Inverse Problems in Imaging: Mini-Project Part A and B

Jaspreet Singh Dhanjan
University College London

Tuesday 28th April 2020

Contents

1 Introduction

This report looks into the Radon transform and its inverse as the final submission for Inverse Problems in Imaging.

Part A of this coursework deals with the key aspect of tomography: how a space X of images and a space Y of data are related and how they are inherently different quantities. The Core Project will show this transition between spaces.

Part B extends this paradigm and shows the effects of improving a sinogram through inpainting techniques to gain better information of the back-projection. The implementation of the answers to this coursework are written in MATLAB R2019b in the form of Live Scripts and included within this submission.

2 Part A: Core Project

2.1 Calculate the Radon transform of an image and test the back-projection method

The Live Script Q1.MLX provides a clear demonstration of this question.

Load an image of the Shepp-Logan Phantom of size 128 $\times$ 128. We will refer to this as f_{true}. Show a picture of f_{true}.

The Shepp-Logan Phantom was loaded using MATLAB's **load** function. An identical image can be generated by using the following function: **phantom('Modified Shepp-Logan', 128)**. The modified Shepp-Logan provides a higher contrast than the regular Shepp-Logan image.

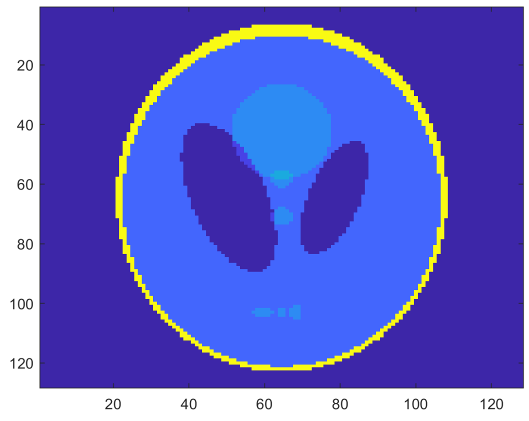

Figure 1: 128 $\times$ 128 Shepp-Logan Phantom (f_{true})

Generate the Radon transform $g = Rf$ of this Phantom in 1-degree intervals from 0-179. Display g as a 2D-image; this is referred to as the sinogram of f_{true}. What is the size of this sinogram and how is this determined?

Mathematically, the Radon transformed can be defined as a series of line integrals though $f(x, y)$ at different lengths δ from the origin:

$$R(s, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta(s - (y\cos(\theta) - x\sin(\theta))\, dx\, dy \tag{1}$$

This can be difficult to integrate analytically. However, a numerical approach can be to discretise this into pixels and approximate the integral. This approximation will be evaluated later.

Fortunately, MATLAB has a **radon** function to achieve this. The Radon transform $g = Rf$ was applied to the Phantom in 1-degree intervals from 0-179.

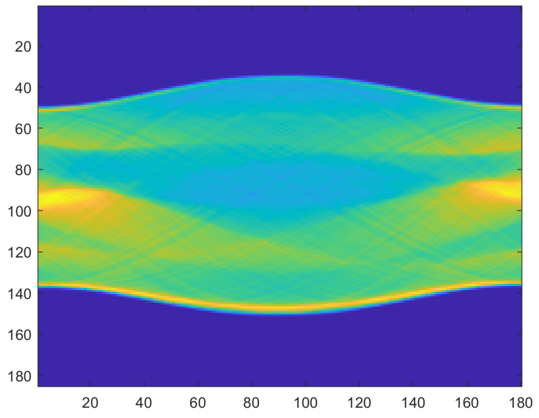

Figure 2: Sinogram of f_{true}, otherwise known as g

The above sinogram has been rescaled, however the original size of this image is 185 $\times$ 180. The columns of 180 is determined by the Radon transform's projection of intensities along the radial line orientated at the given angles (0-179). The rows is determined by the number of projection samples and can be calculated by dividing the number of angles by the degree separation.

Compute the unfiltered back-projection and apply it to the sinogram data you generated. What is the size of the back-projected image?

To reconstruct the original image, the inverse Radon transform must be applied to the sinogram. This reconstruction is also referred to as a back-projection. Numerically, this is defined as:

$$h(x,y) = \int_{-\infty}^{\infty}\int_{0}^{\pi} b(s,\theta)\delta(s-(y\cos(\theta)-x\sin(\theta))\,d\theta\,ds \tag{2}$$

To specifically compute the unfiltered back-projection, a special set of parameters must be passed to MATLAB: **iradon(g, angles, 'linear', 'none')**.

