THE RADON TRANSFORM, WAVELET ANALYSIS, AND APPLICATIONS

JOHN U. CARVALHO

January, 1996

for Prof. D. Lawson

ABSTRACT. The objective of this paper is to examine the Radon transform, its properties, and wavelet analysis applied to numerical analysis of the inverse Radon transform. With applications in computer science in mind.

THE RADON TRANSFORM 1.1

The Radon Transform was developed in 1917 by Johann Radon in his paper "*Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten*"(translated as "*On the Determination of Functions from their Integrals along certain Manifolds* ").

For the purposes of the definitions that follow all functions will be of the class $\mathcal{D}$ of $C^\infty(\mathbb{R}^n)$. Which is the class of functions that have compact support in $\mathbb{R}^n$ and are infinitely differentiable.

The Radon Transform in $\mathbb{R}^2$ is defined as follows see Figure 1.1.

$$\check{f}(p,\phi) = \Re f = \int_L f(x,y)\,ds \tag{1-1}$$

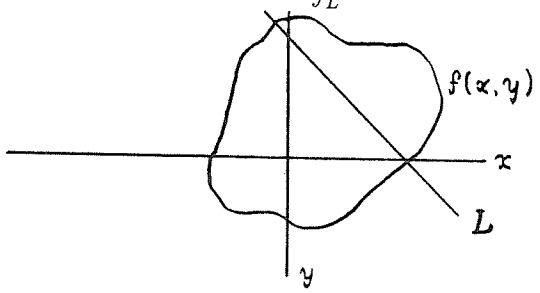

FIGURE 1.1.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

Now if the line L is given in normal form see Figure 1.2.

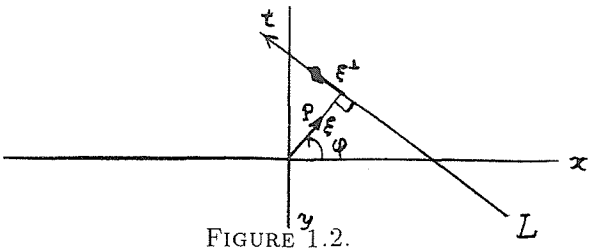

FIGURE 1.2.

$$p = x\cos\phi + y\sin\phi$$
$$\xi = (\cos\phi, \sin\phi)$$

and

$$\xi^{\perp} = (-\sin\phi, \cos\phi)$$

thus any point (x, y) on the line L can be represented as

$$(x, y) = \mathbf{x} = p\xi + t\,\xi^{\perp} \tag{1-2}$$

Also the equation for the line L can be written as

$$p = \xi \cdot \mathbf{x} \tag{1-3}$$

So eqn. 1–1 becomes

$$\Re f = \check{f}(p, \phi) = \int_{-\infty}^{\infty} f(p\,\xi + t\xi^{\perp})\, dt \tag{1-4}$$

using eqn. 1–2.
Also by using eqn. 1–3 in eqn. 1–1 , the Radon transform can be written as

$$\Re f = \check{f}(p, \xi) = \iint_{\mathbb{R}^2} f(\mathbf{x})\, \delta\,(p - \xi \cdot \mathbf{x})\; dx\, dy$$
$$\Re f = \check{f}(p, \xi) = \int f(\mathbf{x})\, \delta\,(p - \xi \cdot \mathbf{x})\; d\mathbf{x} \tag{1-5}$$

where δ is the Dirac delta functional.[1]

[1]The Dirac Delta functional has the following properities:

$$\int_{-\epsilon}^{\epsilon} f(x)\,\delta(x)\,dx = f(0)\,, \qquad \int_{\epsilon}^{\infty} f(x)\,\delta(x)\,dx = \int_{-\infty}^{-\epsilon} f(x)\,\delta(x)\,dx = 0\ ,\ \forall \epsilon \geq 0$$

The Radon Transform Extensions 1.2

In higher dimensions the extensions should be quite obvious from eqn. 1–3. Namely if we extend eqn. 1–3 to

$$p = \xi \cdot \mathbf{x} = \sum_{i=1}^{n} \xi_i \mathbf{x}_i \tag{1-6}$$

Which defines a hyperplane in $\mathbb{R}^n$ and if ξ is written in generalized spherical coordinates as follows

$$\begin{aligned}
x_1 &= r\cos\theta_1 \\
x_2 &= r\sin\theta_1\cos\theta_2 \\
x_3 &= r\sin\theta_1\sin\theta_2\cos\theta_3 \\
&\vdots \\
x_{n-2} &= r\sin\theta_1\sin\theta_2\ldots\sin\theta_{n-3}\cos\theta_{n-2} \\
x_{n-1} &= r\sin\theta_1\sin\theta_2\ldots\sin\theta_{n-2}\cos\phi \\
x_n &= r\sin\theta_1\sin\theta_2\ldots\sin\theta_{n-3}\sin\phi \qquad (1\text{-}7a) \\
r \geq 0, &\qquad 0 \leq \theta_j \leq \pi, \qquad 0 \leq \phi < 2\pi
\end{aligned}$$

with Volume element:

$$d\mathbf{x} = r^{n-1}(\sin\theta_1)^{n-2}(\sin\theta_2)^{n-3}\ldots(\sin\theta_{n-2})\,dr\,d\theta_1\ldots d\theta_{n-2}\,d\phi \tag{1-7b}$$

and with Surface element:

$$d\omega = r^{n-1}(\sin\theta_1)^{n-2}(\sin\theta_2)^{n-3}\ldots(\sin\theta_{n-2})\,d\theta_1\ldots d\theta_{n-2}\,d\phi \tag{1-7c}$$

Thus eqn. 1–5 still is valid with the exception that the integration is now over $\mathbb{R}^n$.

$$\Re f = \check{f}(p,\xi) = \int_{\mathbb{R}^n} f(\mathbf{x})\,\delta\left(p - \xi\cdot\mathbf{x}\right)\,d\mathbf{x} \tag{1-8}$$

The extension of the Radon Transform definition to Manifolds is obvious by integrating over submanifolds as opposed to lines and planes.

$$\Re f = \check{f}(p,\xi) = \int_{H_{p,\xi}} f(\mathbf{x})\,|\nabla_{\mathbf{x}}\,u\,(\mathbf{x},\xi)\,|\,ds \tag{1-9a}$$

where

$$H_{p,\xi} = \{\, x \in \Omega : u(\mathbf{x},\xi) = p \,\} \tag{1-9b}$$

Now $H_{p,\xi}$ can be identified with a *Hyperplane* or Submanifold and Ω as the Manifold. Also $u(\mathbf{x},\xi) = p$ can be identifed with $\mathbf{x}\cdot\xi = p$ which is eqn. 1–6.

The Radon Transform Properties 1.3

(1) Homogeneous of degree -1 and symmetric.

$$\begin{aligned}\check{f}(sp, s\xi) &= \int f(\mathbf{x})\ \delta\left(sp - s\xi \cdot \mathbf{x}\right)\ d\mathbf{x} \\ &= |s|^{-1} \int f(\mathbf{x})\ \delta\left(p - \xi \cdot \mathbf{x}\right)\ d\mathbf{x} \\ \check{f}(sp, s\xi) &= |s|^{-1} \check{f}(p, \xi) \end{aligned} \tag{1-10}$$

with $s = -1$ we have symmetry

$$\check{f}(-p, -\xi) = \check{f}(p, \xi) \tag{1-11}$$

(2) Linearity.

$$\Re\{c_1\, f + c_2\, g\} = c_1\, \check{f} + c_2\, \check{g} \tag{1-12}$$

(3) Transform of a Linear Transformation.

let A be a non–singular matrix

$$\begin{aligned}\Re\{\, f(A\mathbf{x})\,\} &= \int f(A\mathbf{x})\ \delta\left(p - \xi \cdot \mathbf{x}\right)\ d\mathbf{x} \\ &= \frac{1}{|det\, A\,|} \int f(\mathbf{y})\ \delta\left(p - \xi \cdot A^{-1}\mathbf{y}\right)\ d\mathbf{y} \\ &= \frac{\check{f}(p, (A^{-1})^{\mathbf{T}}\, \xi)}{|det\, A\,|} \end{aligned} \tag{1-13}$$

(4) Shift Property.

$$\Re\{f(\mathbf{x} - \mathbf{a})\} = \check{f}(p - \xi \cdot \mathbf{a}, \xi) \tag{1-14}$$

(5) Transform of Derivatives.

$$\Re\{\frac{\partial}{\partial \mathbf{x}_k} f\} = \xi_k \frac{\partial}{\partial p}\, \check{f}(p, \xi) \tag{1-15}$$

(6) Derivatives of the Transform.

$$\left(\frac{\partial}{\partial \xi_k}\right)\ \check{f} = -\frac{\partial}{\partial p}\, \Re\{\mathbf{x}_k\ f(\mathbf{x})\} \tag{1-16}$$

(7) Transform of a Convolution.

$$f(\mathbf{x}) = g * h = h * g = \int g(\mathbf{y})\, h(\mathbf{x} - \mathbf{y})\, d\mathbf{y} = \int h(\mathbf{y})\, g(\mathbf{x} - \mathbf{y})\, d\mathbf{y}$$

then

$$\Re f = \check{f}(p, \xi) = \check{g} * \check{h} \tag{1-17}$$

This last property is a major difference between the Radon Transform versus its counterpart the Fourier Transform.

