Pseudo-Polar and Discrete Radon Transforms Software Package Documentation

Yoel Shkolnisky

February 19, 2003

Contents

Chapter 1

Introduction

The Radon software package computes the the 2-D pseudo-polar Fourier transform and the discrete 2-D Radon transform.

The functions are written in Matlab and do not use any external packages.

All functions are documented using Matlab's comments. For help on any function in the package type `help` ***`function name`***.

Chapter 2

Computing the pseudo-polar Fourier transform

2.1 Definition

The 2-D pseudo-polar Fourier transform is defined as the samples of the trigonometric polynomial

$$\widehat{I}(x_1, x_2) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} I(u, v) e^{-2\pi\imath(x_1 u + x_2 v)/m} \tag{2.1}$$

on the pseudo-polar grid P given by

$$P_1 = \widehat{I}(-2lk/n, k) \tag{2.2}$$

$$P_2 = \widehat{I}(k, -2lk/n) \tag{2.3}$$

$$P = P_1 \cup P_2 \tag{2.4}$$

where $k = -n \dots n$ and $l = -n/2 \dots n/2$. See Figs. 2.1 and 2.2 for illustration of the scattering of the pointsets P_1 and P_2 in the frequency domain.

Using the pointsets P_1 and P_2, defined in Eqs. 2.2 and 2.3, respectively, we define the arrays PP_1 and PP_2 as samples of the trigonometric polynomial, given in Eq. 2.1, on the sets P_1 and P_2. Formally,

$$PP_1(k, l) = \widehat{I}(-2lk/n, k) \tag{2.5}$$

$$PP_2(k, l) = \widehat{I}(k, -2lk/n) \tag{2.6}$$

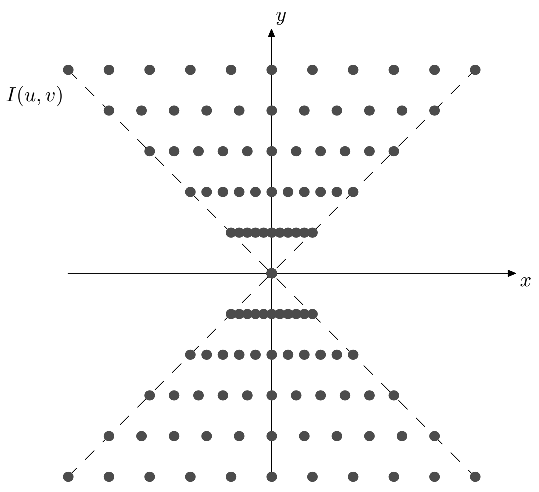

Figure 2.1: Sampling points used for the array res_1

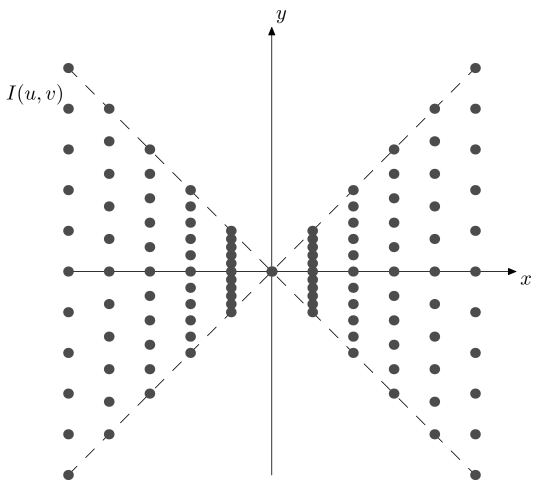

Figure 2.2: Sampling points used for the array res_2

where $k = -n \ldots n$ and $l = -n/2 \ldots n/2$. The values of PP_1 and PP_2 represent the exact frequency values of the image I at the points P_1 and P_2, respectively. As we can see from Eqs. 2.5 and 2.6, the pseudo-polar Fourier transform maps an image I of size $n \times n$ into two arrays PP_1 and PP_2, each of size $(2n+1) \times (n+1)$.

2.2 The forward transform

The function `PPFT.m` computes the pseudo-polar Fourier transform for an $n \times n$ image using $O(n^2 \log n)$ operations. The function returns two arrays `res1` and `res2` of size $(2n+1) \times (n+1)$ ($2n+1$ rows and $n+1$ columns) such that

$$res_1(k,l) = PP_1(k,l) \tag{2.7}$$
$$res_2(k,l) = PP_2(k,l) \tag{2.8}$$

where PP_1 and PP_2 are defined in Eqs. 2.5 and 2.6, respectively.

The signature of the function `PPFT.m` is

```
function [re1,res2] = PPFT(im);
```

The input image `im` must be a $n \times n$ image (either real or complex) where n must be even. Otherwise, the function terminates with an error message. For example:

```
im = rand(13,17);
[res1,res2] = PPFT(im);
??? Error using ==> ppft
Input image must be square
```

The following example demonstrates how to use the function `PPFT.m`:

```
im = magic(32);
[res1,res2] = PPFT(im);
```

The first line creates a magic square `im` of size 32×32. The second line computes the pseudo-polar Fourier transform of `im` and stores the results in the arrays `res1` and `res2`. See Eqs. 2.7 and 2.8 for description of the contents of `res1` and `res2`.

The function `PPFT.m` follows the notation and pseudo-code detailed in [1]. The package also provides the function `OptimizedPPFT.m`, which is an optimized version of `PPFT.m`. The function `OptimizedPPFT.m` uses the same input and output parameters as `PPFT.m` and uses further optimizations to reduce operations count.

2.3 The inverse transform

The inverse pseudo-polar Fourier transform is computed using the function `ippft.m`. At its basic mode, the function accepts two pseudo-polar sectors `pp1` and `pp2` and recovers the image `im`:

```
    im = ippft(pp1,pp2);
```

The recovered image `im` satisfies

```
    [pp1,pp2] = PPFT(im)
```

The mandatory arguments `pp1` and `pp2` must be 2-D arrays of size $(2n+1)\times(n+1)$ where $n\times n$ is the size of the reconstructed image. If the sizes of the input arrays do not conform these requirements, the function will abort and display an error message.

2.3.1 The inversion algorithm: Theory

The inversion algorithm is based on an iterative conjugate gradient algorithm. Since the pseudo-polar Fourier transform is ill-conditioned, we use a preconditioner to achieve and accelerate the convergence of the inversion.

Mathematically, the conjugate gardiet solver solves the equation

$$[pp1, pp2] = P(im) \tag{2.9}$$

where im is the unknown image that the pseudo-polar Fourier transform P maps into $pp1$ and $pp2$.

The inversion algorithm uses the generic conjugate-gradient solver `CG.m`. The function `CG.m` inverts the operator defined in `PtP.m`. The operator `PtP.m` must be symmetric and positive definite. Since the pseudo-polar Fourier transform is not positive definite, its inversion algorithm works as follows:

1. Given the pseudo-polar Fourier transform sectors Y, we want to find an image im that satisfies

   $$Y = P(im) \tag{2.10}$$

   where P is the pseudo-polar Fourier transform.

2. Compute

   $$\tilde{Y} = D(Y) \tag{2.11}$$

   where D is the preconditioning operator. Note that this is not the operator defined in `precond.m` but rather

   $$\texttt{precond} = D^2. \tag{2.12}$$

3. Define the operator

   $$\tilde{P} = D \circ P \tag{2.13}$$

   to be the preconditioned version of the pseudo-polar Fourier transform, and compute

   $$\tilde{I} = adj\ \tilde{P}(\tilde{Y}). \tag{2.14}$$

4. Define the operator

   $$PtP = adj\ \tilde{P} \circ \tilde{P}. \tag{2.15}$$

   Since the operator D is self-adjoint it follows that

   $$\begin{aligned} PtP &= adj\ (D \circ P) \circ (D \circ P) = \\ &= adj\ P \circ D \circ D \circ P = \\ &= adj\ P \circ D^2 \circ P. \end{aligned} \tag{2.16}$$

   To save computations, the function `precond.m` computes D^2, where D is the preconditioning operator.

   The operator PtP is symmetric and positive definite. Therefore, we can use the function `CG.m` to solve the equation

   $$\tilde{I} = PtP(im). \tag{2.17}$$

We will next verify the im solves the original equation $Y = P(im)$. According to Eq. 2.17

$$\tilde{I} = PtP(im). \tag{2.18}$$

Substituting Eqs. 2.14 and 2.15 in 2.18 gives

$$adj\ \tilde{P}(\tilde{Y}) = (adj\ \tilde{P} \circ \tilde{P})(im). \tag{2.19}$$

Hence, it follows that $\tilde{Y}$ satisfies

$$\tilde{Y} = \tilde{P}(im). \tag{2.20}$$

Using the definition of $\tilde{Y}$ and $\tilde{P}$, given in Eqs. 2.11 and 2.13, we obtain that

$$D(Y) = D(P(im)). \tag{2.21}$$

Since the preconditioning operator D is an element-by-element multiplication it follows that

$$Y = P(im). \tag{2.22}$$

2.3.2 Inversion parameters

The inversion function `ippft.m` uses various parameters, which control the accuracy of the inversion, the time consumption of the algorithm, and the level of progress messages. These parameters are identical to the parameters used by the underlying conjugate-gradient algorithm (`CG.m`).

`ippft.m` accepts following parameters:

`pp1`,`pp2` (Mandatory) The pseudo-polar samples `pp1` and `pp2`.