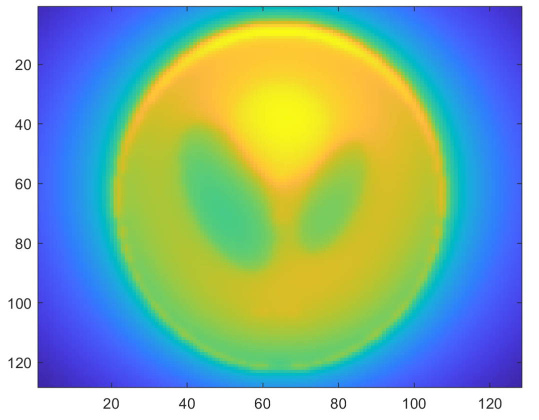

Figure 3: Unfiltered back-projection of sinogram

The output size of this unfiltered back-projection is a 130 × 130 matrix. This is not the size of the original input f_{true}. Therefore, the method call needs an extra two arguments: **'FREQUENCY_SCALING=1'** and **'OUTPUT_SIZE=n'**. The new function call is: **iradon(g, angles, 'linear', 'none', 1, n)**. Below shows the result, a 128 × 128 reconstructed unfiltered image.

Compute the filtered back-projection and apply it to the sinogram data g that you generated. Verify that this gives a good estimate of the inverse of the Radon transform.

Removing the noise from this back-projection would significantly improve the clarity of the reconstruction. To achieve this, a high-pass filter is applied to the sinogram in the frequency domain. Performing this in MATLAB is slightly different, the function call to achieve this is: **iradon(g, angles, 'linear', 'Shepp-Logan', 1, n)**. Comparing this to the original Phantom image f_{true}, they begin to look strikingly similar.

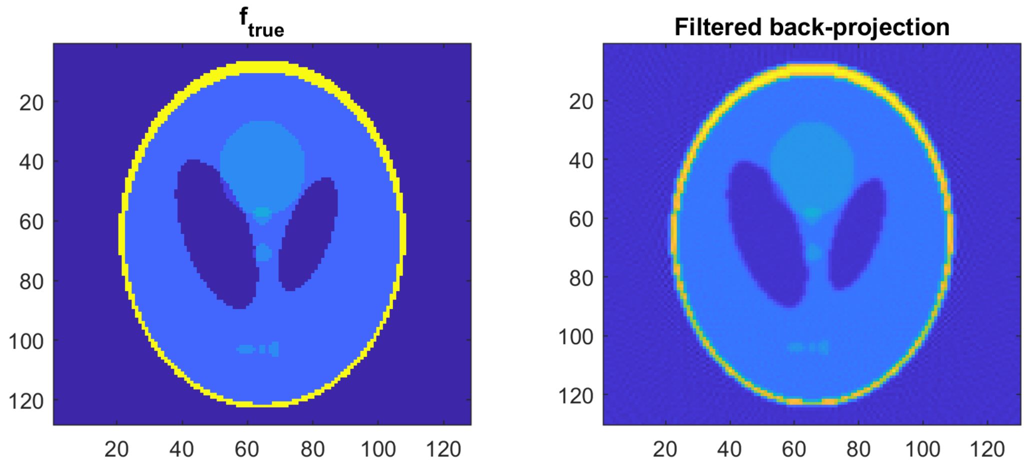

Figure 4: Unfiltered back-projection of sinogram

The intensity values of the reconstruction are normalised, just as the original. However, there appears to be some noise at the high-frequency components of the reconstructed Phantom. To come up with a metric associated with the quality of the reconstruction, the difference of the two matrices was calculated, otherwise known as the residual.

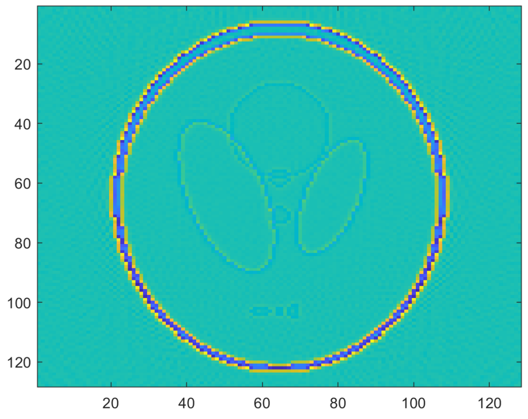

Figure 5: Residual of filtered back-projection and f_{true}

This residual contains the noise from the filtered back-projection and the high-frequency features, such as the edges from the ovals in the Phantom. In an ideal reconstruction, this residual should be a black image, as the difference between the back-projection and f_{true} should be minimised. A numeric value of 0.0043 was associated to this residual, calculated as the sum of squared differences in pixel intensity, divided by the number of pixels, n.

