary

bitrary vector  $x$

$$\langle F_D^* F_D x, x \rangle = \langle F_D x, F_D x \rangle = \|F_D x\|^2 \geq 0,$$

but since  $\text{Ker } F_D = \{0\}$ , the last equation is strictly positive and  $\mathcal{F}_D^* \mathcal{F}_D$  is positive definite and invertible.

The matrix  $\mathcal{F}_D^* \mathcal{F}_D$  is a Toeplitz matrix, whose entries are given by

$$(\mathcal{F}_D^* \mathcal{F}_D)_{k,l} = \sum_{u=-n/2}^{n/2} e^{\frac{4\pi i u}{2n+1}(k-l)}, \quad k, l = -n/2, \dots, n/2 - 1.$$

Moreover, since  $\mathcal{F}_D^* \mathcal{F}_D$  is symmetric and positive-definite,  $x_0$  in the Gohberg-Semencul decomposition (Eq. 32) is positive [17]. Therefore, as shown in Section 7.1, applying  $(\mathcal{F}_D^* \mathcal{F}_D)^{-1}$  on an arbitrary vector requires  $O(n \log n)$  operations. Since the application of  $\mathcal{F}_D^*$  requires also  $O(n \log n)$  operations, we conclude that computing  $x = (\mathcal{F}_D^* \mathcal{F}_D)^{-1} \mathcal{F}_D^* y$  for an arbitrary  $y$  requires  $O(n \log n)$  operations.

We recover the image  $I$  by applying  $(\mathcal{F}_D^* \mathcal{F}_D)^{-1} \mathcal{F}_D^*$  on all rows and columns of  $\hat{I}_D$ . Since each application requires  $O(n \log n)$  operations, the total complexity of recovering  $I$  from  $\hat{I}_D$  is  $O(n^2 \log n)$  operations.

## 7.5 Numerical results

Algorithm 3 was implemented in Matlab and was applied to two types of test images of various sizes. The first image is a Gaussian image of size  $n \times n$  with mean  $\mu_x = \mu_y = 0$  and standard deviation  $\sigma_x = \sigma_y = \frac{n}{6}$ . The second is a random image whose entries are uniformly distributed in  $[0, 1]$ . For each test image we compute its pseudo-polar Fourier transform followed by the application of the inverse pseudo-polar Fourier transform (Algorithm 3). The reconstructed image is then compared to the original image. We use the error measures given by Eq. 31.

The results are summarized in Tables 3–6. Table 3 presents the results of inverting the pseudo-polar Fourier transform of a Gaussian. Tables 4–6 present the results of inverting the pseudo-polar Fourier transform of a random image,

whose entries are uniformly distributed in  $[0, 1]$ , for various values of  $\varepsilon$ . All tests were implemented in Matlab on a Pentium 2.8GHz running Linux. As we see from Tables 3–6, the actual accuracy is higher than the prescribed one. The reason for this is that the error bounds in [4] hold for any sampling geometry. In the special sampling geometry involved in the inversion of the pseudo-polar Fourier transform, these estimates turn out to be too pessimistic. Nevertheless, note that for the random matrix the actual accuracy is consistently three digits more accurate than the prescribed accuracy.

$n$	$E_2$	$E_\infty$	$t_F$	$t_I$
8	1.54826e-013	1.42780e-013	0.062	0.281
16	8.46571e-013	5.40734e-013	0.031	0.047
32	2.30805e-012	2.17171e-012	0.062	0.203
64	1.25906e-012	1.49238e-012	0.156	0.703
128	7.24066e-013	7.32485e-013	0.484	3.702
256	4.32719e-013	4.99887e-013	1.906	18.462
512	2.49692e-013	2.92489e-013	7.435	89.174

Table 3: Inverting the pseudo-polar Fourier transform of a Gaussian with  $\varepsilon = 10^{-7}$ . The first column corresponds to  $n$ , where  $n \times n$  is the size of the original image. The second and third columns correspond to the  $E_2$  and  $E_\infty$  reconstruction errors, respectively. The fourth column, denoted  $t_F$ , corresponds to the time (in seconds) required to compute the forward pseudo-polar Fourier transform. The fifth column, denoted  $t_I$ , corresponds to the time (in seconds) required to invert the pseudo-polar Fourier transform using Algorithm 3.

$n$	$E_2$	$E_\infty$	$t_F$	$t_I$
8	2.94094e-009	4.31691e-009	0.016	0.016
16	7.40180e-009	1.11551e-008	0.016	0.031
32	3.00908e-008	5.76409e-008	0.062	0.110
64	2.28288e-008	3.79261e-008	0.141	0.578
128	1.47706e-008	3.01046e-008	0.515	3.000
256	1.06168e-008	2.58128e-008	1.860	15.390
512	8.40374e-009	1.98289e-008	7.328	73.938

Table 4: Inverting the pseudo-polar Fourier transform of a random image with  $\varepsilon = 10^{-5}$ . See Table 3 for details.

## 8 Conclusions

We presented the pseudo-polar grid together with the pseudo-polar Fourier transform, which is a fast algorithm that samples the Fourier transform on the pseudo-polar grid. We proved the correctness of the algorithm, showed that the transform is invertible, and presented two inversion algorithms.

