Scientific Computing and Simulation, Lab 3: Inverting the Radon Transform

A CT scanner initially yields a certain projection of the density distribution in a slice through a subject (usually a human). Mathematically, what is measured by the X-Ray detectors is (trivially related to) a *Radon transform* of that density function. The problem of forming an image of the interior of the subject is then a matter of *inverting* the Radon transform.

This week I will give you a simulated density distribution for the interior of a subject's skull, and from that derive the Radon transform - which we present as a *sinogram*. Then the main challenge is to reverse that process to recover the original model from the sinogram (or Radon transform).

The Given Program

To give you a head start I will provide one new support class and a starting point for your main program.

The new support class, which will be useful later, is DisplaySinogramFT. This is almost identical to the class Display2dFT you used in the last two weeks, except it is slightly tailored to work better when the Fourier transform displayed only applies in one of two dimensions (the innermost index of the two dimensional Java array provided - or the vertical axis on the display).

The starting point for your main program is in the class Sinogram. Its main method looks like this:

```
static final float GREY_SCALE_LO = 0.95f, GREY_SCALE_HI = 1.05f ;
    // Clipping, for display only.   See for example Figure 1 in:
    //      http://bigwww.epfl.ch/thevenaz/shepplogan/


public static void main(String [] args) {
```

```
double [] [] density = new double [N] [N] ;

for(int i = 0 ; i < N ; i++) {
    double x = SCALE * (i - N/2) ;
    for(int j = 0 ; j < N ; j++) {
        double y = SCALE * (j - N/2) ;

        density [i] [j] = sheppLoganPhantom(x, y) ;
    }
}

DisplayDensity display1 =
        new DisplayDensity(density, N, "Source Model",
                                   GREY_SCALE_LO, GREY_SCALE_HI) ;

// Radon tranform of density (as measured by detectors):

double [] [] sinogram = new double [N] [N] ;

for(int iTheta = 0 ; iTheta < N ; iTheta++) {
    double theta = (Math.PI * iTheta) / N ;
    double cos = Math.cos(theta) ;
    double sin = Math.sin(theta) ;
    for(int iR = 0 ; iR < N ; iR++) {
        double r = SCALE * (iR - N/2) ;
        double sum = 0 ;
        for(int iS = 0 ; iS < N ; iS++) {
            double s = SCALE * (iS - N/2) ;
            double x = r * cos + s * sin ;
            double y = r * sin - s * cos ;
            sum += sheppLoganPhantom(x, y) ;
        }
        sinogram [iTheta] [iR] = sum ;
    }
}

DisplayDensity display2 = new DisplayDensity(sinogram, N, "Sinogram") ;

// inferred integral of density points (actually sum of density
// points, here) for laternormalization of reconstruction

double normDensity = norm1(sinogram [0]) ;


// ... Insert sinogram filtering code here! ...


double [] [] backProjection = new double [N] [N] ;
backProject(backProjection, sinogram) ;

// Normalize reconstruction, to have same sum as inferred for
// original density
```

```
        double factor = normDensity / norm2(backProjection) ;

        for(int i = 0 ; i < N ; i++) {

            for(int j = 0 ; j < N ; j++) {

                backProjection [i] [j] *= factor ;

            }

        }


        DisplayDensity display5 =

                new DisplayDensity(backProjection, N,

                                   "Back projected sinogram") ;

    }
```

You can initially run this code as it is. It should pop up initially a "source model", which is the model of a slice through a body that we are simulating. Then a little later the calculated sinogram will appear. Finally it will pop up a window of the "backprojected sinogram", which corresponds to the "pure" backpropagation approach to reconstruction discussed in the lecture - an intuitive but mathematically crude approach to image reconstruction.

Before refining the image reconstruction, you will need some explanation of the parts of this program.

The "source model" here is a famous test image used in medical image reconstruction. It is called the Shepp Logan Phantom. It represents a cross section of a head, with enlosing skull, brain, and structures within the brain. Densities within the model are quite realistic, with the skull having density 2 (twice the density of water) and all other internal structures having density close to one.

Mathematically the phantom is built up from a series of ellipses layered on top of one another, with some having positive and some negative contribution (but such that the total density at any point inside the skull is close to one).

If you google for sites that discus numerical inversion of the Radon tranform, you will find many of them use something that *looks* like the Shepp Logan Phantom as their case study. But beware that many of these sites grossly simplify the problem by using much more widely varying densities, so that a *linear* gray scale image looks similar to our rendering of the source model. But, in our display of the source model, the grey scale has been clipped so that anything with density below 0.95 looks black and anything about 1.05 looks white. With a linear grey scale, the internal structures would be barely visible. Reconstructing structures with such small differences in density is a real challenge.

The first section of the main code calculates an array density of values of the model, purely so it can be displayed (you will need to import the class DisplayDensity given in earlier weeks).