Add noise to the data g and test how the error in your reconstruction grows with the scale of the measurement noise.

To test the filtered back-projection further, noise is added to the Phantom. This is referred to as $g_{noisy}(\sigma)$, where σ is the standard deviation of Gaussian noise. Three levels of σ were chosen in this experiment: 1, 2 and 5.

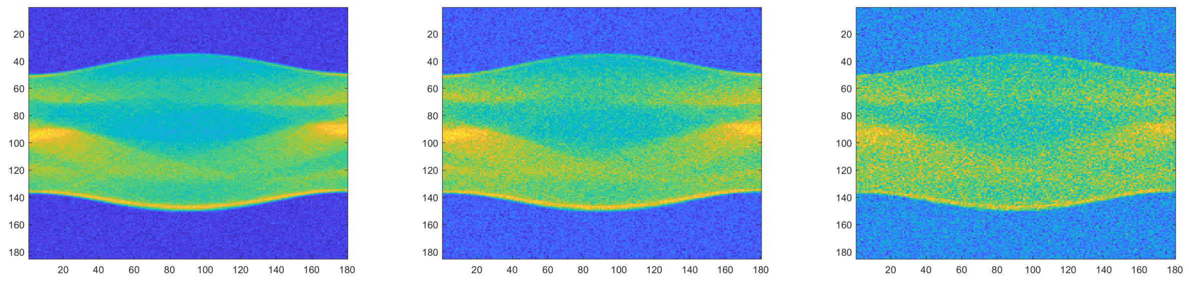

Figure 6: Three noisy Radon-transformed Phantom test images ($g_{noisy}(\sigma)$). Left: $\sigma = 1$. Middle: $\sigma = 2$. Right: $\sigma = 5$

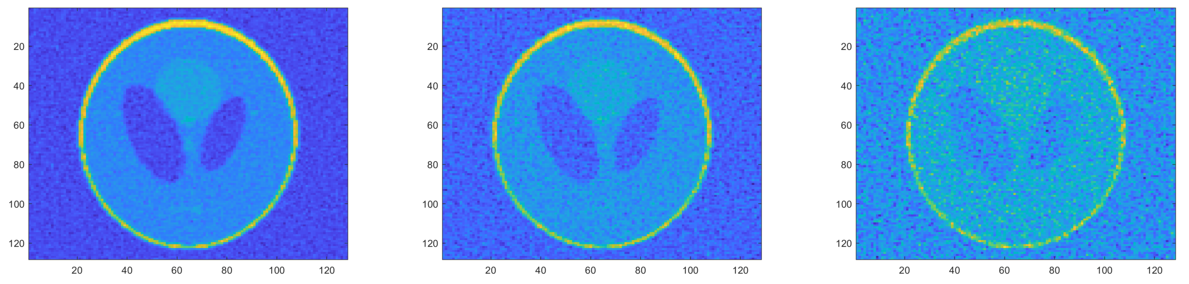

Figure 7: The result. Filtered back-projection of $g_{noisy}(\sigma)$ images. Left: $\sigma = 1$. Middle: $\sigma = 2$. Right: $\sigma = 5$

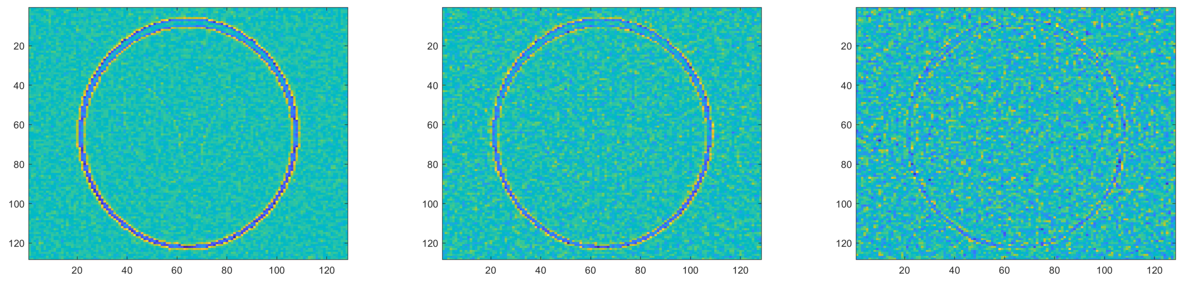

Figure 8: The residual. Difference between $g_{noisy}(\sigma)$ images and f_{true}. Left: $\sigma = 1$. Middle: $\sigma = 2$. Right: $\sigma = 5$