`ErrTol` (Optional) Error tolerance of the conjugate-gradient solver. The conjugate-gradient algorithm terminates if the residual error goes below this threshold. If not specified, the default value is 10^{-2}.

`MaxIts` (Optional) Maximum number of iterations to be used by the conjugate-gradient algorithm. If not specified, the default value is 10.

`verbose` (Optional) If a value different from 0 is specified, the function displays verbose conjugate-gradient information. The value 0 will suppress verbose messages.

The function `ippft.m` returns the reconstructed image as well as various convergence parameters as described in 5.25.

2.3.3 Example

The following code demonstrates how to compute the pseudo-polar transform and then reconstructing back the original image.

```
im = magic(64);
[pp1,pp2] = PPFT(im);
im2 = ippft(pp1,pp2,1.e-3,15,1);
```

This code creates an image `im` as a magic square of size 64×64. It then computes the pseudo-polar Fourier transform of the image `im` and stores the result in the arrays `pp1` and `pp2`. Finally, the code reconstructs the image `im2` from `pp1` and `pp2` by using the inverse pseudo-polar Fourier transform. The inverse pseudo-polar transform `ippft.m` uses error tolerance of 10^{-3}, allowed to perform up to 15 iterations, and required to print verbose progress information.

Executing the code above produces the following output:

```
Iteration  1:  Residual error=1.0063078e+003
Iteration  2:  Residual error=1.7512995e+002
Iteration  3:  Residual error=2.8871289e+001
Iteration  4:  Residual error=3.1772645e+000
Iteration  5:  Residual error=4.3737801e-001
Iteration  6:  Residual error=7.4718024e-002
Iteration  7:  Residual error=1.1149069e-002
Iteration  8:  Residual error=2.6761862e-003
Iteration  9:  Residual error=5.4707409e-004
```

As we can see from the execution log, the inversion algorithm converged to the required error tolerance (10^{-3}) within 9 iterations. The residual error at the end of the computation is 5.47×10^{-4}. If we check the absolute error between `im` and `im2` using the following code:

```
err = abs(im-im2);
max(err(:))

ans =

    9.6128e-004
```

we see that the absolute error is less the 10^{-3}

Chapter 3

Computing the discrete Radon transform

3.1 Getting started

We compute the 2-D discrete Radon transform by calling the function `Radon.m` for the given image. For example, type the following lines:

```
im = magic(64);
[pp1,pp2] = Radon(im);
```

The first line creates a magic square `im` of size 64×64, which we use as our input image. We then call the function `Radon` on the image `im`, which returns two arrays (in this case each of size 129×65) with the discrete Radon samples of `im`. We will explain the structure and content of `pp1` and `pp2` in section 3.2.1.

To restore `im` from `pp1` and `pp2` we use the function `iRadon` that computes the inverse discrete Radon transform:

```
b = iRadon(pp1,pp2);
```

The function `iRadon` uses an iterative algorithm to recover `im` from `pp1` and `pp2` with any required accuracy. When called without additional arguments (except `pp1` and `pp2`), the function uses the algorithm's default parameters as described in 5.26.

3.2 The forward transform

The function Radon.m computes the forward 2-D discrete Radon transform defined in [1]. It requires $O(n^2 \log n)$ operations, where n^2 is the size of the input image.

The signature of the function Radon.m is

```
function [res1,res2]=Radon(im);
```

where im is a 2-D square image with even side. If given invalid arguments, the function aborts with an error message. For example:

```
im = magic(3);
Radon(im);
??? Error using ==> optimizedppft
Input image must have even side

Error in ==> C:\Private\University\MyFuncs\Radon\Radon.m
On line 26  ==> [res1,res2] = optimizedppft(im);
```

3.2.1 Output arrays

For an image of size $n \times n$, the function Radon.m returns two arrays of size $2n+1 \times n+1$. These arrays, named res1 and res2, contain the samples of the discrete Radon transform for basically horizontal and vertical lines, respectively. See [1] for a dentition of basically horizontal and vertical lines.

Each entry (k,l) in res1 and res2 represents the discrete Radon transform for a line determined by the parameters k and l. For the array res1 (basically horizontal line) this line is given by

$$y = \frac{2l}{n}x + k \quad -n/2 \le l \le n/2 \quad -n \le k \le n$$

and for the array res2 (basically vertical line) it is given by

$$x = \frac{2l}{n}y + k \quad -n/2 \le l \le n/2 \quad -n \le k \le n.$$

Specifically, the l^{th} column of res1 represents lines with slopes

$$\theta = \arctan(2l/n) \qquad l = -n/2 \ldots n/2$$

or, when using angles, lines with slopes between $-\pi/4$ and $\pi/4$. Similarly, the l^{th} column of `res2` represents lines with slopes

$$\theta = \pi/2 - \arctan(2l/n) \qquad l = -n/2 \ldots n/2$$

or when using angles, slopes between $3\pi/4$ and $\pi/4$ (the reverse angles ordering is intentional).

Note that the transform values that correspond to lines with slopes $-\pi/4$ and $\pi/4$ appear in both arrays.

The structure of the arrays `res1` and `res2` is summarized in Figs 3.1 and 3.2. Each row in `res1` and `res2` (fixed k) represents a family of lines with fixed intercept and variable slope. Similarly, a column in both arrays (fixed l) represents a family of lines with fixed direction (slope) and variable intercept.

The $2n+1$ rows of `res1` and `res2` represent the intercepts $-n, \ldots, n$ according to the following setting: row number 1 refers to lines with intercept n, row number 2 refers to lines with intercept $n-1$,..., row number $2n+1$ refers to lines with intercept $-n$.

Although different from Matlab's indexing scheme, we will assume that the columns of `res1` and `res2` are indexed from $-n/2$ to $n/2$. We refer to such indexing scheme as "aliased indexing". Specifically, we refer to column 1 in `res1` and `res2` as column $-n/2$, to column 2 as column $-n/2+1$,..., and to column $n+1$ as column $n/2$ (`res1` and `res2` have $n+1$ columns). We simply map Matlab's indexing, from 1 to $n+1$, into symmetric indexing from $-n/2$ to $n/2$. The function `toUnaliasedIdx` (see 5.42) is used throughout the code to perform this mapping.

3.2.2 Warnings and rounding errors

The discrete Radon transform maps real-valued images into real valued arrays. Therefore, the code of the function `Radon.m` contains assertions to verify that the output arrays are real-valued. When the imaginary components in the output exceed a threshold (specified in the code), a warning will be displayed. These assertions are useful in detecting software bugs due to code changes.

However, small imaginary values are possible even when the function `Radon.m` works correctly. This is due to round off errors as a result of Matlab's

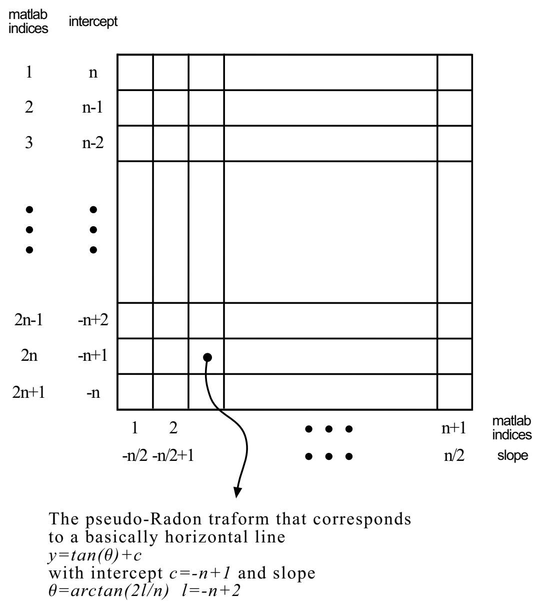

Figure 3.1: The output array `res1`

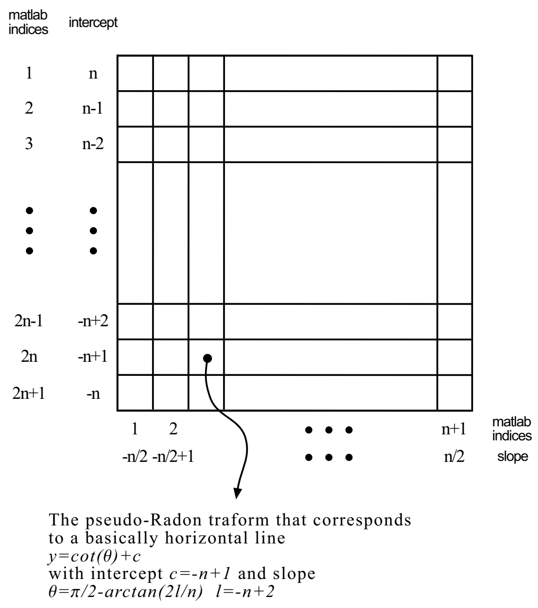
matlab indices
intercept
1
n
2
n-1
3
n-2
2n-1
-n+2
2n
-n+1
2n+1
-n
1 2
n+1
matlab indices
-n/2 -n/2+1
n/2
slope
The pseudo-Radon traform that corresponds
to a basically horizontal line
y=cot(θ)+c
with intercept c=-n+1 and slope
θ=π/2-arctan(2l/n) l=-n+2

Figure 3.2: The output array `res1`

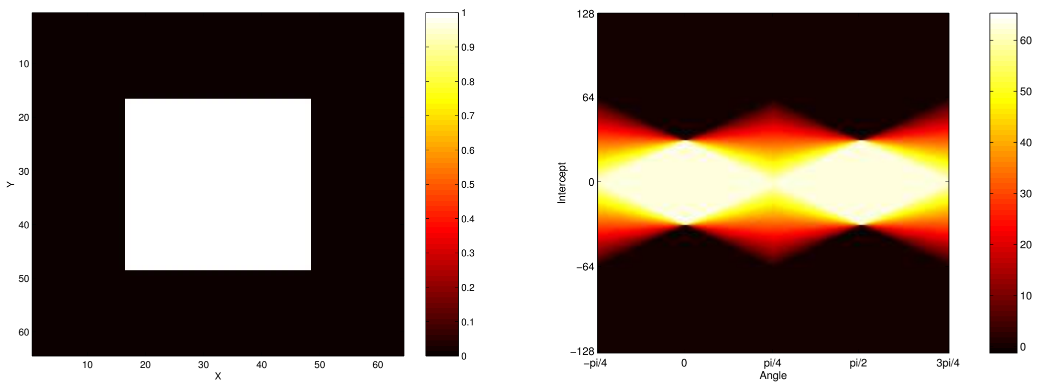

Figure 3.3: Discrete Radon transform of a square

finite accuracy. In such cases, the threshold of acceptable imaginary values should be tuned in the code of `Radon.m`.