RADON TRANSFORM INVERSION 1.4

• *Odd Dimension.*

Starting with the following integral

$$\int f(\mathbf{y})|\,\xi\cdot(\mathbf{y}-\mathbf{x})|\,d\mathbf{y} = \int d\mathbf{y}\, f(\mathbf{y}) \int_{-\infty}^{\infty} d\,p\ |p|\ \delta(p-\xi\cdot(\mathbf{y}-\mathbf{x}))$$
$$= \int_{-\infty}^{\infty} d\,p\ |p| \int d\,\mathbf{z}\, f(\mathbf{x}+\mathbf{z})\ \delta(p-\xi\cdot\mathbf{z})$$

where $\mathbf{y}=\mathbf{x}+\mathbf{z}$.

$$= \int_{-\infty}^{\infty} d\,p\,|p|\,\check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-18}$$

Thus by integrating eqn. 1–18. over $|\xi|=1$

$$\int_{|\xi|=1} d\xi \int d\mathbf{y}\, f(\mathbf{y})|\,\xi\cdot(\mathbf{y}-\mathbf{x})| = \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} d\,p\,|p|\,\check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-19}$$

Now, Hilbert and Courant have shown that for n odd $n \geq 3$.

$$4(2\pi)^{n-1}(-1)^{\frac{n-1}{2}}\, f(\mathbf{x}) = \Delta_{\mathbf{x}}^{\frac{n+1}{2}} \int_{|\xi|=1} d\xi \int d\mathbf{y}\, f(\mathbf{y})\,|\xi\cdot(\mathbf{y}-\mathbf{x})| \tag{1-20}$$

Where $\Delta_{\mathbf{x}}$ is the Laplacian operator. Thus

$$4(2\pi)^{n-1}(-1)^{\frac{n-1}{2}}\, f(\mathbf{x}) = \Delta_{\mathbf{x}}^{\frac{n+1}{2}} \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp\,|p|\,\check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-21}$$

Now, by evaluating

$$\Delta_{\mathbf{x}} \int_{-\infty}^{\infty} d\,p\,|p|\,\check{f}(p+\xi\cdot\mathbf{x},\xi) = \Delta_{\mathbf{x}} \int_{\xi\cdot\mathbf{x}}^{\infty} (t-\xi\cdot\mathbf{x})\check{f}(t,\xi)\,dt$$
$$- \Delta_{\mathbf{x}} \int_{-\infty}^{\xi\cdot\mathbf{x}} (t-\xi\cdot\mathbf{x})\check{f}(t,\xi)\,dt$$

where $p=t-\xi\cdot\mathbf{x}$. So by applying Leibnitz rule gives

$$= 2\,\xi\cdot\xi\,\check{f}(\xi\cdot\mathbf{x},\xi) \tag{1-22}$$

but $\xi\cdot\xi=1$ (since this will be integrated over $|\xi|=1$). Thus eqn. 1–21 becomes

$$4(2\pi)^{n-1}(-1)^{\frac{n-1}{2}}\, f(\mathbf{x}) = 2\Delta_{\mathbf{x}}^{\frac{n-1}{2}} \int_{|\xi|=1} d\xi\ \check{f}(\xi\cdot\mathbf{x},\xi)$$
$$f(\mathbf{x}) = \frac{1}{2\,(2\pi\imath)^{n-1}} \int_{|\xi|=1} d\xi\,(\tfrac{\partial}{\partial p})^{n-1}\check{f}(\xi\cdot\mathbf{x},\xi) \tag{1-23}$$

• *Even Dimension.*

Starting with the following integral

$$\int f(\mathbf{y}) \log|\xi\cdot(\mathbf{y}-\mathbf{x})|\, d\mathbf{y} = \int d\mathbf{y} f(\mathbf{y}) \int_{-\infty}^{\infty} dp\ \log|p|\ \delta(p-\xi\cdot(\mathbf{y}-\mathbf{x}))$$
$$= \int_{-\infty}^{\infty} dp\, \log|p|\, \check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-24}$$

Thus by integrating eqn. 1–24 over $|\xi|=1$.

$$\int_{|\xi|=1} d\xi \int d\mathbf{y}\, f(\mathbf{y}) \log|\xi\cdot(\mathbf{y}-\mathbf{x})| = \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp\, \log|p|\, \check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-25}$$

Now, Hilbert and Courant have shown that for n even $n \geq 2$.

$$(2\pi)^n(-1)^{\frac{n-2}{2}} f(\mathbf{x}) = \Delta_{\mathbf{x}}^{\frac{n}{2}} \int_{|\xi|=1} d\xi \int d\mathbf{y}\, f(\mathbf{y}) \log|\xi\cdot(\mathbf{y}-\mathbf{x})| \tag{1-26}$$

Where $\Delta_{\mathbf{x}}$ is the Laplacian operator. Thus

$$(2\pi)^n(-1)^{\frac{n-2}{2}} f(\mathbf{x}) = \Delta_{\mathbf{x}}^{\frac{n}{2}} \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp\, \log|p|\, \check{f}(p+\xi\cdot\mathbf{x},\xi) \tag{1-27}$$

Now, by evaluating the Laplacian operator first,

$$\Delta_{\mathbf{x}} \int_{-\infty}^{\infty} dp\, \log|p|\, \check{f}(p+\xi\cdot\mathbf{x},\xi) = \int_{-\infty}^{\infty} dp\, |\xi|^2\, \log|p|\, \left[\tfrac{\partial^2}{\partial p^2}\check{f}(p,\xi)\right]\Big|^{p=p+\xi\cdot\mathbf{x}} \tag{1-28}$$

by making the change of variables $p = t - \xi\cdot\mathbf{x}$,and noting that $|\xi|^2 = 1$ since this will be integrated over $|\xi| = 1$, and integrating by parts we have

$$= \int_{-\infty}^{\infty} \log|t-\xi\cdot\mathbf{x}|\check{f}_{tt}(t,\xi)\, dt$$
$$= -\int_{-\infty}^{\infty} dp \frac{\check{f}_p(p,\xi)}{p-\xi\cdot\mathbf{x}} \tag{1-29}$$

This all leads to the following inversion result

$$f(\mathbf{x}) = \frac{1}{(2\imath\pi)^n}\Delta_{\mathbf{x}}^{\frac{n-2}{2}} \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp \frac{\check{f}_p(p,\xi)}{p-\xi\cdot\mathbf{x}}$$

where by applying $\Delta_{\mathbf{x}}$ and integrating by parts twice and using $|\xi| = 1$ gives

$$f(\mathbf{x}) = \frac{1}{(2\imath\pi)^n} \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp \frac{(\frac{\partial}{\partial p})^{n-1}\check{f}(p,\xi)}{p-\xi\cdot\mathbf{x}} \tag{1-30}$$

THE ADJOINT OF THE RADON TRANSFORM 1.5

From the definition of an adjoint we must have

$$\langle f, \Re^* g \rangle_{\mathbb{R}^n} = \langle \Re f, g \rangle_{\mathbb{R}\times\mathbb{S}^{n-1}}$$

Now by starting with the right side $\langle \Re f, g \rangle_{\mathbb{R}\times\mathbb{S}^{n-1}}$

$$\begin{aligned}
\langle \Re f, g \rangle_{\mathbb{R}\times\mathbb{S}^{n-1}} &= \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp\, \check{f}(p,\xi)\, g(p,\xi) \\
&= \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp \int d\mathbf{x} f(\mathbf{x})\, \delta(p - \xi\cdot\mathbf{x})\, g(p,\xi) \\
&= \int d\mathbf{x}\, f(\mathbf{x}) \int_{|\xi|=1} d\xi \int_{-\infty}^{\infty} dp\, g(p,\xi)\, \delta(p - \xi\cdot\mathbf{x}) \\
&= \int d\mathbf{x}\, f(\mathbf{x}) \int_{|\xi|=1} d\xi g(\xi\cdot\mathbf{x},\xi) \qquad (1\text{-}31) \\
&= \langle f, \Re^* g \rangle_{\mathbb{R}^n}
\end{aligned}$$

Thus we have an explicit definition of the adjoint from eqn 1–31.

$$\Re^*\left(g(p,\xi)\right) = \int_{|\xi|=1} d\xi\, g(\xi\cdot\mathbf{x},\xi) \qquad (1\text{-}32)$$

Now from the previous inversion results, these results can be rewritten in the following operator notation.

$$f = \Re^* \Lambda \Re f$$

where the operator Λ is defined as follows

$$\Lambda g(t) = \left\{ \begin{array}{ll} \frac{1}{2}(2\pi\imath)^{1-n}\ (\frac{\partial}{\partial p})^{n-1}\, g(p)\Big|^{p=t} & \text{for } n \text{ odd} \\ \frac{1}{2\imath}(2\pi\imath)^{1-n}\ \left[\mathcal{H}\left((\frac{\partial}{\partial p})^{n-1}\, g(p)\right)\right]\Big|^{p=t} & \text{for } n \text{ even} \end{array} \right\} \qquad (1\text{-}33)$$

where $\mathcal{H}$ is the Hilbert transform defined as

$$\mathcal{H}\, g(t) = \frac{1}{\pi} \int_{-\infty}^{\infty} dp\, \frac{g(p)}{p-t} \qquad (1\text{-}34)$$

Now using operator notation we have the definitions for both $\Re$ inverse and $\Re^*$ inverse, given by:

$$\Re^* \Lambda \Re = I \qquad (1\text{-}35)$$

$$\Lambda \Re \Re^* = I \qquad (1\text{-}36)$$

Fourier Inversion Methods 1.6

The Fourier transform and its inverse are defined as follows

The Fourier transform:

$$\mathcal{F}f = \tilde{f}(\mathbf{k}) = \int f(\mathbf{x}) e^{-2\pi\imath\,\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{x} \tag{1-37}$$

and its inverse:

$$\mathcal{F}^{-1}\tilde{f} = f(\mathbf{x}) = \int \tilde{f}(\mathbf{k}) e^{2\pi\imath\,\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{k} \tag{1-38}$$

Now

$$\begin{aligned}\tilde{f}(\mathbf{k}) &= \int f(\mathbf{x}) e^{-2\pi\imath\,\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{x} \\ &= \int_{-\infty}^{\infty} dt \int d\mathbf{x} f(\mathbf{x})\, e^{-2\pi\imath t}\, \delta(t - \mathbf{k}\cdot\mathbf{x})\end{aligned}$$

by letting $\mathbf{k} = s\xi$, and $t = sp$

$$\begin{aligned}\tilde{f}(s\xi) &= |s| \int_{-\infty}^{\infty} dp \int d\mathbf{x} f(\mathbf{x})\, e^{-2\pi\imath sp}\, \delta(sp - s\xi\cdot\mathbf{x}) \\ &= \int_{-\infty}^{\infty} dp\, e^{-2\pi\imath sp} \int d\mathbf{x} f(\mathbf{x})\, \delta(p - \xi\cdot\mathbf{x}) \\ &= \int_{-\infty}^{\infty} dp\, e^{-2\pi\imath sp}\, \check{f}(p,\xi)\end{aligned} \tag{1-39}$$

Thus, from eqn. 1–39 we have a relationship between the Fourier transform and the Radon transform.

$$\tilde{f}(s\xi) = \mathcal{F}(\check{f}(p,\xi)) \tag{1-40}$$

(note this is a one dimensional Fourier transform in the p variable.)

$$f = \mathcal{F}_n^{-1}\, \mathcal{F}_1\, \check{f} \tag{1-41}$$

This last inversion formulae in eqn. 1–41 is known as the Fourier central slice theorem which can be depicted by the following commuttator diagram.

$$\begin{array}{ccc} f(x,y) & \xrightarrow{\Re} & \check{f}(p,\phi) \\ {\scriptstyle \mathcal{F}_2}\downarrow & & {\scriptstyle \mathcal{F}_1}\downarrow \\ \tilde{f}(\mathbf{k}_x,\mathbf{k}_y) & = & \tilde{\check{f}}(k,\phi) \end{array}$$

where

$$\begin{aligned}\tilde{f}(\mathbf{k}_x,\mathbf{k}_y) &= \tilde{\check{f}}(k,\phi) \\ k &= (\mathbf{k}_x^2 + \mathbf{k}_y^2)^{\frac{1}{2}} \\ \phi &= \tan^{-1}(\mathbf{k}_y\,/\mathbf{k}_x)\end{aligned}$$

Thus the one dimentional Fourier transform of the Radon transform in the radial variable p is the n dimentional Fourier transform of the original function.