A filtered back-projected set of images with varying levels of noisy is shown in Figure 7, against the corresponding Radon images in Figure 6. The difference between $g_{noisy}(\sigma)$ and f_{true} is shown in Figure 8. The numeric value of this residual is used to evaluate the growth of error in the reconstruction against f_{true}.

Noise (σ)	**Residual**
0 (No noise)	0.0043
1	0.0058
2	0.01
5	0.04

Table 1: Residual of noise in three examples of $g_{noisy}(\sigma)$

The outcome of the experiment above shows a 0.5% loss of colour accuracy when $\sigma = 1$. The reconstruction appears stable and similar to f_{true}. When the standard deviation is doubled, $\sigma = 2$, there is a 1% loss of colour accuracy. The image is still recognisable, but lossy. When $\sigma = 5$, the reconstruction begins to lose structure and there is a 4% loss in colour accuracy.

It is unknown whether or not using the two additional arguments **'FREQUENCY_SCALING=1'** and **'OUTPUT_SIZE=n'** in the inverse Radon transform and if it could have contributed to a further increase in noise of the resultant image. More investigation is required to determine this.

2.2 Calculate an explicit matrix form of the Radon transform and investigate its SVD

The Live Script Q2.MLX provides a clear demonstration of this question.

One can learn more about the Radon transform from looking at the data space. The forward projection of a single point can be represented as:

$$f^{\delta} = \delta(r - r_0) \equiv \delta(x - x_0, y - y_0)$$

Thus, performing the Radon transform of this is:

$$g(s, \theta) = \int_{\infty}^{\infty} \int_{-\infty}^{\infty} \delta(x - x_0, y - y_0)\delta(s - (y\cos(\theta) - x\sin(\theta))\, dx\, dy$$

$$= \delta(s - y_0\cos(\theta) - x_0\sin(\theta)) = \delta(s - r_0\cos(\theta_0 - \theta)$$

where both r_0 and θ_0 is the polar representation of r_0.

i) Full range of angles with four-degree separation

This experiment can be executed on a larger scale. For example, given an image of size 64×64, for each pixel the Radon transform of that image can be calculated. Doing so will show the polar equivalent of that pixel in the sinogram space.

The image below is a visualisation of the sparse matrix A. This matrix represents the explicit matrix for the Radon transform of angles from $0 \rightarrow 180$ with 45 projections at four-degree separation.

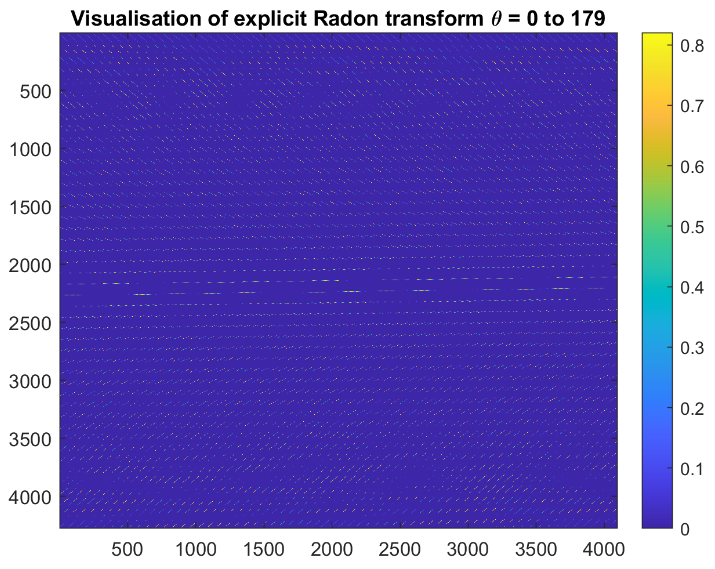

Figure 9: Visualisation of explicit matrix A from $0 \rightarrow 179$ with a four-degree separation

Although difficult to understand initially, the matrix shows the radial position move as the y-axis grows. As it reaches parallel with the x-axis, it begins to move back to its original position. This is the spacial representation of the sinusoidal nature of the sinogram.