To eliminate these small imaginary components in the output, the function `Radon.m` performs truncation of imaginary components before returning the results. In the code this is performed by the lines:

```
res1 = real(res1);
res2 = real(res2);
```

3.3 Examples

We will give examples of the application of the 2-D discrete Radon transform to several test images.

Figure 3.3 describes the applications of the transform to a square of size 64×64 centered within an image of size 128×128. The image is generated by the function `RadonSquare.m` given in A.1.

Figure 3.4 describes the applications of the 2-D discrete Radon to a circle with radius 32 centered with in an image of size 128×128 . The image is generated by the function `RadonCircle.m` given in A.2.

The two arrays returned from the function `Radon.m` are combined using the function `combinePPsectors.m`. This function takes the two arrays returned from `Radon.m` and combines them into a single array while preserving increasing angle ordering from $-\pi/4$ to $3\pi/4$. See 5.12 for description of `combinePPsectors.m`.

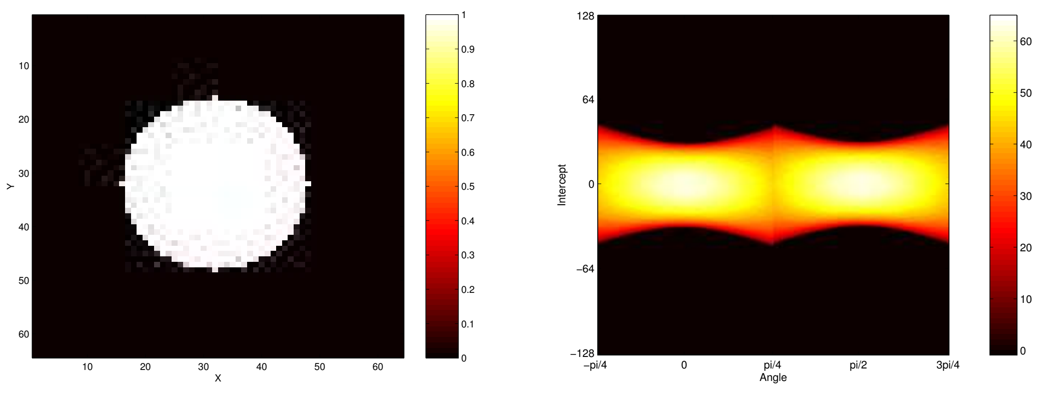

Figure 3.4: Discrete Radon transform of a circle

3.4 The inverse transform

The inverse discrete Radon transform is computed by the function `iRadon.m`. The functions accepts two Radon sectors `res1` and `res2` and reconstructs an image `im` such that

```
[res1,res2] = Radon(im);
```

The arrays `res1` and `res2` must be 2-D arrays of size $(2n+1) \times (n+1)$ where $n \times n$ is the size of the reconstructed image. If the sizes of the input arrays do not conform these requirements, the function will abort and display an error message.

3.4.1 The inversion algorithm

The inversion algorithm of the 2-D discrete Radon transform is based on the discrete Fourier slice theorem and on the inverse pseudo-polar Fourier transform. The discrete Fourier slice theorem states that the 1-D Fourier transform of the discrete Radon transform along the intercept parameter is equal to the pseudo-polar Fourier transform. See [1] for more information. Therefore, to invert the discrete Radon transform we transform it into the pseudo-polar Fourier transform by applying 1-D Fourier transform to the Radon transform, and we then use the function that inverts the pseudo-polar Fourier transform. The function that inverts the pseudo-polar Fourier transform is described in section 2.3.

3.4.2 Inversion parameters

The inversion function `iRadon.m` uses various parameters, which control the accuracy of the inversion, the time consumption of the algorithm, and the level of progress messages. These parameters are identical to the parameters used by the function `ippft.m`. See section 2.3.2 for more information.

The function `iRadon.m` returns the reconstructed image as well as various convergence parameters as described in 5.26.

3.4.3 Example

The following code demonstrates how to compute the discrete Radon transform and then reconstructing back the original image.

```
im = magic(64);
[res1,res2] = Radon(im);
im2 = iRadon(res1,res2,1.e-3,15,1);
```

The code above creates an image `im` as a magic square of size 64×64. It then computes the discrete Radon transform of the image `im` and stores the results in the arrays `res1` and `res2`. Finally, the code reconstructs an image `im2`, which is an approximation of `im`, from `res1` and `res2` by using the inverse discrete Radon transform. The inverse discrete Radon transform `iRadon.m` uses error tolerance of 10^{-3}, allowed to perform up to 15 iterations, and required to print verbose progress information.

Executing the code above produces the following output:

```
Iteration  1:  Residual error=1.0063078e+003
Iteration  2:  Residual error=1.7512995e+002
Iteration  3:  Residual error=2.8871289e+001
Iteration  4:  Residual error=3.1772645e+000
Iteration  5:  Residual error=4.3737801e-001
Iteration  6:  Residual error=7.4718024e-002
Iteration  7:  Residual error=1.1149069e-002
Iteration  8:  Residual error=2.6761862e-003
Iteration  9:  Residual error=5.4707409e-004
```

As we can see from the execution log, the inversion algorithm converged to the required error tolerance (10^{-3}) within 9 iterations. The residual error at the end of the computation is 5.47×10^{-4}. The absolute error is given by

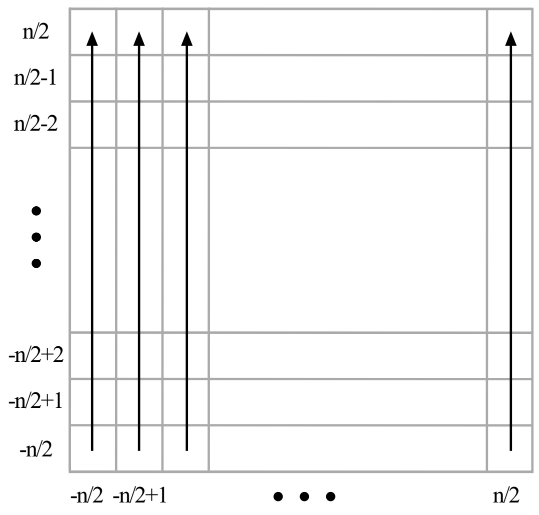

Figure 3.5: Coordinated system used by the algorithm

```
err = abs(im-im2);
max(err(:))

ans =

    9.6128e-004
```

3.5 Geometrical considerations

3.5.1 Coordinate systems

The 2-D discrete Radon transform, defined in [1], assumes that the image is placed on the XY-plane using a cartesian coordinate system. This imposes on the image a grid from $-n/2$ to $n/2$ centered around 0. Furthermore, it dictates a processing order from bottom to top. See Fig. 3.5 for illustration.

Conversely, Matlab always assumes the origin to be at the upper left corner of a matrix (image). This causes processing to be from top to bottom (from low indices to high indices), as illustrated in Fig. 3.6. When translating Matlab's grid to the grid $[-n/2, n/2] \times [-n/2, n/2]$, this imposes on the image a grid whose upper-left corner is $(-n/2, -n/2)$ (due to processing order from top to bottom). Therefore, before applying the Radon function to the input

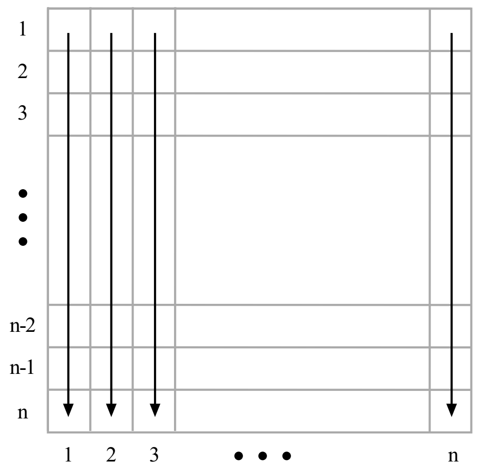

Figure 3.6: Coordinated system used by the algorithm

image, we flip it around the horizontal axis, such that the mathematical coordinate $(-n/2, -n/2)$ of the image becomes the upper-left corner. This explains the call to `flipud` in the function `PPFT.m`.