$n$	$E_2$	$E_\infty$	$t_F$	$t_I$
8	1.52637e-011	2.30025e-011	0.015	0.016
16	5.16820e-011	6.90594e-011	0.031	0.031
32	1.85304e-010	2.67004e-010	0.047	0.125
64	1.16524e-010	1.66030e-010	0.141	0.656
128	6.71729e-011	1.25607e-010	0.500	3.297
256	5.53006e-011	1.16686e-010	1.844	16.875
512	3.94900e-011	9.00832e-011	7.313	81.468

Table 5: Inverting the pseudo-polar Fourier transform of a random image with  $\varepsilon = 10^{-7}$ . See Table 3 for details.

$n$	$E_2$	$E_\infty$	$t_F$	$t_I$
8	1.30958e-015	1.34194e-015	0.000	0.016
16	1.67241e-015	2.24504e-015	0.032	0.031
32	6.50428e-015	1.10842e-014	0.047	0.125
64	1.59849e-014	2.29404e-014	0.156	0.735
128	3.70890e-014	6.79917e-014	0.500	3.875
256	7.27812e-014	1.77150e-013	1.860	19.781
512	3.41732e-013	6.84542e-013	7.250	95.671

Table 6: Inverting the pseudo-polar Fourier transform of a random image with  $\varepsilon = 10^{-11}$ . See Table 3 for details.

Both the forward and inverse transforms can be generalized to higher dimensions. In particular, the direct inversion algorithm is based only on 1D operations, and so its generalization to higher dimensions is straightforward.

The pseudo-polar Fourier transform is applicable to problems that require polar Fourier representations, but whose discretizations need not be uniform. Examples of such applications are image registration [15] and symmetry detection [14]. The pseudo-polar Fourier transform is also closely related to the discrete Radon transform [1]. Like the continuous Radon transform, the discrete Radon transform is related to the Fourier transform of the underlying object through the Fourier slice theorem. Thus, the pseudo-polar Fourier transform provides an efficient algorithm and an infrastructure for the computation and inversion of the discrete Radon transform. This will be reported in a different publication [1].

## References

- [1] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli, Y. Shkolnisky, and I. Sedelnikov. A framework for discrete integral transformations II – the 2D

discrete Radon transform.

- [2] D. H. Bailey and P. N. Swarztrauber. The fractional Fourier transform and applications. *SIAM Review*, 33(3):389–404, September 1991.
- [3] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computations*, 19(60):297–301, April 1965.
- [4] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data, II. *Applied and Computational Harmonic Analysis*, 2(1):85–100, 1995.
- [5] P. Edholm and G. T. Herman. Linograms in image reconstruction from projections. *IEEE Transactions on Medical Imaging*, 6:301–307, 1987.
- [6] P. Edholm, G. T. Herman, and D. A. Roberts. Image reconstruction from linograms: Implementation and evaluation. *IEEE Transactions on Medical Imaging*, 7(3):239–246, 1988.
- [7] H. G. Feichtinger, K. Gröchenig, and T. Strohmer. Efficient numerical methods in non-uniform sampling theory. *Numerische Mathematik*, 69(4):423–440, 1995.
- [8] K. Fourmont. Non-equispaced fast Fourier transforms with applications to tomography. *Journal of Fourier Analysis and Applications*, 9(5):431–450, September 2003.
- [9] I. Gohberg and V. Olshevsky. Fast algorithms with preprocessing for matrix-vector multiplication problems. *Journal of Complexity*, 10:411–427, 1994.
- [10] I. Gohberg and A. Semencul. The inversion of finite Toeplitz matrices and their continual analogues. (Russian). *Mat. Issled.*, 7(2):201–233, 1972.
- [11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1984.

- [12] L. Greengard and J. Y. Lee. Accelerating the nonuniform fast Fourier transform. *SIAM Review*, 46(3):443–454, 2004.
- [13] T. Kailath and A. H. Sayed, editors. *Fast Reliable Algorithms for Matrices with Structure*. SIAM, 1999.
- [14] Y. Keller and Y. Shkolnisky. A signal processing approach to symmetry detection. *IEEE Transactions on Image Processing*, 15(8):2198–2207, 2006.
- [15] Y. Keller, Y. Shkolnisky, and A. Averbuch. The angular difference function and its application to image registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):969–976, June 2005.
- [16] W. Lawton. A new polar Fourier transform for computer-aided tomography and spotlight synthetic aperture radar. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(6):931–933, 1988.
- [17] E. Linzer and M. Vetterli. Iterative Toeplitz solvers with local quadratic convergence. *Computing*, 49(4):339–347, 1993.
- [18] R. M. Mersereau and A. V. Oppenheim. Digital reconstruction of multidimensional signals from their projections. *Proceedings of the IEEE*, 62(10):1319–1338, 1974.
- [19] F. Natterer. *The Mathematics of Computerized Tomography*. Classics in Applied Mathematics. SIAM, 2001.
- [20] J. E. Pasciak. A note on the Fourier algorithm for image reconstruction. Preprint, Applied Mathematics Department, Brookhaven National Laboratory, Upton, New York, 1973.
- [21] D. Potts and G. Steidl. A new linogram algorithm for computerized tomography. *IMA Journal of Numerical Analysis*, 21(3):769–782, 2001.

- [22] L. R. Rabiner, R. W. Schafer, and C. M. Rader. The chirp z-transform algorithm. *IEEE Transactions on Audio Electroacoustics*, 17(2):86–92, June 1969.
- [23] A. F. Ware. Fast approximate Fourier transforms for irregularly spaced data. *SIAM Review*, 40(4):838–856, 1998.