BACKPROJECTION(ADJOINT), FOURIER ANALYSIS AND INVERSION 1.7

The adjoint operator $\Re^*$ is sometimes refered to as the backprojection operator. The explicit use of the adjoint operation leads to an alternative methods of inversion which utilizes the previous Fourier analysis in the inversion process.

$$\begin{aligned}\Re^* \check{f}(p,\phi) &= \int_0^{2\pi} \check{f}(\xi \cdot \mathbf{x}, \xi)\, d\phi \\ &= \int_0^{2\pi} \check{f}(x\cos\phi + y\sin\phi, \phi)\, d\phi\end{aligned}$$

Now by using eqn. 1–41,

$$\begin{aligned}\check{f} &= \mathcal{F}_1^{-1}\, \mathcal{F}_2\, f \\ \Re^* \check{f} &= \Re^* \mathcal{F}_1^{-1}\, \mathcal{F}_2\, f \\ &= \Re^* \mathcal{F}_1^{-1}\ \tilde{f}(k,\phi)\end{aligned} \tag{1-42}$$

where once again

$$\begin{aligned}k &= (\mathbf{k}_x^2 + \mathbf{k}_y^2)^{\frac{1}{2}} \\ \phi &= \tan^{-1}(\mathbf{k}_y\, /\mathbf{k}_x)\end{aligned}$$

$$\begin{aligned}\Re^* \mathcal{F}_1^{-1}\ \tilde{f}(k,\phi) &= \Re^* \int_{-\infty}^{\infty} dk\, \tilde{f}(k,\phi)\, e^{2\pi\imath kp} \\ &= \int_0^{2\pi} d\phi \int_{-\infty}^{\infty} dk\, \tilde{f}(k,\phi)\, e^{2\pi\imath kr\cos(\theta-\phi)} \\ &= \int_0^{2\pi} d\phi \int_{-\infty}^{\infty} k^{-1}\, \tilde{f}(k,\phi)\, e^{2\pi\imath kr\cos(\theta-\phi)}\, k dk \\ &= \mathcal{F}_2^{-1}\ \left(k^{-1}\, \tilde{f}\right)\end{aligned} \tag{1-43}$$

From eqn. 1–35 we have

$$\begin{aligned}f &= \Re^* \Lambda \check{f} \\ &= \Re^* \mathcal{F}_1^{-1} \mathcal{F}_1 \Lambda \check{f} \\ &= \tfrac{1}{4\pi^2}\, \Re^* \mathcal{F}_1^{-1} \mathcal{F}_1\, \tfrac{\partial}{\partial t} \left\{ \tfrac{1}{t} * \check{f}(t,\phi) \right\} \\ &= \tfrac{1}{4\pi^2}\, \Re^* \mathcal{F}_1^{-1} \left\{ (2\pi\imath k)\ \mathcal{F}_1\ \left(\tfrac{1}{t}\right)\ \mathcal{F}_1\, \check{f}(t,\phi) \right\} \\ &= \tfrac{1}{4\pi^2}\, \Re^* \mathcal{F}_1^{-1} \left\{ (2\pi\imath k)\ (-\pi\imath\, sgn(k))\ \tilde{\check{f}}(t,\phi) \right\} \\ &= \tfrac{1}{2}\, \Re^* \mathcal{F}_1^{-1} \left\{ |k|\ \tilde{\check{f}}(t,\phi) \right\}\end{aligned} \tag{1-44}$$

Which gives us yet another inversion method. Also by combining these two results from eqn. 1–43 and 1–44 gives

$$f = \mathcal{F}_2^{-1} \left\{ |k| \mathcal{F}_2\, \Re^*\, \check{f} \right\} \tag{1-45}$$

Both eqns. 1–44 and 1–45 can be depicted again as commutator diagrams just as was the Fourier central slice theorem (eqn. 1–41). The commutator diagram for the filtered backprojection technique which depicts eqn 1–44 is

$$\tfrac{1}{2}\,\Re^*\,\mathcal{F}_1^{-1}\left\{|k|\;\tilde{\check{f}}(t,\phi)\right\}$$

$$\begin{array}{ccc} f(x,y) & \xrightarrow{\;\Re\;} & \check{f}(p,\phi) \\ {\scriptstyle\frac{1}{2}\Re^*}\uparrow & & \downarrow{\scriptstyle\mathcal{F}_1} \\ f^*(s,\phi) & \xleftarrow[\mathcal{F}_1^{-1}|k|\cdot]{} & \tilde{\check{f}}(k,\phi) \end{array}$$

Also, the commutator diagram which corresponds to eqn 1–45. the backprojection filter technique is

$$f = \mathcal{F}_2^{-1}\left\{|k|\mathcal{F}_2\,\Re^*\,\check{f}\right\}$$

$$\begin{array}{ccc} f(x,y) & \xrightarrow{\;\Re\;} & \check{f}(p,\phi) \\ {\scriptstyle\mathcal{F}_2}\downarrow & & \downarrow{\scriptstyle\Re^*} \\ \tilde{f}(\mathbf{k}_x,\mathbf{k}_y) & \xleftarrow[|k|\mathcal{F}_2]{} & \tilde{\check{f}}^* \end{array}$$

Discrete Inversion and Sampling 1.8

Theorem 1). *Any object (function $f(x,y)$) can be determined by an infinite set of Radon Transforms $\check{f}(p,\xi)$ which are defined on a set $A \ni A = \mathbb{R} \times \mathbb{S}^{n-1} - N$ where N is a set of Null measure.*

Theorem 2). $\exists f \in C_c^\infty(\mathbb{R}^n)$, $f \neq 0 \ni \check{f}(p,\omega_i) = 0, \ \forall i.$

Hence there is a non–uniqueness problem associated with the inversion problem if given only finite information.

Now from Fourier analysis there is the Shannon sampling theorem which tells how to recover a function from its sampled values.

Theorem (Shannon Sampling Theorem) 3). *if $x(t)$ is bandlimited (that is $\tilde{x}(\nu) = 0 \quad \forall |\nu| > \nu_b$) then $x(t)$ can be recovered uniquely from its sampled values, if the sample spaces are not separated by more than $\Delta t = \dfrac{1}{\nu_b}$ (known as the Nyquist sampling rate). Explicitly $x(t)$ can be recovered by the following.*

$$x(t) = \sum_{k=-\infty}^{\infty} x\left(\frac{k}{2\nu_b}\right) \frac{\sin\left[2\pi\nu_b\left(t - \frac{k}{2\nu_b}\right)\right]}{2\pi\nu_b\left(t - \frac{k}{2\nu_b}\right)}$$

Discrete Radon Transform Inversion 1.9

It is the objective of this section to construct discrete versions of the Fourier central slice theorem (eqn. 1–41) and the filtered backprojection reconstruction technique (eqn. 1–44). Even though there is no gaurentee that the results will be accurate as a result of Theorem 2. In order to do this we must employ the Shannon Sampling Theorem for discrete bandlimited frequencies so that the original function may be recovered.

Starting with the filtered backprojection reconstruction technique (eqn. 1–44) which follows with the next three equations

$$\tilde{\check{f}}(q,\phi) = \int_{-\infty}^{\infty} \check{f}(p,\phi)\, e^{-2\pi\imath qp}\, dp \tag{1-46}$$

From eqn. 1–44 the original function f can be recovered by

$$f(x,y) = \tfrac{1}{2}\int_0^{2\pi} Q(x\cos\phi + y\sin\phi)\, d\phi \tag{1-47}$$

where

$$Q(s,\phi) = \int_{-\infty}^{\infty} \tilde{\check{f}}(q,\phi)\,|q|\, e^{2\pi\imath sq}\, dq \tag{1-48}$$

Now q has dimensions of a reciprocal length and thus can be seen as a frequency. If the last integral in eqn. 1–48 is bandlimited with a bandwidth of ω_b or if eqn. 1–48 has negligable contributions for $|q| > \omega_b$; then Shannon's sampling theorem can be applied for $|q| > \omega_b$.

$$\check{f}(p,\phi) = \sum_{l=-\infty}^{\infty} \check{f}\left(\frac{l}{2\omega_b}\right) \frac{\sin\left[2\pi\omega_b\left(p - \frac{l}{2\omega_b}\right)\right]}{2\pi\omega_b\left(t - \frac{l}{2\omega_b}\right)} \tag{1-49}$$

By taking the Fourier transform of eqn. 1–49 gives

$$\tilde{\check{f}}(q,\phi) = \frac{1}{2\omega_b} \sum_{l=-\infty}^{\infty} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi i q \frac{1}{2\omega_b}} \chi_{[-1/2,1/2]}\left(\frac{q}{2\omega_b}\right) \tag{1-50}$$

If the assumption is made that $N+1$ sampled values of $\check{f}(\cdot,\phi)$ is accurate for a representation, then eqn. 1–50 can be reduced to.

$$\tilde{\check{f}}(q,\phi) = \frac{1}{2\omega_b} \sum_{l=-N/2}^{N/2} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi i q \frac{1}{2\omega_b}} \chi_{[-1/2,1/2]}\left(\frac{q}{2\omega_b}\right) \tag{1-51}$$