We will demonstrate the flip explained above using an example. Consider the line given in Fig. 3.7. The slope of the line is $\pi/4$ and therefore, in the Radon domain, its energy should be concentrated around $\pi/4$. Applying the discrete Radon transform, we see, in Fig. 3.8, that the energy is indeed concentrated around $\pi/4$. Now, if we process this line without the flip, we get the Radon image given in Fig. 3.9. As we can see, the energy of this image is concentrated around $-\pi/4$ and $3\pi/4$. The reason is that when processing a line with slope $\pi/4$ from top to bottom (as Matlab processes images), the line appears as it has a slope of $-\pi/4$. Such line has a Radon image whose energy is concentrated around $-\pi/4$ and $3\pi/4$ as illustrated in Fig. 3.9.

Thus, to align Matlab's geometry with the standard XY-plane geometry, we must `flipud` the input image before processing it.

3.5.2 Geometry of processed lines

The continuous Radon transform is defined for all lines with angle $\theta \in [-\pi/2, \pi/2)$ and distance s from the origin. The discrete Radon transform uses a different geometry to define lines. For lines with slope $[-\pi/4, \pi4]$, we

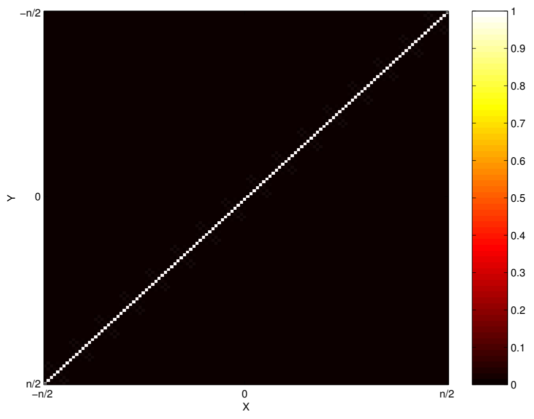

Figure 3.7: Diagonal line (slope=$\pi/4$)

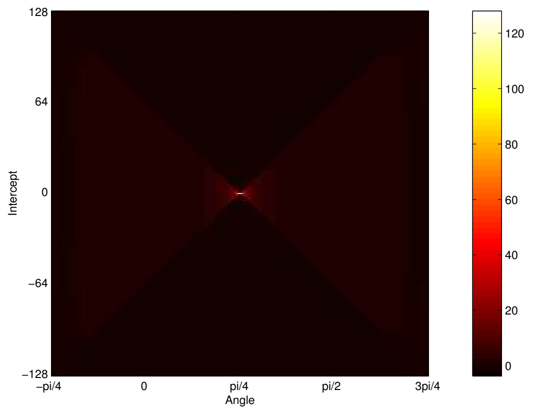

Figure 3.8: Discrete Radon of a line with slope $\pi/4$

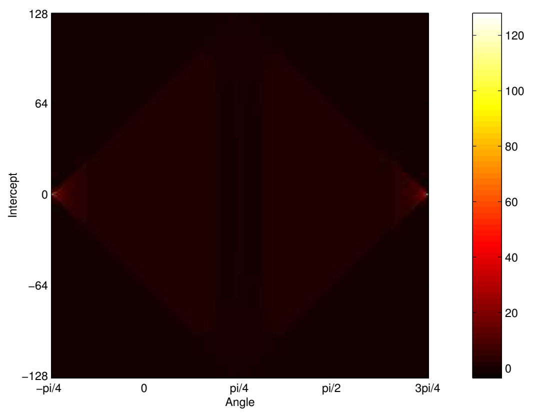

Figure 3.9: Discrete Radon with **no flip** of a line with slope $\pi/4$

define the discrete Radon transform along lines of the form

$$y = (\tan\theta)x + t \qquad t \in \{-n, \dots, n\}. \tag{3.1}$$

For lines with slope $[\pi/4, 3\pi/4]$, we define the discrete Radon transform along lines of the form

$$x = (\cot\theta)y + t \qquad t \in \{-n, \dots, n\}. \tag{3.2}$$

As we can see from Eqs. 3.1 and 3.2, the parameter t has a different meaning for each of the line types. For lines with slope $\theta \in [-\pi/4, \pi 4]$, the parameter t represents the intercept along the y-axis. For lines with slope $\theta \in [\pi/4, 3\pi 4]$, the parameter t represents the intercept along the x-axis. See Figs. 3.10 and 3.11.

Therefore, each of the sectors `res1` and `res2`, returned from the function `Radon.m`, represents a different geometry. Values in `res1` correspond to lines of the form given in Eq. 3.1. Values in `res2` correspond to lines of the form given in Eq. 3.2.

The function `combinePPsectors.m` combines the two discrete Radon sectors `res1` and `res2` and returns a single array in which the angles are ordered from $-\pi/4$ to $3\pi/4$. More specifically, the angles are ordered

$$-\pi/4, \dots, \pi/4\ , \pi/4, \dots, 3\pi/4.$$

However, the function does not adjust the meaning of the parameter t. This means that for the range $-\pi/4 \dots \pi/4$ the parameter t measures the intercept along the y-axis and for the range $\pi/4 \dots 3\pi/4$ it measures the intercept along the x-axis.

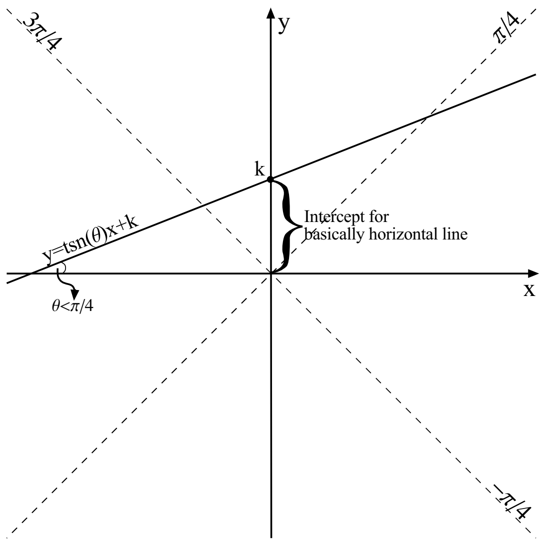

Figure 3.10: Intercept for basically horizontal line

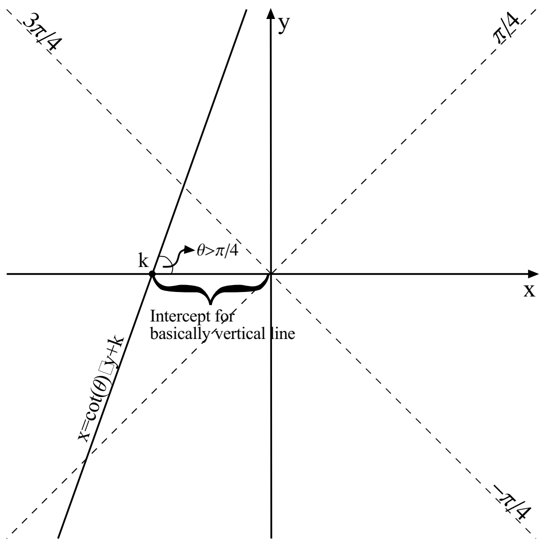

Figure 3.11: Intercept for basically vertical line

3.6 Working with images

For the following discussion, we will assume that an image is a matrix whose entries are integers in the range $[0, 255]$.

When inverting the discrete Radon transform of an image, it is possible to exploit the image's special properties to achieve exact convergence in fewer iterations. Calling the inversion function `iRaon.m` with error parameter of 10^{-1} yields a reconstructed image whose absolute difference from the original image is of order of 10^{-1}. Discarding the imaginary components of the reconstructed image and rounding to the closest integer will give exactly the original image.

For example, consider the image given in Fig. 3.12 and its discrete Radon image given in Fig. 3.13. Figure 3.12 is generated by

```
im = imread('brain.bmp');
imagesc(im);
```

and Fig. 3.13 is generated by

```
[res1,res2] = Radon(im);
imagesc(combinePPsectors(res1,res2));
```

(we ignored the typesetting of labels of axes).

If we execute the inversion command:

```
im2 = iRadon(res1,res2,1.e-1,10,1);
```

we get the following output:

```
Iteration  1:  Residual error=3.4219951e+001
Iteration  2:  Residual error=1.8288970e+000
Iteration  3:  Residual error=1.2325605e-001
Iteration  4:  Residual error=2.3425353e-002
```

By discarding the imaginary components of the reconstructed image and rounding it to integer values we obtain exactly the original image. The is demonstrated by the following code:

```
im2 = round(real(im2));
max(max(im-im2));

ans =

  0
```

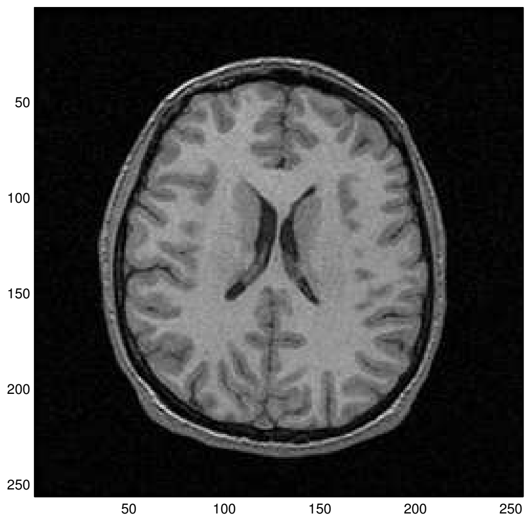
Figure 3.12: Sample image

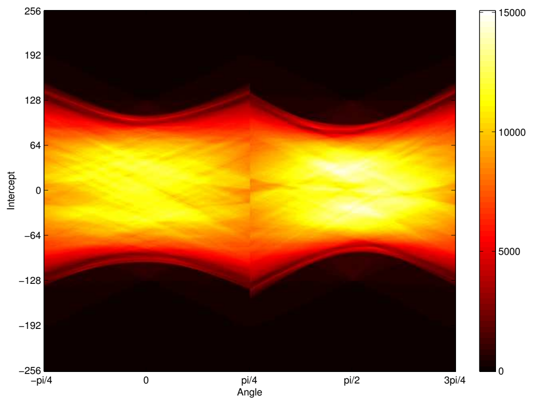

Figure 3.13: discrete Radon transform of the sample image

Chapter 4

Command summary

This section gives a brief description of each of the functions in the package.
We define four types of functions:

Main functions The primary functions in the package. These functions should be called by the user and include:

- Computing the discrete Radon transform and its inverse.
- Computing the pseudo-polar Fourier transform and its inverse.
- Optimized versions of the above functions.

