Finite Radon Transform

Christopher Garnatz

August 12, 2013

Abstract

This paper provides a method for computing the forward Radon transform and the inverse Radon transform in the discrete case. Because the forward transform is casted as a linear algebra operation, the inverse transform can be broken down into a series of faster operations which make use of the fourier and inverse fourier transforms. Using the Cooley-Tukey algorithm, these fourier transforms are improved for efficiency, dramatically reducing the number of operations for bodies with large values of N, as the order of operations is of order $N \log N$, instead of N^2.

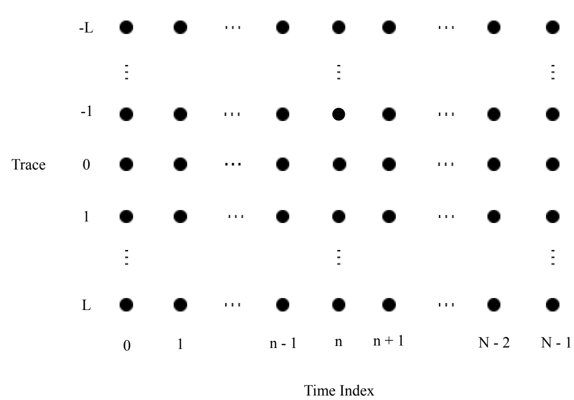
In addition, this paper also references a program written in Python that computes the forward and inverse Radon Transforms, as well as the transform R_m matrices associated with these transforms.

This paper exclusively addresses the discrete case, and outlines an algorithm written in Python for calculating the forward and inverse transforms of a given image.

1 Discrete Radon Transform

For simplicity, the Discrete Radon Transform will be abbreviated as DRT for the remainder of this paper.

Below is a lattice representation of the original body:



Let $x_l(n)$ be a 2-dimensional array, where $-L < l < L$ and $0 \leq n \leq (N-1)$. The quantity $(2L+1)$ gives the total number of traces in the body, which can be thought of as the thickness for simplicity. For a fixed time point n, $x(n)$ is denoted by the following vector:

$$x(n) = \begin{bmatrix} x_{-L}(n) \\ \cdots \\ x_0(n) \\ \cdots \\ x_L(n) \end{bmatrix}$$

We assume $x(n)$ is defined $\forall n \in \mathbb{Z}$, and that $x(n)$ is a periodic vector sequence with period N, so $x(n) = x(N+n)$. Stacking the column vectors in a row from $x(0)$ to $x(N-1)$ produces a matrix with a one-to-one correspondence with the original lattice of the body, with the entry $x_{l,n}$ corresponding to the lattice point at (l,n). The full matrix x is given as follows:

$$x = \begin{bmatrix} | & | & \cdots & | \\ x(0) & x(1) & \cdots & x(N-1) \\ | & | & \cdots & | \end{bmatrix}$$

Thus, to recover the original structure of the body, the inversion procedure attempts to recover these $x(n)$ vectors. This procedure is nontrivial because $x(n)$ cannot be directly determined from the body in question. Instead, scans of the body, whose values can be calculated, are used to recover the values in the $x(n)$ vectors, thus determining the physical makeup of the body in question. These scans are taken by (finish this section)

We define the forward Radon transform for x as follows:

$$y(n) = \sum_{m=-M}^{m=M} R_m \cdot x(n+m) \tag{1}$$

where $2M+1 \leq N$ and R_m are $(2J+1+J') \times (2L+1)$ transform matrices. J gives the maximum slope of scans used in the inversion procedure, J' gives the number of row sums used, and, for a fixed n, M gives the number of adjacent vectors to $x(n)$ used to compute the output vector, $y(n)$. For a fixed time point n, $y(n)$ is denoted by the following vector:

$$y(n) = \begin{bmatrix} y_{-J}(n) \\ \cdots \\ y_0(n) \\ \cdots \\ y_{J+J'}(n) \end{bmatrix}$$

where J is the magnitude of the maximum slope involved in the scan, and J' is the number of row sums taken in the scan. In the accompanying Python program, $J' = (2L+1)$, which accounts for scanning each row in the original body. If $J' = (2L+1)$, then a $(2L+1)$ identity matrix can be appended to the bottom of each R_m matrix.

Thus, y is a $(2J+1+J') \times N$ matrix, and in the special case involving all row scans, y is a $2(J+L+1) \times N$ matrix.