Clearly $\tilde{\check{f}}(q,\phi)$ is zero outside the interval $q \in [-\omega_b, \omega_b]$. Also if $\tilde{\check{f}}(q,\phi)$ is evaluated at the points

$$q = m\frac{2\omega_b}{N} \text{ where } m = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2}.$$

then

$$\tilde{\check{f}}(m\frac{2\omega_b}{N},\phi) = \frac{1}{2\omega_b} \sum_{l=-N/2}^{N/2} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi i (ml/N)} \tag{1-52}$$

Which is a Discrete Fourier Transform (DFT). So by continuing in evaluating, proceeding with eqn. 1–48.

$$\begin{aligned} Q(s,\phi) &= \int_{-\infty}^{\infty} \tilde{\check{f}}(q,\phi)\,|q|\, e^{2\pi i s q}\, dq \\ &\simeq \frac{2\omega_b}{N} \sum_{m=-N/2}^{N/2} \tilde{\check{f}}\left(m\frac{2\omega_b}{N},\phi\right) |m\frac{2\omega_b}{N}|\, e^{2\pi i s m(2\omega_b/N)} \end{aligned} \tag{1-53}$$

by evaluating this at the sampled $p = \frac{k}{2\omega_b}$ gives

$$Q(\frac{k}{2\omega_b},\phi) \simeq \frac{2\omega_b}{N} \sum_{m=-N/2}^{N/2} \tilde{\check{f}}\left(m\frac{2\omega_b}{N},\phi\right) |m\frac{2\omega_b}{N}|\, e^{2\pi i (mk/N)} \tag{1-54}$$

The final computation required to recover $f(x,y)$ is

$$f(x,y) \simeq \frac{\pi}{K} \sum_{l=1}^{K} Q(x\cos\phi_l + y\sin\phi_l) \tag{1-55}$$

where the angles ϕ_l with $l = 1, 2, \ldots, K$ are the angles used to sample the Radon Transform $\check{f}(p,\phi)$. Thus the full discrete version of the filtered backprojection reconstruction technique is given by eqns. 1–54 and 1–55.

Continuing with the Fourier central slice theorem given by eqn. 1–41 we have the following

$$\tilde{\check{f}}(q,\phi) = \int_{-\infty}^{\infty} \check{f}(p,\phi)\, e^{-2\pi\imath qp}\, dp \tag{1-56}$$

Which gives the following recovery technique

$$f(x,y) = \mathcal{F}_2^{-1}\tilde{\check{f}}(q\cos\phi, q\sin\phi) \tag{1-57}$$

Now if eqn 1–56 is bandlimited then we can apply Shannon's Sampling Theorem to $\check{f}(p,\phi)$. Thus by repeating the previous arguments used in developing the back-projection technique, the variable q has dimentions of reciprocal length and can be seen as a frequency. If the frequencies q are bandlimited by ω_b or if has neglibable contributions for frequencies $|q| > \omega_b$; then Shannon's sampling theorem can be applied for $|q| > \omega_b$.

$$\check{f}(p,\phi) = \sum_{l=-\infty}^{\infty} \check{f}\left(\frac{l}{2\omega_b}\right) \frac{\sin\left[2\pi\omega_b\left(p - \frac{l}{2\omega_b}\right)\right]}{2\pi\omega_b\left(t - \frac{l}{2\omega_b}\right)} \tag{1-58}$$

So, by taking the Fourier transform of eqn 1–58, gives

$$\tilde{\check{f}}(q,\phi) = \frac{1}{2\omega_b} \sum_{l=-\infty}^{\infty} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi\imath q\frac{1}{2\omega_b}}\, \chi_{[-1/2,1/2]}\left(\frac{q}{2\omega_b}\right) \tag{1-59}$$

Now again if the assumption is made that $N+1$ sampled values of $\check{f}(\cdot,\phi)$ is accurate for a representation, then eqn. 1–59 can be reduced to.

$$\tilde{\check{f}}(q,\phi) = \frac{1}{2\omega_b} \sum_{l=-N/2}^{N/2} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi\imath q\frac{1}{2\omega_b}}\, \chi_{[-1/2,1/2]}\left(\frac{q}{2\omega_b}\right) \tag{1-60}$$

Clearly $\tilde{\check{f}}(q,\phi)$ is zero outside the interval $q \in [-\omega_b, \omega_b]$. Also if $\tilde{\check{f}}(q,\phi)$ is evaluated at the points

$$q = m\frac{2\omega_b}{N} \text{ where } m = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2}.$$

then

$$\tilde{\check{f}}(m\frac{2\omega_b}{N}, \phi) = \frac{1}{2\omega_b} \sum_{l=-N/2}^{N/2} \check{f}\left(\frac{l}{2\omega_b}\right) e^{-2\pi\imath(ml/N)} \tag{1-61}$$

Which is a Discrete Fourier Transform (DFT). So by continuing in evaluating, proceeding with eqn. 1–57 which uses a polor to rectangular co–ordinate conversion, thus 1–61, must undergo a polar to rectangular co–ordinate conversion. Thus some form of interpolation must be performed if the polar co-ordinate points do not coincide with the rectangular co-ordinate system. This can be accomplished by using a nearest neighbour or a weighted interpolation scheme to map polar points (q,ϕ) to $(q\cos\phi, q\sin\phi)$. Continuing once this co-ordinate transformation is completed and by following similar logic, the original function $f(x,y)$ can be recovered by applying a 2–dimentional discrete inverse fast Fourier transform.

Wavelet Analysis 2.0

Wavelet analysis is similar to Fourier analysis in that each seeks to find a orthonormal convergent series in the Hilbert space $\mathcal{H} = L^2(\Omega)$ that converges in the $L^2(\Omega)$ norm. Where $\mathcal{H} = L^2(\Omega) = \{f(x) \mid \int_\Omega |f(x)|^2\, dx < \infty\}$. Their differences however can be observed from the following example in $L^2([0,1])$. Consider a function $f(x) \in \mathcal{H} = L^2([0,1])$, it has Fourier series

$$f(x) = b_0 + \sum_1^\infty (b_k \cos 2\pi kx + a_k \sin 2\pi kx)$$

Also the same function has a Haar series (Wavelet series)

$$f(x) = c_0 + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j-1} c_{jk}\, \psi(2^j x - k)$$

where $\psi(x)$ is defined as

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \le x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le c < 1 \\ 0 & \text{otherwise.} \end{cases}$$

The whole basic principle behind wavelet analysis is the Multiresolution Analysis for constructing the wavelet basis functions ($\psi(x)$ in the previous example).

Multiresolution Analysis. *A multiresolution analysis* $\cdots \subseteq V_{-1} \subseteq V_0 \subseteq V_1 \subseteq \cdots$ *with a scaling function* φ *is an increasing sequence of subspaces of* $L^2(\mathbb{R})$ *such that:*

(1) *(density)* $\bigcup_j V_j$ *is dense in* $L^2(\mathbb{R})$
(2) *(separation)* $\bigcap_j V_j = \{0\}$
(3) *(scaling)* $f(x) \in V_j \iff f(2^j x) \in V_0$
(4) *(orthonormality)* $\{\varphi(x-k)\}_{k\in\mathbb{Z}}$ *is an orthonormal basis for* V_0

From part 3) and 4) of the Multiresolution Analysis it follows that the set $\{2^{j/2}\varphi(2^j x - k)\}_{k\in\mathbb{Z}}$ is an orthonormal basis for V_j.

The construction of the wavelet basis then proceeds as follows:
Starting with the basis set $\{\varphi(x-k)\}_{k\in\mathbb{Z}}$ which is an orthonormal basis for V_0. Hence

$$V_0 = \overline{span}\,(\{\varphi(x-k)\}_{k\in\mathbb{Z}}) \tag{2-1}$$

Now, in order to construct a basis for V_1, look at the orthogonal complement of V_0 in V_1, that is

$$V_1 = V_0 \oplus W_0 \tag{2-2}$$

In order to do this look at a basis generator of the following form

$$\psi(x) = \sum_{k\in\mathbb{Z}} c_k\, \varphi(2x - k) \tag{2-3}$$

This is known as the dilation equation. Once 2-3 is solved and ψ found then continue in a similar fashion by repeating the process where

$$V_{j+1} = V_j \oplus W_j = V_{j-1} \oplus W_{j-1} \oplus W_j \cdots \tag{2-4}$$

This discussion is, however incomplete. Since there is no explicit means given for solving the dilation equation(eqn. 2–3). Now by rewriting the dilation equation (using φ as opposed to ψ as in the Multiresolution Analysis) and taking its Fourier transform we have

$$\varphi(x) = \sum_{k\in\mathbb{Z}} c_k\, \varphi(2x-k)$$

$$\tilde{\varphi}(\xi) = \frac{1}{2}\sum_{k\in\mathbb{Z}} c_k\, e^{\imath k\xi/2}\tilde{\varphi}(\frac{\xi}{2}) \tag{2-5}$$

$$= P(\frac{\xi}{2})\,\tilde{\varphi}(\frac{\xi}{2}) \tag{2-6}$$

where

$$P(\xi) = \frac{1}{2}\sum_{k\in\mathbb{Z}} c_k\, e^{\imath k\xi} \tag{2-7}$$

called the *symbol.*
When $\xi = 0$ then $P(0) = 1$ or $\sum c_k = 2$, also for $\xi = 0$ the solution φ must statisfy $\tilde{\varphi}(0) = \int \varphi(x)\,dx = 1$. However what also can be said about φ is that

$$\tilde{\varphi}(\xi) = P(\frac{\xi}{2})\,\tilde{\varphi}(\frac{\xi}{2}) = P(\frac{\xi}{2})\,P(\frac{\xi}{4})\,\tilde{\varphi}(\frac{\xi}{4}) = \cdots = \prod_{j=1}^{\infty} P(\frac{\xi}{2^j}) \tag{2-8}$$

These contructions lead to many different wavelet basises such as the Haar wavelets, Poisson wavelets, Bessel wavelets, etc.. However, as a strong word of **WARNING** there is no guarantee about the continuity, differentiablity, nor whether the expansion makes sense in L^1 nor in L^p. As an example the Weierstrass class of functions defined by $\sum a^n \cos(b^n x)$ are continuous and nowhere differentiable and can be built up from multiple scales. As such, wavelets are used in the analysis of these functions and other self similar objects such as fractals.