Helper functions Auxiliary functions used by the main functions. These auxiliary functions are used to increase modularity, readability, and maintainability of the code.

Reference functions Implementation of the key functions in the package according to their direct mathematical definition. Typically, this implementation is much slower.

The reference functions are used to test the correctness of the fast implementations. For example, the reference implementation for the discrete Radon transform (`slowRadon.m`) requires $O(n^3)$ operations while the fast implementation requires $O(n^2 \log n)$ operations.

Test functions Functions that compare the fast implementation against the reference implementation for automatic validity test.

4.1 Main functions

`ippft.m`	Inverse 2-D pseudo-polar Fourier transform
`iRadon.m`	Inverse 2-D discrete Radon transform
`OptimizedPPFT.m`	2-D fast pseudo-polar Fourier transform (optimized version)
`PPFT.m`	2-D fast pseudo-polar Fourier transform
`Radon.m`	2-D discrete Radon transform

4.2 Helper functions

`adjGKN.m`	Adjoint operator of `GKN.m`
`adjPPFT.m`	Adjoint pseudo-polar Fourier transform
`adjRadon.m`	Adjoint discrete Radon transform
`cfft.m`	One dimensional aliased discrete Fourier transform
`cfft2.m`	Two dimensional aliased discrete Fourier transform
`cfftd.m`	One dimensional aliased discrete Fourier transform along dimension d
`cfrft.m`	One dimensional aliased fractional Fourier transform
`cfrftV2.m`	One dimensional aliased fractional Fourier transform
`CG.m`	Conjugate-gradient solver
`combinePPsectors.m`	Combine discrete Radon sectors
`dirichlet.m`	Dirichlet kernel
`extractPPsectors.m`	Separate discrete Radon sectors
`fftshift1d.m`	Shift zero-frequency component to center of spectrum
`GKN.m`	Frequency resampling operator
`hiIdx.m`	Highest index of an aliased sequence

`icfft.m`	Inverse one dimensional aliased discrete Fourier transform
`icfft2.m`	Inverse two dimensional aliased discrete Fourier transform
`icfftd.m`	Inverse one dimensional aliased discrete Fourier transform along dimension d
`iffshift1d`	Inverse FFT shift
`ip.m`	Inner product
`lowIdx.m`	Low index of aliased sequence
`OptimizedadjPPFT.m`	Adjoint pseudo-polar Fourier transform (Optimized version)
`precondadjPPFT.m`	Preconditioned adjoint pseudo-polar Fourier transform
`PtP.m`	Gram Operator of the pseudo-polar Fourier transform
`toUnaliasedIdx.m`	Convert aliased indices to unaliased indices

4.3 Reference functions

`cdft.m`	One dimensional aliased discrete Fourier transform
`cdft2.m`	Two dimensional aliased discrete Fourier transform
`frft.m`	One dimensional aliased fractional Fourier transform
`icdft.m`	Inverse one dimensional aliased discrete Fourier transform
`precond.m`	Preconditioner for the pseudo-polar Fourier
`slowPPFT.m`	2-D pseudo-polar Fourier transform
`slowRadon.m`	2-D discrete Radon transform transform

4.4 Test functions

`testAdj.m`	Test the adjoint operators
`testCfft.m`	Test the Fourier transform functions
`testIRadon.m`	Test the Radon and pseudo-polar inversion functions
`testOptimizedAdjPPFT.m`	Test the function `OptimizedAdjPPFT`
`testRadon.m`	Test the Radon and pseudo-polar functions

Chapter 5

Reference

5.1 `adjGKN.m`

Purpose	Adjoint of the resampling operator $G_{k,n}$
Type	Helper
Syntax	v = adjGKN(w,k)

Description Computes the adjoint of the operator GKN for the vector w and the row k. For a vector w of length n the operator $G_{k,n}$ is defined by

$$G_{k,n}(w) = (U_{2n+1,n+1} \circ F_{2n+1}^{2k/n} \circ E_{n,2n+1} \circ F^{-1})(w)$$

where

$U_{2n+1,n+1}$	Truncation operator from length $2n+1$ to length $n+1$.
$F_{2n+1}^{2k/n}$	Fractional Fourier transform with factor $\alpha = 2k/n$ on a sequence of length $2n+1$.
$E_{n,2n+1}$	Extension (padding) operator from length n to length $2n+1$.
F^{-1}	Inverse discrete Fourier transform.

Therefore *adj* $G_{k,n}$ is given by

$$adj\ G_{k,n} = adj\ F^{-1} \circ adj\ E_{n,2n+1} \circ adj\ F_{2n+1}^{2k/n} \circ adj\ U_{2n+1,n+1}.$$

In the equations above, the length of the input vector to the function `adjGKN` is $n+1$ and the length of the output vector is n. We use these length notations to keep notations consistent with the function `GKN` that maps vectors of length n to vectors of length $n+1$.

Input parameters

w The sequence to resample. Can be of odd or even length.

k The row whose transform is computed.

See [1] for more information.

See also GKN

5.2 `adjPPFT.m`

Purpose Adjoint pseudo-polar Fourier transform

Syntax im = adjPPFT(pp1,pp2)

Type Helper

Description Compute the adjoint operator of the pseudo-polar Fourier transform.

Input parameters

pp1,pp2 The pseudo-polar sections in the pseudo-polar domain. For example, `pp1` and `pp2` are the pseudo-polar sections generated by the function `PPFT`. `pp1` and `pp2` must be of size $(2n+1) \times (n+1)$ as results from PPFT.

See also PPFT, OptimizedPPFT, OptimizedadjPPFT

5.3 `adjRadon.m`

Purpose Adjoint discrete Radon transform

Syntax im = adjRadon(r1,r2)

Type Helper

Description Computes the adjoint Radon transform.

Input parameters

r1,r2 The discrete Radon sections. `r1` and `r2` must be of size $(2n+1) \times (n+1)$ as results from the function `Radon`.

See also Radon, adjPPFT

5.4 `cdft.m`

Purpose One dimensional aliased discrete Fourier transform

Syntax y=cdft(x)

Type Reference

Description Computes the aliased discrete Fourier transform of the sequence `x`. The discrete Fourier transform is computed directly using $O(n^2)$ operations.

The aliased discrete Fourier transform for a sequence x of length n is defined is

$$n = 2l \qquad y_k = \sum_{j=-l}^{l-1} x_j e^{-2\pi\imath jk/n}$$

$$n = 2l+1 \qquad y_k = \sum_{j=-l}^{l} x_j e^{-2\pi\imath jk/n}.$$

Input parameters

x The sequence to transform. Can be of odd or even length.

The function is used as a reference to test the Fourier functions.

See also icdft, cfft

5.5 `cdft2.m`

Purpose Two dimensional aliased discrete Fourier transform

Syntax y=cdft2(x)

Type Reference

Description Computes the 2-D aliased discrete Fourier transform of the image `x`. The DFT is computed directly using $O(n^4)$ operations.

For a $n \times n$ image the 2-D aliased DFT is defined by

$$n = 2m \qquad y_{k,l} = \sum_{u=-m}^{m-1} \sum_{v=-m}^{m-1} x_{u,v} e^{-2\pi\imath(uk+vl)/n}$$

$$n = 2m+1 \qquad y_{k,l} = \sum_{u=-m}^{m} \sum_{v=-m}^{m} x_{u,v} e^{-2\pi\imath(uk+vl)/n}.$$

Input parameters

x An image of size $n \times m$ where n and m can be odd or even.

The function supplied for completeness of the library and can be used as a reference to test the fast 2-D Fourier functions.

See also cfft2

5.6 `cfft.m`

Purpose One dimensional aliased discrete Fourier transform

Syntax y=cfft(x)

Type Helper

Description Fast algorithm for Computing the 1-D aliased discrete Fourier transform. The transform is computed using $O(n \log n)$ operations.

The aliased discrete Fourier transform for a sequence x of length n is defined by

$$n = 2l \qquad y_k = \sum_{j=-l}^{l-1} x_j e^{-2\pi \imath jk/n}$$

$$n = 2l+1 \qquad y_k = \sum_{j=-l}^{l} x_j e^{-2\pi \imath jk/n}.$$

Input parameters

x The sequence to transform. Can be of odd or even length. Must be a 1-D vector.

See also icfft, cdft

5.7 `cfft2.m`

Purpose Two dimensional aliased discrete Fourier transform

Syntax y=cfft2(x)

Type Helper

Description Fast algorithm for Computing the 2-D aliased discrete Fourier transform. The transform is computed using $O(n^2 \log n)$ operations.