2 Inverse Radon Transform

The DRT is inverted by treating the inversion process as a linear algebra problem. Consider the adjoint matrices R^*, defined as $R_i^* = \mathrm{adj}(R_i)$ for $i = -J, ..., J+J'$. The transform given by R^* is as follows:

$$z(n) = \sum_{m=-M}^{m=M} R_{-m}^* \cdot y(n+m) \tag{2}$$

We now introduce the matrix $\mathbf{H} = \mathbf{R}^* \cdot \mathbf{R}$. The transform associated with H mimics the operations of the DRT and adjoint transform. The transform is:

$$z(n) = \sum_{m=-2M}^{m=2M} H_m \cdot x(n+m) \tag{3}$$

where the H_m matrices are defined as follows:

$$H_m = \sum_{m'=-M}^{m=M-m} R_{m'}^* \cdot R_{m'+m} \tag{4}$$

and

$$H_{-m} = H_m^* \tag{5}$$

for $m = 0, ..., 2M$.

Now, we introduce a key lemma that includes the Discrete Fourier Transforms (DFTs) of H and Z.

Lemma 1 If z(n) is as defined in equation (3), and x is a periodic vector sequence with period N, then,

$$\hat{z}(k) = \hat{H}(k) \cdot \hat{x}(k) \tag{6}$$

for k = 0, 1, ..., N - 1. The DFTs of z, x, and H are defined as follows:

$$\hat{z}(k) = \sum_{n=0}^{n=N-1} z(n) e^{2\pi i (nk/N)} \tag{7}$$

$$\hat{x}(k) = \sum_{n=0}^{n=N-1} x(n) e^{2\pi i (nk/N)} \tag{8}$$

$$\hat{H}(k) = \sum_{m=-2M}^{m=2M} H_m e^{-2\pi i(mk/N)} \tag{9}$$

Note: the difference in sign on the H is attributed to a physical property in seismology.

Proof (Beylkin) Given equation (3), we apply the DFT to both sides to get

$$\hat{z}(k) = \sum_{m=-2M}^{m=2M} H_m \sum_{n=0}^{n=N-1} z(n) e^{2\pi i(nk/N)}$$

or

$$\hat{z}(k) = \sum_{m=-2M}^{m=2M} H_m e^{-2\pi i(mk/N)} \sum_{\tilde{n}=m}^{\tilde{n}=N+m-1} z(\tilde{n}) e^{2\pi i(\tilde{n}k/N)}$$

The assumption that $x(n)$ is periodic with period N implies that

$$\sum_{\tilde{n}=0}^{\tilde{n}=m-1} z(\tilde{n}) e^{2\pi i(\tilde{n}k/N)} = \sum_{\tilde{n}=N}^{\tilde{n}=N+m-1} z(\tilde{n}) e^{2\pi i(\tilde{n}k/N)}$$

for $m \geq 1$. A similar identity holds when $m \leq -1$ by changing N on the right summation to $-N$, and thus

$$\hat{z}(k) = \sum_{m=-2M}^{m=2M} H_m e^{-2\pi i(mk/N)} \sum_{n=0}^{n=N-1} x(n) e^{2\pi i(nk/N)}$$

which is exactly Lemma 1 in equation (6).

Because $x(n)$ is a real vector sequence, DFT of x contains complex conjugation. Namely, $\hat{x}(N-k) = \bar{\hat{x}}(k)$, for $k = 0, 1, ..., N-1$. This means only values of k in the range $[0, N/2]$ if N is even, $[0, (N-1)/2]$ if N is odd, must be considered. This dramatically cuts down the number of computations, especially for large values of N This begins the discussion of invertible frequencies, k.

Definition The DRT given in equation (1) is uniquely invertible within the normalized frequency band $[k_{min}/N, kmax/N]$ for $0 \leq k_{min}/N$ and $k_{max}/N \leq .5$ if $\forall k = k_{min}, ..., k_{max}$

$$\det(\hat{H}(k)) \neq 0 \tag{10}$$

In summary, the algorithm for inverting the DRT, y, of a given body, x, given transform matrices R is

1. Compute the R^* adjoint transform matrices of R $\forall m$

2. Find the adjoint transform of y using the formula

$$z(n) = \sum_{m=-M}^{m=M} R^*_{-m} \cdot y(n+m)$$

3. Compute the $(2L+1) \times (2L+1)$ H_m matrices from the R and R^* matrices for $m = -2M, ..., 2M$

4. Compute the DFTs of z and H, $\hat{z}$ and $\hat{H}$

5. Compute $\hat{H}^{-1}(k) \cdot \hat{z}(k) = \hat{x}(k)$ for invertible values of k

6. Compute the IDFT of $\hat{x}(k)$ for $k = 0, 1, ..., (N-1)$ to recover the original image, x

3 Python Code

3.1 R_m Transform Matrix Maker

This first function, Rm, in this program takes two parameters, J and L, and $J, L \in \mathbb{Z}$. L is related to the number of distinct traces of the body, as $2L+1$ gives the total number of traces. J gives the bound for the maximum and minimum magnitude of the slopes, for a total of $2J+1$ slopes ranging from $-J$ to J. With these definitions, $M = J \times L$, where M is the number of adjacent vectors used on each side of the given vector for the set of scans. Thus, $2M+1$ gives the total number of vectors used for each set of scans, and it follows that there are $2M+1$ transform matrices, indexed as follows: $R_m, m = -M, ..., 0, ..., M$. These matrices are stored as a list of arrays, seen in the code as "R", with R[0] $= R_{-M}$, ..., R[M] $= R_0$, ..., R[$2M$] $= R_M$. This "R" list is the returned product of the Rm function.

