

ASSIGNMENT 14: RADON TRANSFORM IN MATLAB

1. USEFUL MATLAB FUNCTIONS

Phantom image.

```
> P = phantom(n);
```

Here, n specified the size of the phantom image (e.g., $n = 64$, or 128, etc.).

Syntax and Description of the ‘radon’ function:

```
> R = radon(I, theta);
```

This function returns the Radon transform R of the intensity image I for the angle ‘theta’ degrees. I can be of class double, logical, or any integer class.

Recall that the Radon transform is the projection of the image intensity along a radial line oriented at a specific angle. If ‘theta’ is a scalar, then R is a column vector containing the Radon transform for theta degrees. If ‘theta’ is a vector, then R is a matrix in which each column is the Radon transform for one of the angles in ‘theta’. If you omit theta, it defaults to $0 : 179$.

ALGORITHM OF CALCULATING THE RADON TRANSFORM IN MATLAB

The Radon transform of an image is the sum of the Radon transforms of each individual pixel.

The algorithm first divides pixels in the image into four subpixels and projects each subpixel separately, as shown in the Figure 1.

Each subpixel’s contribution is proportionally split into the two nearest bins, according to the distance between the projected location and the bin centers. If the subpixel projection hits the center point of a bin, the bin on the axes gets the full value of the subpixel, or one-fourth the

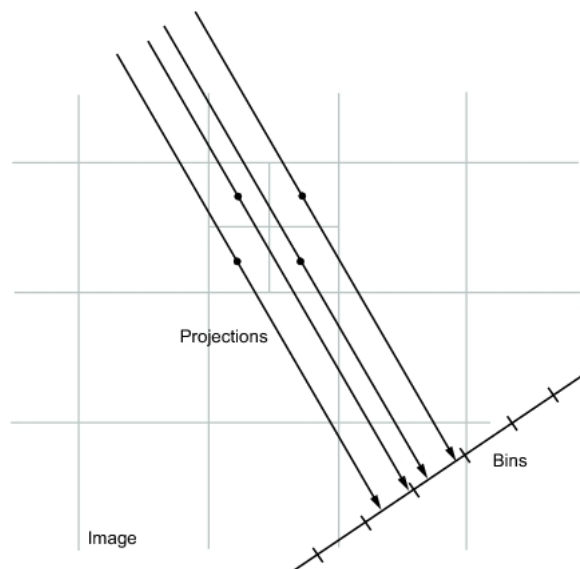


FIGURE 1. Radon Transform algorithm

value of the pixel. If the subpixel projection hits the border between two bins, the subpixel value is split evenly between the bins.

Syntax and Description of the ‘iradon’ function:

```
> I = iradon(R, theta);
```

This function reconstructs the image I from projection data in the two-dimensional array R . The columns of R are parallel beam projection data. ‘iradon’ assumes that the center of rotation is the center point of the projections.

The variable ‘theta’ describes the angles (in degrees) at which the projections were taken. It can be either a vector containing the angles or a scalar specifying ‘Dtheta’, the incremental angle between projections. If ‘theta’ is a vector, it must contain angles with equal spacing between them. If ‘theta’ is a scalar specifying ‘Dtheta’, the projections were taken at angles ‘theta’ = $m \cdot \text{Dtheta}$, where $m = 0, 1, 2, \dots, \text{size}(R, 2) - 1$. If the input is the empty matrix (`[]`), ‘Dtheta’ defaults to $180/\text{size}(R, 2)$.

‘iradon’ uses the filtered back-projection algorithm to perform the inverse Radon transform. The filter is designed directly in the frequency domain and then multiplied by the FFT of the projections. The projections are zero-padded to a power of 2 before filtering to prevent spatial domain aliasing and to speed up the FFT.

```
> I = iradon(P, theta, interp, filter, frequency_scaling, output_size);
```

This extended version specifies parameters to use in the inverse Radon transform. R can be double or single. You can specify any combination of the last four arguments. ‘iradon’ uses default values for any of these arguments that you omit.

- ‘interp’ specifies the type of interpolation to use in the back projection. The available options are listed in order of increasing accuracy and computational complexity:

- ‘nearest’ — Nearest-neighbor interpolation

- ‘linear’ — Linear interpolation (the default)

- ‘spline’ — Spline interpolation

- ‘cubic’ — Shape-preserving piecewise cubic interpolation

- ‘filter’ specifies the filter to use for frequency domain filtering. filter can be any of the strings that specify standard filters.

- ‘Ram-Lak’ — Cropped Ram-Lak or ramp filter. This is the default. The frequency response of this filter is $|f|$. Because this filter is sensitive to noise in the projections, one of the filters listed below might be preferable. These filters multiply the Ram-Lak filter by a window that deemphasizes high frequencies.

- ‘Shepp-Logan’ — Multiplies the Ram-Lak filter by a sinc function

- ‘Cosine’ — Multiplies the Ram-Lak filter by a cosine function

- ‘Hamming’ — Multiplies the Ram-Lak filter by a Hamming window

- ‘Hann’ — Multiplies the Ram-Lak filter by a Hann window

- ‘None’ — No filtering. When you specify this value, ‘iradon’ returns unfiltered backprojection data.

- ‘frequency_scaling’ is a scalar in the range $(0, 1]$ that modifies the filter by rescaling its frequency axis. The default is 1.

• ‘output_size’ is a scalar that specifies the number of rows and columns in the reconstructed image. If ‘output_size’ is not specified, the size is determined from the length of the projections: $output_size = 2 * floor(size(R, 1)/(2 * sqrt(2)))$.

2. ASSIGNMENT

- (1) Create a phantom image first, you can use $n = 64$, or 128, or 256, etc. See what size works better. Plot this image to check how it looks (recall the command ‘imshow’).
- (2) Take the Radon transform of this phantom (you can use the angle $0 : 179$, but check what happen if you use $0 : 359$, or $0 : 89$, or just a single number). Plot the sinogram(s).
- (3) Now we will invert this sinogram:

```
> I = iradon(R,0:179);
```

Next, experiment with filters and interpolations, e.g.,

```
> I = iradon(R,0:179,'linear','none');
```

```
\% This gives an unfiltered back projection
```

Finally, plot what you got and write a short comparison of pictures.

When plotting try to put similar pictures together, for example,

```
> subplot(1,3,1), imshow(P), title('Original')
```

```
> subplot(1,3,2), imshow(I1,[ ]), title('Unfiltered backprojection')
```

```
> subplot(1,3,3), imshow(I2,[ ]), title('Ram-Lak filter')
```

If you have more curiosity, take any other image (try to use the size of power 2 - then no zero padding will be needed for FFT), and try to do the same: take a Radon transform first, see what sinogram you get. Then invert it with various filters.