For a $n \times n$ image the 2-D aliased DFT is defined by

$$\begin{aligned} n &= 2m & \qquad y_{k,l} &= \sum_{u=-m}^{m-1} \sum_{v=-m}^{m-1} x_{u,v} e^{-2\pi\imath(uk+vl)/n} \\ n &= 2m+1 & \qquad y_{k,l} &= \sum_{u=-m}^{m} \sum_{v=-m}^{m} x_{u,v} e^{-2\pi\imath(uk+vl)/n}. \end{aligned}$$

Input parameters

x An image of size $n \times m$ where n and m can be odd or even.

See also icfft2, cdft2

5.8 `cfftd.m`

Purpose One dimensional aliased discrete Fourier transform along dimension *d*

Syntax y=cfftd(x)

Type Helper

Description Computes the 1-D aliased FFT of a multi-dimensional image `x` along dimension `d`.

For an image `im` of size $n_1 \times n_2 \times \ldots \times n_k$, the call `cfftd(im,d)` calls the FFT function $n_1 \times n_2 \times n_{d-1} \times n_{d+1} \times \ldots n_k$ times, each time on a vector with n_d elements.

For example, the call

```
cfftd(cfftd(im,1),2)
```

computes the 2-D discrete Fourier transform of the image `im`.

Input parameters

x Multi-dimensional image to transform.

d Dimension along which to transform.

See also icfftd, cfft

5.9 `cfrft.m`

Purpose One dimensional aliased fractional Fourier transform

Syntax w=cfrft(x,alpha)

Type Helper

Description Fast algorithm for Computing the 1-D aliased fractional Fourier transform. The transform is computed using $O(n \log n)$ operations.

For a sequence x of length n and a parameter α the fractional Fourier transform is defined by

$$\begin{aligned} n &= 2l & y_k &= \sum_{j=-l}^{l-1} x_j e^{-2\pi\imath kj\alpha/n} & -n/2 \le k \le n/2-1 \\ n &= 2l+1 & y_k &= \sum_{j=-l}^{l} x_j e^{-2\pi\imath kj\alpha/n} & -n/2 \le k \le n/2. \end{aligned}$$

Input parameters

x The sequence to transform. Can be of odd or even length. Must be a 1-D row vector.

alpha Frequency spacing parameter α. Can be any real-valued number.

See also cfrftV2, frft

5.10 `cfrftV2.m`

Purpose One dimensional aliased fractional Fourier transform

Syntax w=cfrftV2(x,alpha)

Type Helper

Description Fast algorithm for Computing the 1-D aliased fractional Fourier transform. The transform is computed using $O(n \log n)$ operations.

This function performs the same operations as `cfrft`. However, it pads intermediate sequences required by the algorithm to length 2^k, as the `FFT` algorithm for these lengths is faster.

Replace calls to `cfrft` with calls to `cfrftV2` if needed.

Input parameters

x	The sequence to transform. Can be of odd or even length. Must be a 1-D row vector.
alpha	Frequency spacing parameter α. Can be any real-valued number.

See also cfrft

5.11 `CG.m`

Purpose Conjugate-gradient solver

Syntax [Y,flag,relres,iter,absres] = CG(X,ErrTol,MaxIts,verbose,RefY)

Type Helper

Description Solves the system $X = PtP(Y)$ using the conjugate gradient method. The operator PtP is defined in `PtP.m`. Modify `PtP.m` to change the operator used by this CG (conjugate gradient) solver. The operator PtP must be hermitian and positive definite.

Input parameters

X A matrix in the range space of the operator `PtP`.

ErrTol (Optional) Error tolerance of the CG method. Default 10^{-9}.

MaxIts (Optional) Maximum number of iterations. Default 10.

versboe (Optional) By default, if more than one output argument is specified, then all output messages are suppressed. Set this flag to any value other than 0 to avoid suppression of output messages.

RefY (Optional) The untransformed matrix Y. Used only for checking absolute error. If not specified, absolute error is not computed.

Output parameters

Y The result matrix of the CG method. This is the estimate of the CG solver to the solution of the system $X = PtP(Y)$.

flag A flag that describes the convergence of the CG method.

0 CG converged to the desired tolerance `ErrTol` within `MaxIts` iterations.

	1 CG did not converge to `ErrTol` within `MaxIts` iterations.
relres	Residual error at the end of the CG method. Computed using max-norm.
iter	The iteration number at which `ErrTol` was achieved. Relevant only if `flag` $\neq 0$.
absres	The absolute error at the end of the CG method. Relevant only if `RefY` was given as a parameter. Computed using max-norm.

See also PtP

5.12 `combinePPsectors.m`

Purpose Combine discrete Radon sectors

Syntax function ppim = combinePPsectors(sec1,sec2)

Type Helper

Description Take two sectors of discrete Radon samples and combine them into a single discrete Radon image.

The values along the columns of `sec1` correspond to angles $\theta = \arctan(2l/n)$, $l = -n/2 \ldots n/2$ (from $-\pi/4$ to $\pi/4$). The values along the columns of `sec2` correspond to angles $\theta = \pi/2 - \arctan(2l/n)$, $l = -n/2 \ldots n/2$ (from $3\pi/4$ to $\pi/4$).

For the combined image `ppim(k,l)`, k is the radius parameter and l is the angle parameter. The angles in the combined matrix `ppim` goes from $-\pi/4$ to $3\pi/4$. More specifically, the angles are ordered as $-\pi/4, \ldots, \pi/4, \pi/4, \ldots, 3\pi/4$.

The functions does not adjust the meaning of the parameter k (pseudo-radius) for each of the sectors. For $-\pi/4, \ldots, \pi/4$ it measures the intercept along the y-axis. For $\pi/4, \ldots, 3\pi/4$ it measures the intercept along the x-axis.

Input parameters

sec1,sec2 Two discrete Radon sections as returned from the function `Radon`. `sec1(k,l)` and `sec2(k,l)` represent samples at “radius” k and “angle” l.

See also extractPPsectors

5.13 `dirichlet.m`

Purpose Dirichlet kernel

Syntax y=dirichlet(t,m)

Type Helper

Description Compute the value of the Dirichlet kernel D_m (of length m) at point t. The Dirichlet kernel D_m is defined by

$$D_m(t) = \frac{\sin(\pi t)}{m \sin(\pi t/m)}.$$

Input parameters

t The point at which to evaluate the Dirichlet kernel ($t \in \mathbb{R}$).

m Length of the Dirichlet kernel.

5.14 `extractPPsectors.m`

Purpose Separate discrete Radon sectors

Syntax [sec1,sec2] = extractPPsectors(ppim)

Type Helper

Description Split a discrete Radon image created by `combinePPsectors` into two discrete Radon sectors.

The values along the columns of `sec1` correspond to angles $\theta = \arctan(2l/n), \quad l = -n/2 \ldots n/2$ (from $-\pi/4$ to $\pi/4$). The values along the columns of `sec2` correspond to angles $\theta = \pi/2 - \arctan(2l/n), \quad l = -n/2 \ldots n/2$ (from $3\pi/4$ to $\pi/4$).

Input parameters

ppim Combined discrete Radon image.

See also combinePPsectors

5.15 `fftshift1d.m`

Purpose Shift zero-frequency component to center of spectrum

Syntax y=fftshift1d(x)

Type Helper

Description A fast implementation of Matlab's `fftshift` for 1-D vectors.

Input parameters

x 1-D vector. Can be of odd or even length.

See also ifftshift1d

5.16 `frft.m`

Purpose One dimensional aliased fractional Fourier transform

Syntax y=frft(x,alpha)

Type Reference

Description Directly computes the 1-D aliased fractional Fourier transform. The transform is computed using $O(n^2)$ operations.

For a sequence x of length n and a parameter α the fractional Fourier transform is defined by

$$\begin{array}{lll} n = 2l & y_k = \sum_{j=-l}^{l-1} x_j e^{-2\pi \imath k j \alpha / n} & -n/2 \leq k \leq n/2 - 1 \\ n = 2l+1 & y_k = \sum_{j=-l}^{l} x_j e^{-2\pi \imath k j \alpha / n} & -n/2 \leq k \leq n/2. \end{array}$$

Input parameters

x	The sequence to transform. Can be of odd or even length.
alpha	Frequency spacing parameter α. Can be any real-valued number.

The result vector is always a row vector.

See also cfrft, cfrftV2

5.17 `GKN.m`

Purpose Pseudo-polar resampling operator

Syntax y=GKN(x,k)

Type Helper

Description The function accepts a sequence of samples in the frequency domain and generates a new sequence whose frequency spacing is $2k/n$. This is an implementation of the the resamlping operator $G_{k,n}$ described in [1].

For a vector x of length n the operator $G_{k,n}$ is defined by

$$G_{k,n}(w) = (U_{2n+1,n+1} \circ F_{2n+1}^{2k/n} \circ E_{n,2n+1} \circ F^{-1})(x)$$

where

$U_{2n+1,n+1}$	Truncation operator from length $2n+1$ to length $n+1$.
$F_{2n+1}^{2k/n}$	Fractional Fourier transform with factor $\alpha = 2k/n$ on a sequence of length $2n+1$.
$E_{n,2n+1}$	Extension (padding) operator from length n to length $2n+1$.
F^{-1}	Inverse discrete Fourier transform.

Input parameters

x The sequence to resample. Can be of odd or even length. Must be a 1-D row vector.

k The row whose transform is being computed. This determines the frequency spacing of the resampled sequence as $2k/n$.