There still is one thing left to do. That is calculating the Fourier and Wavelet coefficients in the their respective function expansions. Starting with Fourier analysis on the interval $[0,1]$.

$$f(x) = \sum_{k\in\mathbb{Z}} \langle f(x)|\, e_k(x)\rangle\; e_k(x) \tag{2-9}$$

where

$$e_k(x) = e^{2\pi\imath kx}$$

and

$$\langle f(x)|\, g(x)\rangle = \int_{-\infty}^{\infty} f(x)\,\bar{g}(x)\,dx$$

thus eqn. 2–9 can be expanded as follows

$$
\begin{aligned}
f(x) &= \sum_{k\in\mathbb{Z}} \langle f(x)|\, e_k(x)\rangle\; e_k(x) \\
&= \sum_{k\in\mathbb{Z}} \left(\int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx \right) e_k(x) \\
&= \sum_{k\in\mathbb{Z}} \tilde{f}(k)\, e_k(x)
\end{aligned}
\tag{2-10}
$$

As for wavelet series, a similar procedure as above will be employed. The wavelet transform can be defined as follows

$$
[\mathcal{W}_\varphi f]\,(b,a) = \int_{-\infty}^{\infty} dx\, \frac{1}{a}\, \bar{\varphi}\left(\frac{x-b}{a}\right)\, f(x) \tag{2-11}
$$

As such the function can be recovered similarly as it was in eqn. 2–10 by

$$
f(x) = \sum_{j=0}^{\infty} \sum_{k=0}^{2^j-1} \left\{ [\mathcal{W}_\varphi f]\,(k2^j, 2^j) \right\}\; \varphi(2^j x - k) \tag{2-12}
$$

One major advantage of useing wavelets is that the convergence is of exponential rate in L^2 and the wavelet series can be truncated with minimal loss in accuracy. In contrast Fourier series require many terms due to cancelation. Another major advantage of wavelet series is that numerically their discrete versions can be implemented very efficiently. This is due to the *pyramid scheme* created by Burt, Adelson, Mallat and others; a discrete Fast Wavelet Transform (FWT). This algorithm is has a computational cost of $2n$. Where as Fast Fourier Transforms (FFT) has a cost of $\frac{1}{2}\, n \log_2 n$. Wavelets have been very successful in such areas as data compresion, namely the compression of fingerprint data by the FBI. Also, wavelets have had some successes in signal analysis, acoustics, high resolution video, etc..

Wavelets Applied to Radon Transform Inversion 2.1

Just as before with Fourier analysis wavelet techniques can be applied to the inverse Radon transform problem. Also as before there are wavelet techniques which mirror the Fourier central slice theorem and there are wavelet techniques which mirror the filter backprojection techniques. However the latter of the two is of greater interest. Primarily, in that wavelat methods which are similar to backprojection have had some success in limited angle tomography. Which is the ability to reconstruct local areas of the function without scanning the entire function and being able to do so by using scans passing through a limited range of angles. This is of great importance in that with some objects it is difficult if not impossible to create scans at any arbritary angle and also by being able to scan just the region of interest one can reduce the radiation exposure of the object. This has been a theoretical barrier in that in even dimentions local values are globally dependent on the integral over hyperplanes, where as in odd dimensions local values of the function can be recovered from local integrals over the hyperplanes.

Some success in local reconstruction has been achieved. Namely, Radon inversion can be achieved using wavelet transforms in a similar fashion as was with Fourier anlaysis. The formulation of the inverse Radon transform proceeds as follows

Theorem (Wavelet Inversion of the Radon Transform) 4). *let* $\{\psi_\phi\}_{\phi\in[0,\pi)}$ *be a collection of even, real–valued, addmissible functions on* $\mathbb{R}$ *such that by admissibility the functions are bounded, integrable and*

$$\sup_{\phi\in[0,\pi)}\int \frac{|\tilde{\psi}_\phi(\omega)|^2}{\omega^3}d\omega < \infty$$

and define $\Psi(\mathbf{x})$ *on* $\mathbb{R}^2$

$$\tilde{\Psi}(\omega\cos\phi,\omega\sin\phi) = 2\tilde{\psi}_\phi(\omega)|\omega|^{-1}$$

then

$$\Psi(\mathbf{x}) = \int_0^{2\pi} \psi_\phi(x_1\cos\phi + x_2\sin\phi)d\phi \tag{2-13}$$

and for any function f

$$\mathcal{W}(\Psi,f)(a,\mathbf{b}) = a^{-1/2}\int_0^{2\pi} \mathcal{W}(\psi_\phi,\check{f}\,)\,(a,b_1\cos\phi + b_2\sin\phi)\,d\phi \tag{2-14}$$

The above theorem tells us how to use wavelet transforms to construct the inverse Radon transform of a function if given its Radon transform and given a family of functions $\{\psi_\phi\}_{\phi\in[0,\pi)}$ with the required addmissibility constraints. This method outlined above is very similar to the Fourier technique employed using backprojection operator in that eqn. 2–14 almost mirrors the backprojection operation.

APPLICATIONS OF THE RADON TRANFORM 3.0

• *X − Ray Tomography.*

If a homogeneous object is exposed to an X-Ray beam then the beam's intensity upon leaving the object decreases as follows

$$I = I_0\, e^{-\mu x} \tag{3-1}$$

where I is the beam's intensity upon leaving the object, I_0 is the beam's original intensity, μ is called the *linear attenuation coefficient* which depends on the density of the material ρ and the atomic number Z. That is

$$\mu = \mu(\rho, Z)$$

Now if the beam passes through many different mediums in succession then the eqn. 3–1 changes to

$$\frac{I}{I_0} = e^{-\sum_i \mu_i x_i} \tag{3-2}$$

If however the medium has a continuous *attenuation coefficient* then

$$\frac{I}{I_0} = e^{-\int_L \mu(x)\, ds} \tag{3-3}$$

where I and I_0 are known. This is related to the Radon transform by

$$\begin{aligned} -\log \frac{I}{I_0} &= \int_L \mu(x)\, ds \\ &= \Re\mu \end{aligned} \tag{3-3}$$

• *Emission Tomography.*

Emission tomography is the process by which one seeks out to determine the position and concentration of some radioactive isotope. There are two types of Emission Tomography dependent upon the radionuclide utilized. First, if the radionuclide emits positrons e^+ which behave accordingly to the reaction $e^+ + e^- \rightarrow \gamma + \gamma$. Where each γ ray photon is emitted 180° apart. The second just emits a single photon γ.

In either case if there is no *attenuation* then

$$\check{f}(p, \phi) = \int_L f(x, y) ds$$

However, in most situations attenuation is not negligible then an alternative method must be constructed to solve for the distribution of the isotope. Starting with single photon emission problem, if a γ photon is emitted at the point $\mathbf{x} = (x, y)$ and if that photon travels along a line back to a detector defined by the following equation.

$$p = \mathbf{x}' \cdot \xi \tag{3-4}$$

Then, the integration over L is replaced by the portion of L, call this L', for which

$$(\mathbf{x}' - \mathbf{x}) \cdot \xi^{\perp} \geq 0 \tag{3-5}$$

where ξ still has its usual interpretation as in section 1. Thus, there is an modified attenuation of the form

$$\mathcal{A}(x, y; p, \phi) = exp\left(- \int_{\mathbf{x}' \cdot \xi^{\perp} \geq \mathbf{x} \cdot \xi^{\perp}} \mu(\mathbf{x}')\ \delta(p - \xi \cdot \mathbf{x}')\, d\mathbf{x}' \right) \tag{3-6}$$

and the projection data is tied to $\mathcal{A}$ by

$$\check{f}_{\mu}(p, \phi) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{A}(x, y; p, \phi)\, f(x, y)\ \delta(p - x\cos\phi - y\sin\phi)\, dx\, dy \tag{3-7}$$

This is refered to as the *attenuated* or *exponential* Radon transform.

• *Ultrasound Tomography.*

A sound pulse of an orignal pressure amplitude A_0 is transmitted by the scanning equipment and $A(p\phi)$ is received at a dectector. Now under ideal conditions we have once again

$$\begin{aligned} -\log \frac{A}{A_0} &= \int_L \mu(x)\, ds \\ &= \Re\mu \end{aligned} \tag{3-8}$$

which is identical to eqn. 3–3 with the exception that A replaces I.

Another application in the field of ultrasound is the *frequency shift method.* Which can be stated as follows

$$\int_L \mu(x)\, ds \propto \nu_i - \nu_t \tag{3-9}$$

where ν_i is the incident frequency and ν_t is the transmitted frequency.

Yet another application in the field of ultrasound is the analysis of *Propagation Time Data.* Which seeks out to reconstruct velocity distributions or equivalently index of refraction distributions.

Now the time it takes for a pulse to travel from the transmitter to the receiver is given by a line integral of the following form

$$T(p, \phi) = \int_L \frac{ds}{V} \tag{3-10}$$

where $V = V(x, y)$ is a velocity profile. If one normalizes, then

$$\tau(p, \phi) = \int_L \frac{ds}{V_w} - \int_L \frac{ds}{V} \tag{3-11}$$

where V_w is the velocity in water. Now if η is the index of refraction which is given by

$$\eta(x,y) = \frac{V_w}{V(x,y)} \tag{3-12}$$

then we have a connection to the Radon transform via the following

$$\begin{aligned} V_w\, \tau(p,\phi) &= \int_L (1-\eta(x,y))\, ds \\ &= \Re\,[1-\eta(x,y)] \end{aligned} \tag{3-13}$$

• *Nuclear Magnetic Resonance Tomography.*

The underlying principle of *Nuclear Magnetic Resonance Tomography (NMR)* is that the ground state of a nuclei with an even number of protons and even number of neutrons (*even–even nuclei*) is always in a zero intrinsic (*spin*) angular momentum state. Now *odd–even* or *even–odd* nuclei have a spin state which is an odd integer multiple of $\frac{1}{2}$, that is $I = \frac{1}{2}, \frac{3}{2}, \ldots$. Now when one of these nuclei are exposed to a magnetic field B_0 the nuclei then has $2I+1$ energy states which are equally spaced by

$$\Delta E = \frac{\mu B_0}{I} \tag{3-14}$$

where μ is the nuclear magnetic moment. Now when an energy transition occurs a photon will be emitted of the frequency

$$\nu_0 = \frac{\Delta E}{h} = \frac{\mu B_0}{2\pi h I} \tag{3-15}$$

As such, once again this relates back to *Emission Tomography* and Radon Transform theory applies as it did before.