To gain the first thousand singular values of this matrix A, the MATLAB function **svds(A, 1000)** was ran.

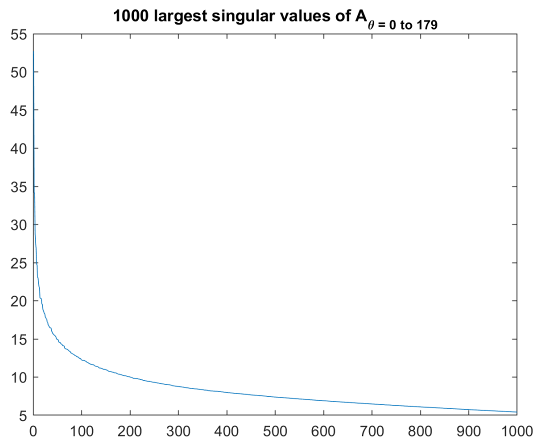

Figure 10: A plot of the largest singular values $\sigma_i = \Sigma_{ii}$ of A

This almost-asymptotic graph shows the fall off of magnitude of singular values for the matrix A.

ii) Limited angles with one-degree separation

Maintaining the same number of projections, but limiting the range from $0 \rightarrow 44$ will yield a different matrix. This will be referred to as A'. Below shows the image of A'.

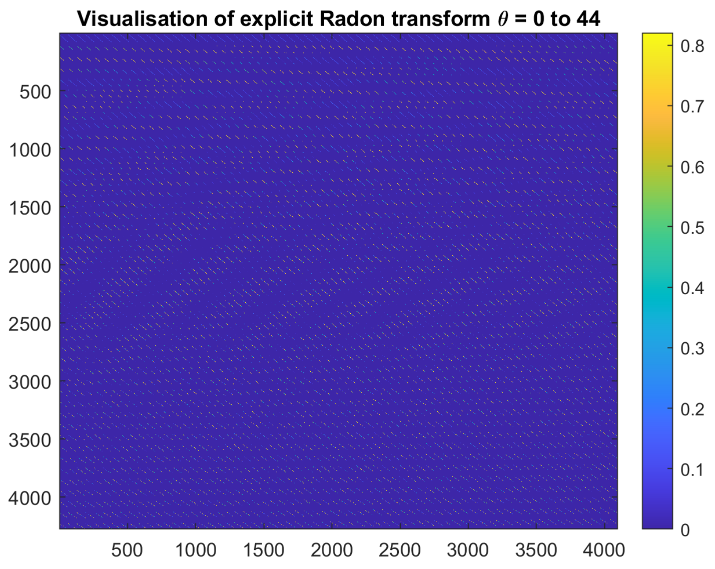

Figure 11: Visualisation of explicit matrix A' from $0 \rightarrow 44$ with a one-degree separation

The matrix A', although similar to matrix A, shows a much different direction of radial positions. The direction is more limited, much like the input angles.

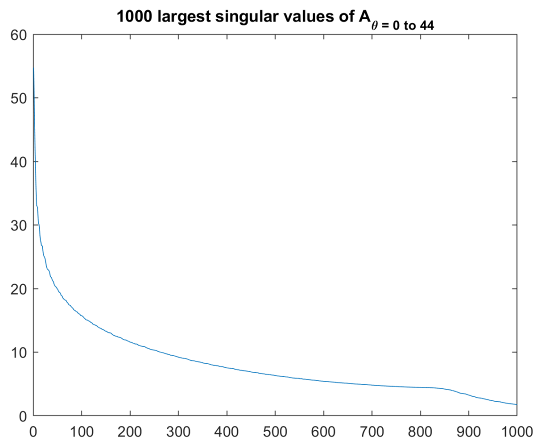

Figure 12: A plot of the largest singular values $\sigma_i = \Sigma_{ii}$ of A'

These singular values tend less towards zeros, until a certain point between the 800th and 900th largest value.

2.3 Implement a matrix-free regularised least-squares solver for the Radon Transform

The Live Script Q3.MLX provides a clear demonstration of this question.