See also adjGKN

5.18 `hiIdx.m`

Purpose Highest index of an aliased sequence

Syntax idx=hiIdx(n)

Type Helper

Description Returns the maximal index for an aliased sequence of length n.

For example, for n=5, the indices of the aliased sequence are

$$-2 \quad -1 \quad 0 \quad 1 \quad 2$$

and hence, `hiIdx(5)` equals 2. Similarly, for n=4, the indices of the aliased sequence are

$$-2 \quad -1 \quad 0 \quad 1$$

and hence, `hiIdx(4)` equals 1.

Input parameters

n Length of the aliased sequence.

See also loIdx, toUnaliasedIdx

5.19 `icdft.m`

Purpose Inverse one dimensional aliased discrete Fourier transform

Syntax y=icdft(x)

Type Reference

Description Computes the inverse DFT of the sequence `x` directly using $O(n^2)$ operations.

The aliased inverse DFT for a sequence x of length n is defined by

$$n = 2l \qquad y_k = \frac{1}{n}\sum_{j=-l}^{l-1} x_j e^{2\pi \imath jk/n}$$

$$n = 2l+1 \qquad y_k = \frac{1}{n}\sum_{j=-l}^{l} x_j e^{2\pi \imath jk/n}.$$

Input parameters

x Vector of frequency domain samples. Can be of odd or even length. Must be a 1-D vector.

The function is used as a reference to test the Fourier functions.

See also cdft, icfft

5.20 `icfft.m`

Purpose Inverse one dimensional aliased discrete Fourier transform

Syntax y=icfft(x)

Type Helper

Description Fast algorithm for Computing the 1-D aliased inverse DFT. The transform is computed using $O(n \log n)$ operations.

The aliased inverse DFT for a sequence x of length n is defined by

$$n = 2l \qquad y_k = \frac{1}{n} \sum_{j=-l}^{l-1} x_j e^{2\pi\imath jk/n}$$

$$n = 2l+1 \qquad y_k = \frac{1}{n} \sum_{j=-l}^{l} x_j e^{2\pi\imath jk/n}.$$

Input parameters

x Vector of frequency domain samples. Can be of odd or even length. Must be a 1-D vector.

See also cfft, icdft

5.21 `icfft2.m`

Purpose Inverse two dimensional aliased discrete Fourier transform

Syntax y=icfft2(x)

Type Helper

Description Fast algorithm for Computing the 2-D aliased inverse DFT. The transform is computed using $O(n^2 \log n)$ operations.

For a $n \times n$ image the 2-D aliased inverse DFT is defined by

$$n = 2m \qquad y_{k,l} = \frac{1}{n^2} \sum_{u=-m}^{m-1} \sum_{v=-m}^{m-1} x_{u,v} e^{2\pi\imath(uk+vl)/n}$$

$$n = 2m+1 \qquad y_{k,l} = \frac{1}{n^2} \sum_{u=-m}^{m} \sum_{v=-m}^{m} x_{u,v} e^{2\pi\imath(uk+vl)/n}.$$

Input parameters

x Matrix of frequency domain samples of size $n \times m$ where n and m can be odd or even.

See also cfft2

5.22 `icffd.m`

Purpose Inverse one dimensional aliased discrete Fourier transform along dimension *d*

Syntax y=icfftd(x)

Type Helper

Description Computes the 1-D aliased inverse FFT of a multi-dimensional image x along dimension d.

For example, the call

```
icfftd(icfftd(im,1),2)
```

computes the 2-D inverse FFT of the image `im`.

Input parameters

x Multi-dimensional frequency image to transform.

d Dimension along which to transform.

See also cfftd, icfft

5.23 `ifftshift1d.m`

Purpose Inverse FFT shift

Syntax y=ifftshift1d(x)

Type Helper

Description A fast implementation of Matlab's `ifftshift` for 1-D vectors.

Input parameters

x 1-D vector. Can be of odd or even length.

See also ifftshift1d

5.24 `ip.m`

Purpose Inner product

Syntax c=ip(a,b)

Type Helper

Description Computes the inner product of two vectors.

The inner product (in the 2-D case) of two images of size $n_1 \times n_2$ is defined by

$$c = \sum_{k=1}^{n_1} \sum_{l=1}^{n_2} a(k,l)\overline{b(k,l)}.$$

Input parameters

a,b Multi-dimensional arrays. `a` and `b` should have the same dimensions.

5.25 `ippft.m`

Purpose Inverse pseudo-polar Fourier transform

Syntax [Y,flag,residual,iter] = ippft(pp1,pp2,ErrTol,MaxIts,verbose)

Type Main

Description Computes the inverse pseudo-polar Fourier transform for the pseudo-polar sectors `pp1` and `pp2`. The inversion algorithm uses the conjugate gradient solver `CG.m`.

Input parameters

pp1,pp2	Pseudo-polar sectors as returned from the function `PPFT`. `pp1` and `pp2` must be of size $(2n+1)\times(n+1)$ (n even).
ErrTol	(Optional) Error tolerance used by the conjugate gradient solver. Default 10^{-2}.
MaxIts	(Optional) Maximum number of iterations. Default 10.
verbose	Display verbose `CG` information. 0 will suppress verbose information. Any non-zero value will display verbose `CG` information.

Output parameters

Y	The reconstructed matrix.
flag	Convergence flag. See `CG` for more information.
residual	Residual error at the end of the inversion.
iter	The iteration number at which `ErrTol` was achieved. Relevant only if `flag` $\neq 0$.

If the inversion does not converge to the desired `ErrTol` within `MaxIts` iterations, the function displays a warning.

See also PPFT, optimizedPPFT, adjPPFT

5.26 `iRadon.m`

Purpose	Inverse discrete Radon transform
Syntax	[Y,flag,residual,iter]=iRadon(res1,res2,ErrTol,MaxIts,verbose)
Type	Main

Description Recover the image whose discrete Radon transform is given by `res1` and `res2`. The inversion algorithm uses the conjugate gradient solver `CG.m`.

Input parameters

res1,res2	Discrete Radon sectors as returned from the function `Radon`. `res1` and `res2` must be of size $(2n+1)\times(n+1)$ (n even).
ErrTol	(Optional) Error tolerance used by the conjugate gradient solver. Default 10^{-2}.
MaxIts	(Optional) Maximum number of iterations. Default 10.
verbose	Display verbose `CG` information. 0 will suppress verbose information. Any non-zero value will display verbose `CG` information.

Output parameters

Y	The reconstructed matrix.
flag	Convergence flag. See `CG` for more information.
residual	Residual error at the end of the inversion.
iter	The iteration number at which `ErrTol` was achieved. Relevant only if `flag` $\neq 0$.

If the inversion does not converge to the desired `ErrTol` within `MaxIts` iterations, the function displays a warning.

See also Radon, PPFT

5.27 `lowIdx.m`

Purpose Lowest index of an aliased sequence

Syntax idx=lowIdx(n)

Type Helper

Description Returns the minimal index for an aliased sequence of length n.

For example, for n=5, the indices of the aliased sequence are

$$-2 \quad -1 \quad 0 \quad 1 \quad 2$$

and hence, `lowIdx(5)` equals -2. Similarly, for n=4, the indices of the aliased sequence are

$$-2 \quad -1 \quad 0 \quad 1$$

and hence, `lowIdx(4)` equals -2.

Input parameters

n Length of the aliased sequence.

See also hiIdx, toUnaliasedIdx

5.28 `OptimizedadjPPFT.m`

Purpose	Adjoint pseudo-polar Fourier transform.
Syntax	im = OptimizedAdjPPFT(pp1,pp2)
Type	Helper
Description	Optimized version of `adjPPFT`.

Input parameters

pp1,pp2 The pseudo-polar sections. For example, `pp1` and `pp2` are the pseudo-polar sections generated by the function `PPFT`. `pp1` and `pp2` must be of size $(2n + 1) \times (n + 1)$ (n even).

See also PPFT, OptimizedPPFT, adjPPFT

5.29 `OptimizedPPFT.m`

Purpose Pseudo-polar Fourier transform

Syntax [res1,res2] = OptimizedPPFT(im)

Type Main

Description Optimized version of `PPFT`. The computation requires $O(n^2 \log n)$ operations as `PPFT` but uses further optimizations to reduce the operation count.

Input parameters

im The image to transform. Must be square with even side.

See also PPFT

5.30 `PPFT.m`

Purpose Pseudo-polar Fourier transform

Syntax [re1,res2] = PPFT(im)

Type Main

Description Fast algorithm for computing the pseudo-polar Fourier transform. The computation requires $O(n^2 \log n)$ operations. The pseudo-polar Fourier transform is defined by

$$PPI_1(k,l) = \widehat{I}(-\frac{2l}{n}k, k) \tag{5.1}$$

$$PPI_2(k,l) = \widehat{I}(k, -\frac{2l}{n}k) \tag{5.2}$$

where $-n \le k \le n, \quad -n/2 \le l \le n/2$ and

$$\widehat{I}(\xi_1,\xi_2) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u,v) e^{-2\pi\imath(\xi_1 u + \xi_2 v)/m}.$$

The function `PPFT` returns `res1` and `res2` (of size $2n+1 \times n+1$) that contain the pseudo-polar Fourier transform of the input image `im`. `res1` contains the values of $PPI_1(k,l)$; `res2` contains the values of $PPI_2(k,l)$. The first argument of `res1` and `res2` corresponds to pseudo-radius; the second argument of `res1` and `res2` corresponds to pseudo-angle.