The next section of the code calculates the Radon tranform, approximating the mathematical integral of the model density using sums along a path at angle *θ*, and perpendicular distance *r* from the origin. So this section of code is simulating the actual process of doing a CT scan. The simulated output of the detectors is recorded in the array called sinogram. This two dimensional array (first index corresponding to *θ*, second to *r*) is then plotted. This looks a little like, and indeed corresponds to, traditional hard-to-interpret X-ray images you may have seen.

Then we make a naive attempt at image reconstruction using pure backprojection. You should take a quick look at the method backProject in Sinogram. As explained in the lecture, it is an integral over theta, approximated here as a discrete sum. Simple linear interpretation in the *r* dimension is used to estimate the value of the Radon transform from measured points neighbouring each required point.

Before displaying the reconstruction backProjection, one final step normalize it to have the same "norm" as the original density. Here the norm is defined as the sum or integral of the density over the whole region of space containing the subject. In a real situation we don't know this norm for the subject in advance; but luckily we can easily extract it from the measured Radon transform, as done here.

Filtered Back Projection

Fill in the code omitted above that filters the sinogram using FFTs, before backprojection. For this you should use the class FFT provided last week.

The first stage will be to calculate the complex Fourier transform of the sinogram, transforming in the innermost *r* array subscript only.

This should be rather straightforward. Here is some skeleton code:

```
double [] [] sinogramFTRe = new double [N] [N],
                          sinogramFTIm = new double [N] [N] ;
```

```
for(int iTheta = 0 ; iTheta < N ; iTheta++) {

    for(int iR = 0 ; iR < N ; iR++) {

        sinogramFTRe [iTheta] [iR] = sinogram [iTheta] [iR] ;

    }

}


for(int iTheta = 0 ; iTheta < N ; iTheta++) {

     ... do 1D FFT on a row ...

}


DisplaySinogramFT display3 =

        new DisplaySinogramFT(sinogramFTRe, sinogramFTIm, N,

                              "Sinogram radial Fourier Transform") ;
```

This should be somewhat simpler than the uses of fft1d() last week, because you are only transforming the *inner* dimension of the 2d array; only single calls to the FFT, and no need for any transposes.

The second stage is to apply a filter to the rows of Fourier transform. For simplicity you can do this in place on the arrays sinogramFTRe, sinogramFTIm. So you need a loop something like this:

```
for(int iTheta = 0 ; iTheta < N ; iTheta++) {

    for(int iK = 0 ; iK < N ; iK++) {

        int kSigned = iK <= N/2 ? iK : iK - N ;

        ... multiply Sinogram FT by abs(kSigned) ...

    }

}
```

Remember to take the *absolute* value of kSigned, and multiply elements of both sinogramFTRe, sinogramFTIm by this number.

We came across the definition kSigned in week 1 lecture and labs.

Now, in another loop over iTheta, invert the FFT on the sinogram. Again you can do this "in place" if you wish - no need to declare any new complex arrays here. (Hint: you already wrote the code to do this - just change a 1 to a -1!)

After completing this filtering, you can display the filtered sinogram by something like:

```
DisplayDensity display5 =
        new DisplayDensity(sinogramFTRe, N, "Filtered sinogram") ;
```

(Now we have done the in place inverse FT, of course, sinogramFTRe is no longer a Fourier transform - it is the filtered sinogram in real space variables.)

If you do things this way, only other change to original main method given above is that this line:

```
backProject(backProjection, sinogram) ;
```

should change to:

```
backProject(backProjection, sinogramFTRe) ;
```

The reconstructed image as displayed will be much sharper than the original version using pure backprojection, but you will have difficulty making out the internal structures of our "brain". This is because we used a linear gray scale.

So finally change the lines:

```
DisplayDensity display5 =
        new DisplayDensity(backProjection, N,
                                "Back projected sinogram") ;
```

to:

```
DisplayDensity display5 =
        new DisplayDensity(backProjection, N,
                                "Back projected sinogram",
                                GREY_SCALE_LO, GREY_SCALE_HI) ;
```

and run the program again.

Now unfortunately you will see a lot of noise in the reconstructed image. But you should be able to identify the main structures from the original model.

Exercises

The raw */K/* filter we have used is called a "ramp filter". In practice a variety of different filters are used to try to reduce noise in the image.

For example the *Ram Lak Filter* multiplies the FT by */K/* but also zeros components with */K/* greater than some CUTOFF. Try this with CUTOFF set to *N/4.*

A *Low Pass Cosine Filter* multiplies the FT by:

$$|K| \cos(\pi K/(2\ \mathrm{CUTOFF}))$$

and also sets the components to zero for */K/* greater than CUTOFF. Try this with the same or other cutoffs.

Real CT scanners will also use iterative improvement of the basic reconstructions we have considered. Unfortunately this is beyond the scope of this module.