• *Other Applications of Radon Theory.*

There are numerous applications of Radon Transform Theory in Tomograpghy. Thses include such areas of interest as Radio Astronomy, Geophysics, Optics, Engineering, Electron Microscopy, Radar, Curve and Edge Detection, Instrumentation, etc..

Concluding Remarks 4.0

The objective of this thesis is to explore wavelet analysis and Fourier analysis as it pertains to the Radon transform. More specifically, the use of efficient, effective and accurate wavlet algorithms and Fourier methods applied to the Radon inversion problem. One reason for persuing this goal is the high efficiency associated with wavelet analysis in a numerical setting via the *pryamid* algorithm, a.k.a. the *Fast Wavelet Transform (FWT)* as opposed to the *Fast Fourier Transform (FFT).*

Thus the thesis objectives are to analize the following three different methodologies in the Radon inversion problem. More specifically to compare the following methods

1. **The Fourier Central Slice Theorem**
2. **The Filter Backprojection Technique**
3. **Wavelet Transform Techniques**

Also within each of these methods analize the following

1. **Timing of Execution compared to problem size**
2. **Memory Usage compared to problem size**
3. **Accuracy of the Results both Locally and Globally**

Fourier Central Slice Theorem Analysis 4.1

The Fourier central slice theorem recalled aimed to invert the Radon transform via eqn 1–41 which restated is the following

$$f = \mathcal{F}_n^{-1}\,\mathcal{F}_1\,\check{f}$$

The discrete version of the method relies on the Fast Fourier Transform(FFT) which can be rapidally implemented an has an execution cost of $\frac{n}{2}\log_2 n$. This execution cost is however dependent upon the number of sampled points used n. If n is not a power of 2 then the execution cost would not be $\frac{n}{2}\log_2 n$. Now if the data associated with the Radon transform is stored in a matrix where each row coincides with a different value of ϕ then the Fourier Central Slice Theorem's job is to apply the FFT on each row. Inorder for it to do this and gaurantee correct results the values at which the object were scanned must coincide with specified radial scans. That is the object must be scanned at incremental values of p starting with $p = 0$ and evenly incremented by a fixed Δp. Once this is complete then the algorithm must make a change of co–ordinates from the polar pair (p, ϕ) to rectangular co–ordinates (x, y). Thus the algorithm must know which angular values were used in scanning. When the algorithm does this it must allocate additional memory inorder to store the results from the interpolation between co–ordinate systems. This means that an additional matrix must be created dynamically to facilitate this. There is an inherent problem with this interpolation. That is geometrically by refuring to figure 4.1 it is obvious that there are more points clustered about the orgin and that points spread out as one increases in p. Thus the interpolation will be very accurate near the orgin but will have larger inaccuracies as one moves further from the orgin. Thus the resulting final image will have degradations further from the orgin. After the interpolation is complete, the algorithm must take an inverse 2–dimentional FFT of the resulting matrix which will complete the reconstruction. **See Code that follows in Appendix.**

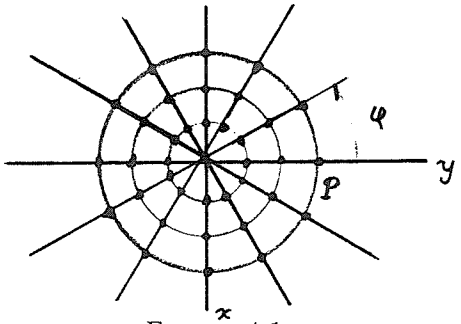

Figure 4.1.