Angle ordering:

`res1(k,l)` Pseudo-polar fourier transform which corresponds to radius k and angle $\arctan(2l/n)$. l goes from $-n/2$ to $n/2$ and therefore, for a constant k, `res1` corresponds to angles from $-\pi/4$ to $\pi/4$.

`res2(k,l)` Pseudo-polar fourier transform which corresponds to radius k and angle $\pi/2 - \arctan(2l/n)$. l goes from $-n/2$ to $n/2$ and therefore, for a constant k, `res1` corresponds to angles from $3\pi/4$ to $\pi/4$.

Input parameters

im The image to transform. Must be square with even side.

See also optimizedPPFT, ippft

5.31 `precond.m`

Purpose Preconditioner for the pseudo-polar Fourier transform

Syntax Y=precond(X)

Type Reference

Description Explicit form of the preconditioner used to invert the pseudo-polar Fourier transform using the `CG` (conjugate-gradient) solver. The function is not used directly but should be used as a reference. The application of the preconditioner to the adjoint pseudo-polar Fourier transform is incorporated into the function `precondAdjPPFT`. See section 2.3.1 for more information.

Input parameters

X 2-D matrix of size (2n+1)x(n+1).

See also precondAdjPPFT

5.32 `precondadjPPFT.m`

Purpose Preconditioned adjoint pseudo-polar Fourier transform

Syntax im = PrecondAdjPPFT(pp1,pp2)

Type Helper

Description Returns the preconditioned adjoint of the pseudo-polar Fourier transform of `pp1` and `pp2`. Applying `PrecondAdjPPFT` is identical to an application of `precond` followed by a call to `adjPPFT`.

The explicit form of the preconditioner is defined in `precond.m`. See section 2.3.1 for more information.

Input parameters

pp1,pp2 Matrices of size $(2n+1)\times(n+1)$ that represent the the pseudo-polar sections.

See also precond, adjPPFT

5.33 `Radon.m`

Purpose 2-D discrete Radon transform

Syntax [res1,res2]=Radon(im)

Type Main

Description Fast algorithm for computing the 2-D discrete Radon transform. The computation requires $O(n^2 \log n)$ operations.

Returns `res1` and `res2`, each of size $(2n+1) \times (n+1)$, that contain the discrete Radon transform of the input image `im`. `res1` contains the values which correspond to rays with intercept $k = -n \ldots n$ and angles $\theta = \arctan(2l/n)$ $l = -n/2 \ldots n/2$ (from $-\pi/4$ to $\pi/4$). `res2` contains the values which correspond to rays with intercept $k = -n \ldots n$ and angles $\theta = \pi/2 - \arctan(2l/n)$ $l = -n/2 \ldots n/2$ (from $3\pi/4$ to $\pi/4$).

Input parameters

im The image to transform. Must be square with even side.

See 3.2 for more information.

See also iRadon, slowRadon, adjRadon

5.34 `PtP.m`

Purpose Gram Operator of the pseudo-polar Fourier transform

Syntax Y = PtP(X)

Type Helper

Description The function computes $(adj\ P) \circ D^2 \circ P$ where

P the pseudo-polar Fourier transform (`PPFT`)

D the preconditioner for the pseudo-polar Fourier transform

adj P adjoint pseudo-polar Fourier transform (`adjPPFT`).

The conjugate-gradient solver `CG` uses the function `PtP` to solve the equation $Y = PtP(X)$ for a given Y. Modify the function `PtP` to change the operator used by the conjugate-gradient solver `CG`.

Note that the operator PtP must be hermitian and positive defined.

Input parameters

X Matrix of size $n \times n$ (n even).

See also CG

5.35 `slowPPFT.m`

Purpose 2-D pseudo-polar Fourier transform

Syntax [res1,res2]=slowPPFT(im)

Type Reference

Description Computes the pseudo-polar Fourier transform directly using $O(n^4)$ operations.

The output of `slowPPFT` is identical to the output of `PPFT`. See `PPFT` (section 5.30) for more information.

Input parameters

im The image to transform. Must be square with even side.

See also PPFT

5.36 `slowRadon.m`

Purpose 2-D discrete Radon transform

Syntax [res1,res2]=Radon(im)

Type Reference

Description Computes the 2-D discrete Radon transform directly using $O(n^3)$ operations.

The output of `slowRadon` is identical to the output of `Radon`. See `Radon` (section 5.33) for more information.

Input parameters

im The image to transform. Must be square with even side.

For more information see 3.2.

See also Radon

5.37 `testAdj.m`

Purpose Test the adjoint operators

Syntax testAdj

Type Test

Description Tests the functions `adjPPFT` and `adjRadon`. The functions checks if

$$\langle \texttt{PPFT(A)}, B \rangle = \langle A, \texttt{adjPPFT(B)} \rangle$$

for various matrices A and B, where $\langle A, B \rangle$ is the inner-product of A and B.

For each test the function reports the following information:

`n`	Matrix size
`a`	$\langle \texttt{PPFT(A)}, B \rangle$
`b`	$\langle A, \texttt{adjPPFT(B)} \rangle$
`error`	Absolute error $a - b$
`Rel a`	Relative error $(a - b)/a$
`Rel b`	Relative error $(a - b)/b$

See also PPFT, adjPPFT, Radon, adjRadon

5.38 `testCfft.m`

Purpose	Test the Fourier transform functions
Syntax	testCfft
Type	Test
Description	Tests the functions `cfft`, `cfrft`, `icfft`.
See also	cfft, icfft, cfrft, frft

5.39 `testIRadon.m`

Purpose Test the Radon and pseudo-polar inversion functions

Syntax testIRadon

Type Test

Description Tests the correctness and performance of the discrete Radon transform inversion algorithm (`iRadon`). The function also tests the inversion of the pseudo-polar Fourier transform (`ippft`) since inversion the discrete Radon transform uses inversion the pseudo-polar Fourier transform.

See also iRadon, ippft

5.40 `testOptimizedAdjPPFT.m`

Purpose Test the function `OptimizedAdjPPFT`

Syntax testOptimizedAdjPPFT

Type Test

Description The functions checks if

$$\langle \texttt{optimizedPPFT(A)}, B\rangle = \langle A, \texttt{optimizedAdjPPFT(B)}\rangle$$

for various matrices A and B, where $\langle A, B\rangle$ is the inner-product of A and B.

For each test the function reports the following information:

`n`	Matrix size
`a`	$\langle \texttt{optimizedPPFT(A)}, B\rangle$
`b`	$\langle A, \texttt{optimizedAdjPPFT(B)}\rangle$
`error`	Absolute error $a - b$
`Rel a`	Relative error $(a - b)/a$
`Rel b`	Relative error $(a - b)/b$

See also optimizedPPFT, optimizedAdjPPFT

5.41 `testRadon.m`

Purpose Test the Radon and pseudo-polar functions

Syntax testRadon

Type Test

Description Tests the functions `PPFT`, `Optimized PPFT`, and `Radon`.

See also Radon, PPFT, OptimizedPPPT

5.42 `toUnaliasedIdx.m`

Purpose Convert aliased indices to unaliased indices

Syntax y=toUnaliasedIdx(idx,n)

Type Helper

Description Converts an index from the range $-n/2 \dots n/2-1$ to an index in the range $1 \dots n$. Both odd and even values of n are handled.

For example, for `n` $= 5$ and `idx` $= -1$ the call `toUnaliasedIdx(idx,n)` will return 2:

$$\begin{array}{lccccc} \text{aliased indices} & -2 & -1 & 0 & 1 & 2 \\ & & \uparrow & & & \\ \text{unaliased indices} & 1 & 2 & 3 & 4 & 5 \end{array}$$

The index of -1 is 2 when scaled to the range $1 \dots n$.

Input parameters

idx	An index from the range $-n/2 \dots n/2-1$.
n	The range of unaliased indices.

See also loIdx, hiIdx

Appendix A

Figures code

A.1 RadonSquare.m

```
% Plot the discrete Radon transform of a centered square.
% The two Radon sectors pp1 and pp2 are combined.
%
% Yoel Shkolnisky 4/1/03

function RadonSquare

s = square(128);
[pp1,pp2] = Radon(s);
figure;
imagesc(combinePPSectors(pp1,pp2));
colormap(hot);
colorbar;
set(gca,'XTick',[1,64,129,192,258])
set(gca,'XTickLabel',{'-pi/4','0','pi/4','pi/2','3pi/4'})
xlabel('Angle');
set(gca,'YTick',[1,64,128,192,256])
set(gca,'YTickLabel',{'128','64','0','-64','-128'})
ylabel('Intercept');
truesize;
```

A.2 RadonCircle.m

```
% Plot the discrete Radon transform of a centered circle.
% The two Radon sectors pp1 and pp2 are combined.
%
% Yoel Shkolnisky 4/1/03

function RadonCircle

s = circle(128);
[pp1,pp2] = Radon(s);
im = combinePPSectors(pp1,pp2);
figure
imagesc(im);
colormap(hot);
colorbar;
set(gca,'XTick',[1,64,129,192,258])
set(gca,'XTickLabel',{'-pi/4','0','pi/4','pi/2','3pi/4'})
xlabel('Angle');
set(gca,'YTick',[1,64,128,192,256])
set(gca,'YTickLabel',{'128','64','0','-64','-128'})
ylabel('Intercept');
truesize;
```

Bibliography

[1] Yoel Shkolnisky and Amir Averbuch. The 2-D discrete Radon transform. *Fourier Analysis and applications*, Submitted, 2002.