The Rm function implements the following formula

$$(R_m)_{j,l} = \delta_{m,jl} \tag{11}$$

where the second subscript of δ is the product of j and l. The R_m matrices have special indexing for rows and columns: rows are indexed top to bottom from $-J$ to J, columns are indexed left to right from $-L$ to L. The equation determining δ is

$$\delta_{m,jl} = \begin{cases} 1 & \text{if } m = j \times l \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Equation (12) is employed for each R_m matrix, $m = -M, ..., M$. Once all entries are filled, the Rm function then returns the transform matrices, properly indexed as a list of arrays.

The program also contains a second function for outputting transform R_m matrices, called Rmrows. It takes the same slope and index parameters as the first Rm function, and its first step is identical to Rm's. The key difference in Rmrows comes next, in the addition of row scans. This function assumes all row scans are taken, so each R_m matrix will have $(2J+1)+(2L+1)$ rows, but the same $(2L+1)$ number of columns as before. These row scans are accounted for by simply appending a $(2L+1)\times(2L+1)$ identity matrix to the bottom of each R_m matrix. Rmrows returns a properly indexed list of $2(L+J+1)\times(2L+1)$ arrays.

3.2 Discrete Radon Transform

This program contains one function, frt, which takes two parameters, a and R. The a is the body to be transformed, and R is a list of arrays containing the transform matrices. For the function to work, the number of columns in the input array a must equal or exceed the number of transform matrices, staying consistent with the condition that $(2M+1)\leq N$. After this check, the program uses equation (1) to calculate the transformed body, $y(n)$ for $n=0,1,...,N-1$. Once complete, the function returns y.

3.3 Inverse Radon Transform

This program contains one function, ifrt, which takes two parameters: the transformed body y, and a list of arrays R of the transform matrices used in the operation.

First, the function computes the adjoint R^* matrices, and stores them in an array called "Rstar". Once computed, the function then uses equation (2) to calculate the adjoint transform of y, and stores it as z. Next, equation (4) is used to calculate H_m for $m=0,...,2M$, then Hm for $m=-2M,...,-1$ are calculated by equation (5).

The next step computes the DFTs of z and H. $\hat{z}$ is calculated by the fast fourier transform in the numpy library, which uses the Tukey-Cooley algorithm to reduce the total number of computations. In the current version of the program, $\hat{H}$ is calculated by means of a brute force fourier transform, as the numpy algorithm produced the incorrect output for the 3 dimensional array, H. In addition, though only values of $k=0,...,N/2$ should be needed to successfully complete the transform, incorrect results were obtained when using this limited band of frequencies. As a consequence, the less efficient, brute force, complete computation of $\hat{H}$ is used.

Once $\hat{z}$ and $\hat{H}$ are computed, the program uses the key lemma (6) to calculate $\hat{x}$ for invertible frequencies of k. If $\hat{H}(k)$ is noninvertible, an error message with the problematic k frequency is printed to the console, and the $\hat{x}$ in question is simply filled with a column of zeroes. This process is carried out for all values of k.

Finally, x is computed by taking the inverse fourier transform of $\hat{x}$. Like the fourier transform of z, this inversion also makes use of the numpy fast fourier transform library. Because the inverse fourier transform is normalized, $\hat{x}$ must be multiplied by a factor of N to receive the desired output.

4 Discussion and Open problems

Though this paper outlines an algorithm for calculating the forward and inverse DRTs of a given body, this algorithm has not been optimized for maximum efficiency. Open questions for maximizing this efficiency include:

- Is the 0 frequency recoverable without row scans?
- What is the smallest number of scans needed for successful inversion?
- Can the $\hat{H}$ matrices be computed directly from R and R^* matrices, without computation of the H matrices?
- Optimizing the computation of $\hat{H}$ in the current program, by successfully implementing the FFT
- Can one create a user friendly interface to select scans with unusual shapes (v's, x's, odd shapes) and output the corresponding transform R matrices?

.

5 References

1. G. Beylkin. Finite Radon Transform. IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. ASSP-35, NO. 2. Feb. 1987.
2. J. Tittlefitz.
3. J. Morrow.