1. Sigurdur Helgason, *The Radon Transform*, Birkhauser, Boston, Ma. U.S.A, 1980.
2. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes in Fortran, 2nd Edition*, Cambridge University Press, New York, N.Y., U.S.A., 1992.
3. Morgan Pickering, *An Introduction to Fast Fourier Transform Methods for Partial Differential Equations, with Applications*, John Wiley and Sons, Rexdale, Ont., Canada, 1986.
4. Avinash C. Kak , Malcolm Slaney, *Principles of Computerized Imaging*, IEEE Press, New York, N.Y.,U.S.A., 1988.
5. Stanley R. Deans, *The Radon Transform*, John Wiley, New York,N.Y.,U.S.A, 1983.
6. Fadil Sanotosa, Michael Vogelius, *A Backprojection Algorithm for Electrical Impedance Imaging*, S.I.A.M Journal of Applied Mathematics **50** no. 1 (Feb 1990), 216–243.
7. Robert S. Strichartz, *How to make Wavelets*, Amer. Math. Monthly (Jun–July 1993), 539–556.
8. Robert S. Strichartz, *Radon Inversion - Variations on a Theme*, Amer. Math. Monthly (Jun–July 19882), 377–384,420–423.
9. Yves Nievergelt, *Elemantary Inversion of Radon's Transform*, S.I.A.M Review **28** no. 1 (1986), 79–84.

```
#define RADIALSIZE 16
#define THETASIZE  16
#define XSIZE      16
#define YSIZE      16
#define ULI (long unsigned int)
#include <complex.h>
#include <math.h>
#include <time.h>
#include <fstream.h>
#include "invradbp.cpp"
#include "invrad2f.cpp"

double rad_func(double , double );
double my_funct(double , double );

void main(void)
{
 ofstream out("rdtest.out");
 out << "Radon Information Following\n";
 //typedef complex (*& CPTR)[MSIZE];
 //typedef complex (* CPT) [MSIZE];
 typedef complex (& MPTR) [THETASIZE][RADIALSIZE];
 //complex radon_array[THETASIZE][RADIALSIZE];
 //complex rdsave[THETASIZE][RADIALSIZE];
 //CPTR ptr1 = CPT(radon_array);
 //CPTR ptr2 = CPT(rdsave);
 //MPTR ptr1 = radon_array;
 //MPTR ptr2 = rdsave;
 complex ** radon_array = allocator( ULI(THETASIZE), ULI(RADIALSIZE) , complex(0.) );
 complex ** rdsave      = allocator( ULI(THETASIZE), ULI(RADIALSIZE) , complex(0.) );

 complex * pt100 = &(radon_array[0][0]);
 complex * pt200 = &(rdsave[0][0]);
 //CPTR ptr1   = radon_array;
 //complex ** ptr1 = (complex ** )radon_array;
 //complex ** ptr2 = (complex ** )rdsave;
 int i;
 int j;

 for ( i = 0 ; i < THETASIZE ; i++ )
   {
    for ( j = 0 ; j < RADIALSIZE ; j++ )
      {
       radon_array[i][j] =  complex(rad_func(double(j)/5.,
                                             M_PI*double(i)/double(THETASIZE)
                                            )
                                    );
       rdsave[i][j]      = radon_array[i][j];
       out << i << ' ' << j << ' ' << rdsave[i][j] << endl;
      }
   }

 unsigned long int Xdim = (unsigned long int) (XSIZE);
 unsigned long int Ydim = (unsigned long int) (YSIZE);
 unsigned long int Rdim = (unsigned long int) (RADIALSIZE);
 unsigned long int Tdim = (unsigned long int) (THETASIZE);

 clock_t t0 = clock();

 double ** out1 =
       inv_radon_via_FFTs(radon_array,Tdim,Rdim,Xdim,Ydim);

  lock_t t1 = clock();

 double ** out2 =
       inv_radon_via_bkpr(rdsave,Tdim,Rdim,Xdim,Ydim);
```

```
 clock_t t2 = clock();
 clock_t t = t1-t0;

 double  tau = double(t)/CLK_TCK;

 out << "elapsed time is for Fourier Slice  " << tau << endl;

 t = t2-t1;
 tau = double(t)/CLK_TCK;

 out << "elapsed time is for Backprojection " << tau << endl;

 out << "Output from two FFT method follows\n";

 for ( i = 0 ; i < Ydim ; i++ )
   {
    for ( j = 0 ; j < Xdim ; j++ )
      {
       out << i << ' ' << j << ' ' << out1[i][j] << endl;
      }
   }
 out << "Output from back projection method follows\n";

 for ( i = 0 ; i < Ydim ; i++ )
   {
    for ( j = 0 ; j < Xdim ; j++ )
      {
       out << i << ' ' << j << ' ' << out2[i][j] << endl;
      }
   }

 deallocat( radon_array , ULI(THETASIZE),ULI(RADIALSIZE) );
 deallocat( rdsave ,      ULI(THETASIZE),ULI(RADIALSIZE) );

 double temp;
 out << "Output true values\n";
 int halfy = Ydim/2;
 int halfx = Xdim/2;
 for ( i = -halfy+1 ; i <= halfy ; i++ )
   {
    for ( j = -halfx+1 ; j <= halfx ; j++ )
      {
       temp = my_funct(double(j)/5.,double(i)/5.);
       out << i << ' ' << j << ' ' << temp << endl;
      }
   }
}




double rad_func(double p , double phi )
{
 const long double root_pi = sqrtl( 4.0L * atanl(1.0L) );
       long double pl      = (long double) (p);
       long double phil    = (long double) (phi);
       long double temp;

 temp  = pl*pl*cosl(phil)*cosl(phil);
 temp += 0.5L*( 1.0L - 3.0L*cosl(phil)*cosl(phil) );
 temp *= root_pi*pl*expl(-pl*pl)*sinl(phil);

 return double(temp);

}
```

```
double my_funct(double x , double y )
{
 long double xl = (long double) (x);
 long double yl = (long double) (y);
 long double temp;

 temp = xl*xl*yl*expl(-xl*xl - yl*yl);

 return double(temp);

}
```

```
#if defined(INVRAD2F_CPP)
/* already defined */
#else
#define INVRAD2F_CPP
#if defined (MATH_H)
/* already defined */
#else
#include <math.h>
#endif
#if defined(COMPLEX_H)
/* already defined */
#else
#include <complex.h>
#endif
#if defined(FRFFT_CM_CPP)
/* already defined */
#else
#include "frfft_cm.cpp"
#endif
#if defined(DNORMALZ_CPP)
/* already defined */
#else
#include "dnormalz.cpp"
#endif
#if defined(ALLOCTDD_CPP)
/* already defined */
#else
#include "alloctdd.cpp"
#endif
#if defined(DEALLODD_CPP)
/* already defined */
#else
#include "deallodd.cpp"
#endif

// Fourier Central Slice Theorem

template < class kind >
double ** inv_radon_via_FFTs ( kind &              array ,
                               unsigned long int theta_dim,
                               unsigned long int Rdim,
                               unsigned long int Xdim,
                               unsigned long int Ydim
                             )
{
 kind data1 = array;

 int log2_Rdim = d_frfft_i( data1[0] , Rdim , 1. , 1u );
 int log2_Xdim = d_frfft_i( data1[0] , Xdim , 1. , 1u );
 int log2_Ydim = d_frfft_i( data1[0] , Ydim , 1. , 1u );

 if ( ( log2_Rdim >= 0 ) && ( log2_Xdim >= 0 ) && ( log2_Ydim >= 0 ) )
   {
    complex N = complex( 1./double( Rdim ) );

    for ( unsigned long int i = 0 ; i < theta_dim ; i++ )
      {
       d_frfft_i ( data1[i] , Rdim , 1. );
       d_normalizer_d( data1[i] , Rdim , N );
      }

    unsigned long int halfx = Xdim >> 1;
    unsigned long int halfy = Ydim >> 1;

    long int x1;
    long int y1;
    long int r;
```

```
long int t;

complex ** data2 = allocator( Ydim, Xdim , complex(0.) );

for ( unsigned long int y = 0 ; y < Ydim ; y++ )
  {
   if ( y < halfy )
      y1 = y;
   else
      y1 = y - Ydim;

   for ( unsigned long int x = 0 ; x < Xdim ; x++ )
     {
      if ( x < halfx )
         x1 = x;
      else
         x1 = x - Xdim;

      if ( x1 == 0 && y1 == 0 )
         r = t = 0;
      else
        {
         r = (int)
             ( sqrt( double( x1 * x1 + y1 * y1 ) ) + 0.5 );
         t = (int)
             ( atan2( double( y1 ) , double( x1 ) )
              * theta_dim / M_PI + 0.5
             );
         if ( t < 0 )
           {
            t += theta_dim;
            r  = Rdim - r;
           }
        }
      if ( ( t >= 0 ) && ( t < theta_dim ) && ( r >= 0 ) && ( r < Rdim ) )
        {
         data2[y][x] = data1[t][r];
        }
      else
        {
         data2[y][x] = complex(0.0);
        }
     }
  }

for ( i = 0 ; i < Ydim ; i++ )
  {
   d_frfft_i ( data2[i] , Xdim , -1. );
  }

complex * temp = new complex[Ydim];

for ( unsigned long int j = 0 ; j < Xdim ; j++ )
  {
   for ( i = 0 ; i < Ydim ; i++ )
     {
      temp[i] = data2[i][j];
     }

   d_frfft_i ( temp , Ydim , -1.);

   for ( i = 0 ; i < Ydim ; i++ )
     {
      data2[i][j] = temp[i];
     }
  }
```

```
    delete [] temp;

    double ** data3 = allocator( Ydim, Xdim , double(0.) );

    for ( y = 0 ; y < Ydim ; y++ )
      {
       for ( unsigned long int x = 0 ; x < Xdim ; x++ )
         {
          data3[y][x] = real( data2[(y+halfy)%Ydim][(x+halfx)%Xdim] );
         }
      }

    deallocat( data2 , Ydim , Xdim );

    return( data3 );
   }
 else
   {
    return ( 0 );
   }
}
#endif
```

```
#if defined(INVRADBP_CPP)
/* already defined */
#else
#define INVRADBP_CPP
#if defined (MATH_H)
/* already defined */
#else
#include <math.h>
#endif
#if defined(COMPLEX_H)
/* already defined */
#else
#include <complex.h>
#endif
#if defined(FRFFT_CM_CPP)
/* already defined */
#else
#include "frfft_cm.cpp"
#endif
#if defined(DNORMALZ_CPP)
/* already defined */
#else
#include "dnormalz.cpp"
#endif
#if defined(ALLOCTDD_CPP)
/* already defined */
#else
#include "alloctdd.cpp"
#endif

// Filtered Backprojection Technique

template < class kind >
kind ** inv_radon_via_bkpr ( kind &             array ,
                             unsigned long int theta_dim,
                             unsigned long int Rdim,
                             unsigned long int Xdim,
                             unsigned long int Ydim
                           )
{
 kind data1 = array;

 int log2_Rdim = d_frfft_i( data1[0] , Rdim , 1. , 1u );
 int log2_Xdim = d_frfft_i( data1[0] , Xdim , 1. , 1u );
 int log2_Ydim = d_frfft_i( data1[0] , Ydim , 1. , 1u );

 if ( ( log2_Rdim >= 0 ) && ( log2_Xdim >= 0 ) && ( log2_Ydim >= 0 ) )
   {
    unsigned long int Width = Rdim >> 1;

    double* filter = new double[Rdim];

    filter[0] =0.;

    for ( unsigned long int i = 1 ; i <= Width ; i++ )
      {
       filter[Rdim-i] = filter[i] = double(i);
      }

    complex N = complex( 1./double( Rdim ) );

    for ( i = 0 ; i < theta_dim ; i++ )
      {
       d_frfft_i ( data1[i] , Rdim , 1. );
       d_normalizer_d( data1[i] , Rdim , N );

       for ( unsigned long int j = 0 ; j < Rdim ; j++ )
```

```
        {
         data1[i][j] *= filter[j];
        }
       d_frfft_i ( data1[i] , Rdim , -1. );
      }

    delete [] filter;

    double * sin_table = new double[theta_dim];
    double * cos_table = new double[theta_dim];

    for ( i = 0 ; i < theta_dim ; i++ )
      {
       sin_table[i] = sin( (double (i))*M_PI/ double(theta_dim) );
       cos_table[i] = cos( (double (i))*M_PI/ double(theta_dim) );
      }
    long int halfx = Xdim >> 1;
    long int halfy = Ydim >> 1;

    double scale     = 1./(double(theta_dim*theta_dim) );
    double val       = 0.0;
    double position;

    double ** data2  = allocator( Ydim, Xdim , double(0.) );

    long int j;

    for ( long int y = -halfy ; y < halfy ; y++ )
      {
       for ( long int x = -halfx ; x < halfx ; x++ )
         {
          val = 0.0;

          for ( i = 0 ; i < theta_dim ; i++ )
            {
             position = double(x) * cos_table[i] + double(y) * sin_table[i];
             j = (position >= 0.0) ?
                                  ( (int) (position + 0.5 ) ):
                                  ( (int) (position - 0.5 ) );
             if ( j < 0 )
                j += Rdim;

             val += real( data1[i][j] );
            }
          data2[y+halfy][x+halfx] = val*scale;
         }
      }
    delete [] sin_table;
    delete [] cos_table;

    return( data2 );
   }
  else
    {
     return ( 0 );
    }

}
#endif
```

```
/* Function d_frfft_i Fractional fast fourier transform of an   */
/* double array                                                 */
/*                                                              */
/* USAGE: int d_frfft_i( kind &             data_p              */
/*                       unsigned long int n_sampled_i,         */
/*                       double            fractional_d,        */
/*                       unsigned long int log_only )           */
/* INPUT PARAMETERS:                                            */
/*                                                              */
/* This finction requires the following                         */
/*   1) kind & data_p    NOTE: kind is a templated type         */
/*       a)    a pointer pointing to an array of complex        */
/*             where the elements of the array are to           */
/*             be treated as the sampled function at evenly     */
/*             spaced points. The type kind is a template       */
/*             where upon dereferencing *data_p returns a       */
/*             complex also pointer addition is to be supported.*/
/*                                                              */
/*             data_d[0] = complex(f(t0))                       */
/*             data_d[1] = real(f(t0+1*delta_t))  .....         */
/*             data_d[n_sampled_i - 1] =                        */
/*                     complex(f(t0+(n_sampled_i - 1)*delta_t))*/
/*       b)    upon completion of this function the fractional  */
/*             fast fourier transfrom will be stored in the     */
/*             array accessed by the pointer data_p with the    */
/*             same format as above                             */
/*   2) unsigned long int n_sampled_i                           */
/*             an unsigned long integer specifying the number   */
/*             of sampled pts.  PLEASE NOTE that this number    */
/*             must be a POWER OF 2 and that this number        */
/*             is HALF of the TOTAL NUMBER OF ELEMENTS OF       */
/*             THE ARRAY.                                       */
/*   3) double fractional_d                                     */
/*             a double real where this number defines          */
/*             the fractional fast fourier transfrom            */
/*             as in S.I.A.M. Review                            */
/*             Vol 33. No 3. pp. 389-404 Sept 1991              */
/*                                                              */
/*             F(f(t),alpha)(k) =                               */
/*             sum(j=0,n-1) {f(j)exp(-2_pi i j k alpha/n)}      */
/*                                                              */
/*             PLEASE NOTE: alpha above is fractional_d         */
/*                                                              */
/*             PLEASE NOTE: fractional_d = 1                    */
/*             gives the standard Fast Fourier Transform        */
/*                                                              */
/*                          fractional_d = -1                   */
/*             gives the inverse Fast Fourier Transfrom         */
/*                                                              */
/* RETURNS:                                                     */
/*                                                              */
/* This function upon completion will return an int where       */
/* if this value is:                                            */
/*                  positive: successful completion             */
/*                         where value is log2(number of pts.)  */
/*                  -2:improper array size ie. not a power of 2 */
/*                  0 :array size of zero                       */
/*                  -1:negative array size                      */
#if defined(FRFFT_CM_CPP)
/* already defined */
#else
#define FRFFT_CM_CPP


#include <complex.h>
#if defined(MATH_H)
```

```
/* already defined */
#else
#include <math.h>
#   dif
    defined(BITREV_U_C)
/* already defined */
#else
#include "bitrev_u.c"
#endif

template < class kind >

   int d_frfft_i( kind &             data_p,
                  unsigned long int  n_sampled_i,
                  double             fractional_d = 1.0 ,
                  unsigned       int log_only = 0 )
{

/* LOCAL PARAMETERS:                                              */

   unsigned long int  i;
   unsigned long int  j;
   unsigned long int  istep_i;
   unsigned long int  m;
   unsigned long int  m_max_i;

   unsigned       int  log2_n_sampled_i = 0;

         long double  theta_d;
         long double  trig_real_d;
         long double  trig_imag_d;

   const long double  neg_2_pi_d       = -8.0L*atanl(1.0L);
   // -6.28318530717959;


            complex  temp;
            complex  trig_cmplx;
            complex  weight_cmplx;

            complex* temp_cmplx;

/* First thing check to see if n_sampled_i is a power of 2        */
/* or 0                                                           */

/*{{{0 program start                                             */

   j = n_sampled_i;

/*{{{1 IS number of sample points positive ?                     */

   if ( j > 0)
      {

/*{{{2 DO: shift bits right :UNTIL righter most bit is 1         */

      while ( (j%2) != 1)
        {
         j  >>= 1;
         log2_n_sampled_i++;
        }

   }}} END DO WHILE                                              */

/*{{{2 IS number of sample points NOT a power of 2 ?             */

      if ( j > 1)
```

```
        {
         return (-2);
        }
      else

/*       calculate fractional fast fourier transform                */

        {
         if ( log_only )
            return log2_n_sampled_i;

         theta_d        = (long double)(fractional_d) * neg_2_pi_d;

         temp_cmplx     = & temp;

/*{{{3 FOR LOOP : perform bit reversal on the array                  */

         for ( i = 1 ; i < n_sampled_i ; i ++ )
            {
            j = bitrev_uli( i , log2_n_sampled_i );

/*{{{4 IS j greater than i ? exchange the complex numbers            */

            if ( j > i )
               {
               *(temp_cmplx)            =  *(data_p + j );
               *(data_p + j )           =  *(data_p + i );
               *(data_p + i )           =  *temp_cmplx;
               }

/*4}}} END IF : finished exchange                                    */

            }

/*3}}} END FOR : finished bitreversal                                */


/*{{{3 the Danielson-Lanczos routine where the outer loop is         */
/*    to be performed (ln(n_sampled_i)) / ln(2)  - 1  times          */

         for ( m_max_i = 1 ; m_max_i < n_sampled_i ; m_max_i = istep_i )
            {
            istep_i = m_max_i << 1 ;

            theta_d      /= 2.0L ;

            trig_real_d   = sinl( 0.50L * theta_d);
            trig_real_d  *= (-2.0L * trig_real_d);
            trig_imag_d   = sinl(theta_d);

            trig_cmplx    = complex(trig_real_d,trig_imag_d);
            weight_cmplx  = complex(1.0);

/*{{{4 first For nested loop for Danielson-Lanczos routine           */

            for ( m = 0 ; m < m_max_i ; m ++ )
               {


/*{{{5 second For nested loop for Danielson-Lanczos routine          */

               for ( i = m ; i < n_sampled_i ; i += istep_i)
                  {
                  j = i + m_max_i;

                  *temp_cmplx      = *(data_p + j ) * weight_cmplx;
```

```
                    *(data_p + j )    = *(data_p + i ) - *temp_cmplx;
                    *(data_p + i)    += *temp_cmplx;

                    }

/*5}}} finished inner For loop for Danielson-Lanczos routine    */

                weight_cmplx  *= (trig_cmplx + 1.0);

                }

/*4}}} finished outer For loop for Danielson-Lanczos routine    */

            }

/*3}}} END FOR Danielson-Lanczos routine finished               */

          return (log2_n_sampled_i);
          }

/*2}}} END IF: finished fractional fast fourier transfrom       */

       }

    else
       {
       return (-1);
       }

/*1}}} ENDIF: finished processing                               */

/*0}}} END OF FUNCTION                                          */

#endif
```

```
/* This function performs bitreversal on an unsigned long       */
/* integer.                                                     */
/* INPUT PARAMETERS: 1)  unsigned long integer t_uli            */
/*                       number to be bitreversed               */
/*                   2)  unsigned integer k                     */
/*                       number of binary digits                */

#if defined(BITREV_U_C)
/* already defined */
#else
#define BITREV_U_C
unsigned long int bitrev_uli( unsigned long int t_uli,
                              unsigned      int k )
{
   unsigned long int j;
   unsigned long int rev_uli = 0;
   unsigned          int i;

   for ( j = t_uli, i = 1 ; i <= k ; i++ )
     {
      rev_uli += ( (j%2) << (k - i) );
      j      >>= 1;
     }

   return (rev_uli);

}
#endif
```

```
#if defined(DNORMALZ_CPP)
/* already defined */
#else
#define DNORMALZ_CPP

template < class kind >
void d_normalizer_d(
                     kind &              array,
                     unsigned long int   num_of_elements,
                     complex             number
                   )

{
 for ( unsigned long int i = 0 ; i < num_of_elements ; i++ )
   {
    *( array + i ) *= number;
   }
}
#endif
```

```
template <class kind ,unsigned long int size>
class col_vector
{
   private:

   typedef kind (*PKA)[size];

   PKA ptr_to_array;

   public:

   col_vector( PKA p )                          //constr. from ptr to array
         { ptr_to_array = p; }
   col_vector( kind * p )                       //constr. from a ptr
         { ptr_to_array = PKA(p); }
               operator PKA()                   //type cast to ptr to array
         {return ptr_to_array; }
               operator kind * ()               //type cast to a ptr
         {return ( (kind *) ptr_to_array ); }
   kind &      operator*()                      //dereference
         {return **ptr_to_array; }
   kind &      operator[]( int i )              //index
         {return **(ptr_to_array+i); }
   col_vector  operator+( int i)                //addition
         {return col_vector<kind,size>(ptr_to_array + i); }
   col_vector  operator-( int i)                //subtraction
         {return col_vector<kind,size>(ptr_to_array - i); }
   col_vector & operator+=( int i )             // +=
         {ptr_to_array += i; return *this; }
   col_vector & operator-=( int i )             // -=
         {ptr_to_array -= i; return *this; }
   col_vector & operator++()                    // prefix ++
         {++ptr_to_array; return *this; }
   col_vector & operator--()                    // prefix --
         {--ptr_to_array; return *this; }
   col_vector  operator++(int)                  //postfix P++
         {col_vector<kind,size> temp = *this; ptr_to_array++;
          return temp; }
   col_vector  operator--(int)                  //postfix P--
         {col_vector<kind,size> temp = *this; ptr_to_array--;
          return temp; }
   col_vector  operator>>=( int i)              //shift over i cols left
         {ptr_to_array =  PKA( (kind *)(ptr_to_array) + i );
          return *this; }
   col_vector  operator<<=( int i)              //shift over i cols right
         {ptr_to_array =  PKA( (kind *)(ptr_to_array) - i );
          return *this; }
   col_vector  operator>>( int i)               //shift over i cols left
         {return col_vector<kind,size>( (kind *)(ptr_to_array) + i ); }
   col_vector  operator<<( int i)               //shift over i cols left
         {return col_vector<kind,size>( (kind *)(ptr_to_array) - i ); }
};
```

```
#include <complex.h>
#include "frfft_cm.cpp"
#if defined(COLVEC_HPP)
/* already defined */
#else
#include "colvec.hpp"
#endif
template < class kind >
int dd_frfft_i ( kind &              array,
                 unsigned long int   num_of_rows,
                 unsigned long int   num_of_cols,
                 double              frac_d
               )
{
 int log2_rows = d_frfft_i( array , num_of_rows , frac_d , 1u );
 int log2_cols = d_frfft_i( array , num_of_cols , frac_d , 1u );


 if ( log2_rows >= 0 && log2_cols >= 0 )
   {
    kind P_col_cmplx = array;


    complex * ptr_cmplx;
    for ( unsigned long int i = 0 ; i < num_of_rows ; i++ )
      {
       ptr_cmplx = (complex *)(P_col_cmplx++) ;

       d_frfft_i( ptr_cmplx , num_of_cols , frac_d );
      }

    P_col_cmplx -= num_of_rows;

    for ( unsigned long int j = 0 ; j < num_of_cols ; j++ )
      {
       d_frfft_i( P_col_cmplx , num_of_rows , frac_d );

       P_col_cmplx >>= 1;
      }

    return ( log2_rows + log2_cols );
   }
 else
   {
    return ( (log2_rows > log2_cols )?log2_cols:log2_rows ) ;
   }
}
```

```
#include "dnormalz.cpp"

template < class kind >
void dd_normalizer_d(
                     kind &              array,
                     unsigned long int   num_of_rows,
                     unsigned long int   num_of_cols,
                     complex             number
                    )

{
 kind      P_col_cmplx = array;
 complex * ptr_cmplx;
 for ( unsigned long int i = 0 ; i < num_of_rows ; i++ )
   {
    ptr_cmplx = (complex *) (P_col_cmplx++);
    d_normalizer_d( ptr_cmplx , num_of_cols , number);
   }
}
```

```
#if defined(ALLOCTDD_CPP)
/* already defined */
#else
#define ALLOCTDD_CPP

//#if defined(EXCEPT_H)
/* already defined */
//#else
//#include <except.h>
//#endif
#if defined(IOSTREAM_H)
/* already defined */
#else
#include <iostream.h>
#endif

template < class kind >
kind **  allocator( unsigned long int rows ,
                    unsigned long int cols ,
                    kind dummy
                  )
{
 kind ** data;
// try
    {
      data = new kind *[rows];
      for ( unsigned long int j = 0 ; j < rows ; j++ )
       {
         data[j] = new kind[cols];
       }
      return ( data );
    }
/* catch( xalloc )
    {
      cerr << "\n Could not allocate data\n";
      return( 0 );
    }  */
}
#endif
```

```
#if defined(DEALLODD_CPP)
/* already defined */
#else
#define DEALLODD_CPP
#if defined(EXCEPT_H)
/* already defined */
//#else
//#include <except.h>
//#endif
#if defined(IOSTREAM_H)
/* already defined */
#else
#include <iostream.h>
#endif

template < class kind >
void deallocat( kind **           data,
                unsigned long int rows ,
                unsigned long int cols
              )
{
 //try
    {
      for ( unsigned long int j = 0 ; j < rows ; j++ )
       {
         delete [] data[j];
       }
      delete [] data;
    }
/* catch( xalloc )
    {
      cerr << "\n Could not deallocate data\n";
    }  */

#endif
```