In the last section, the explicit matrix to the Radon transform was 4275×4096. This translates to 17,510,400 32-bit floating point values, or around seventy megabytes of memory. Even by today's standards this is quite a large figure to store in memory, especially considering that the explicit matrix had a limited range or wide degree separation.

Instead, one could consider a matrix-free method. Since the back-projection is not the inverse of the Radon transform, but the adjoint and since the unfiltered back-projection is a convolution, it can be sovled using a Krylov solver.

This example will use preconditioned conjugate gradients (PCG) to find the regularised solution to:

$$(A^T A + \alpha L) f_* = A^T g$$

Firstly, the solver is passed a function that computes $(A^T A + \alpha I) f_*$, this can be found in **ATA.m**. This uses a value of α found from the minimum of the discrepancy function, which used zeroth and first-order Tikhonov regularisation. A handle to this was passed to the PCG function

i) Full range of angles with four-degree separation

For this, a noiseless Phantom was given to the solver. The tolerance used a value of 1×10^{-8} and a maximum iteration value of five hundred. This produced the following:

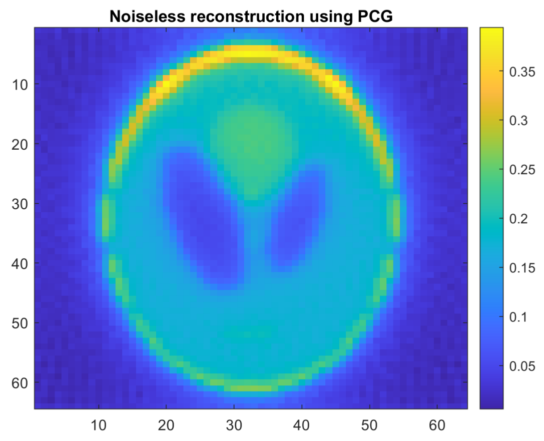

Figure 13: Noiseless PCG Reconstruction for angles $0 \rightarrow 179$

To test this further, white Gaussian noise with a standard deviation of $\sigma = 0.1$ was added to the original Phantom. This produced the following results:

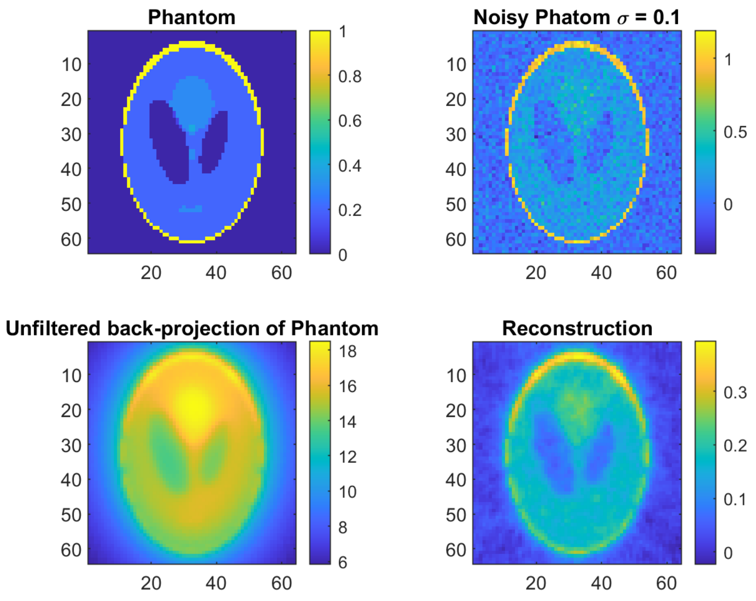

Figure 14: Top left: noiseless Phantom. Top right: noisy Phantom. Bottom left: unfiltered back-projection of the Phantom. Bottom right: reconstruction using PCG

Interestingly, the reconstruction looked very similar to the original filtered back-projection. According to the PCG function output, this reconstruction converged at iteration 484 to a solution with relative residual 2.5×10^{-9}.

ii) Limited angles with one-degree separation

Using the same tolerance and same number of maximum iterations, but with a limited range of angles produced the following:

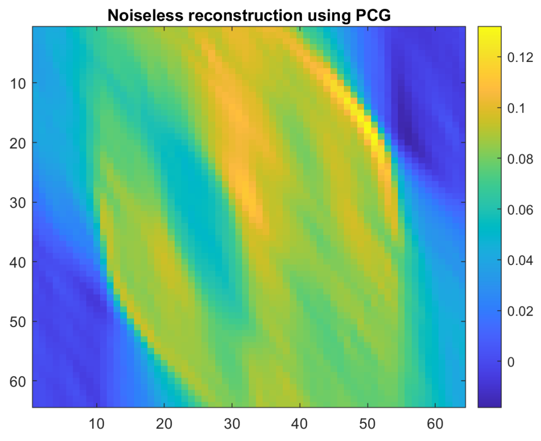

Figure 15: Noiseless PCG Reconstruction for angles $0 \to 44$

Again, this appeared to be a good reconstruction. Adding the same amount of noise produced the following results:

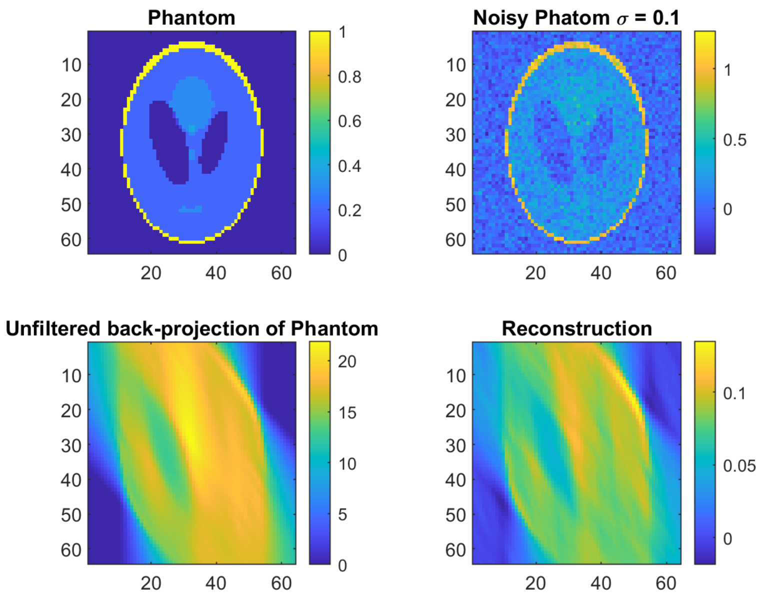

Figure 16: Top left: noiseless Phantom. Top right: noisy Phantom. Bottom left: unfiltered back-projection of the Phantom. Bottom right: reconstruction using PCG

This reconstruction converged at iteration 469 to a solution with relative residual 6.1×10^{-9}.

2.4 Write a Haar wavelet denoiser

The Live Script Q4.MLX provides a clear demonstration of this question.

Take any (monochrome) image of your choice. Calculate the Haar wavelet transform of this image. Plot some of the coefficients and explain what you see.

In image processing techniques, a Laplacian pyramid is used to decompose an image into high and low frequencies. This is achieved by subtracting the Gaussian of an image from the original image, also known as the Laplacian of a Gaussian. This technique can be used to blend two images together seamlessly. However, this is achieved in the spatial domain and can take up a lot of memory!

Instead of this, an image can be decomposed into the frequency domain using the Fourier transform. This is good, however, the Fourier transform fails to capture both frequency and location in time information.

Another approach would be to use a discrete wavelet transform as it can capture both frequency and location in time information. Just as before, this image information can be decomposed into an orthonormal basis with multiple scales. It is important to choose an image of size $2^n \times 2^n$ as the Haar transform operates over $\log_2(N)$ scales. This example will use an image with strong edges to further illustrate the denoising mechanism, which will be explained in more detail later.

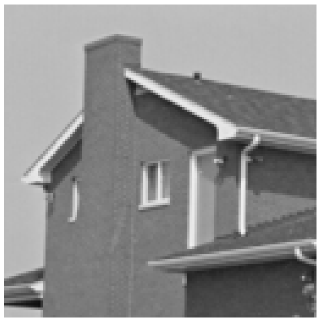

Figure 17: Original house image

The function **haart2** will perform the Haar transform of a given 2D matrix in MATLAB. The output is four variables:

- Approximation coefficients (A)
- Horizontal detail coefficients (H)
- Vertical detail coefficients (V)
- Diagonal detail coefficients (D)

For horizontal, vertical and diagonal detail coefficients, a cell of seven matrices are returned. Each of the seven matrices represent the coefficients at different scales. Again, since this is a 128 $\times$ 128 images, the number of scales will be $\log_2(128) = 7$. The size of these detail coefficients will half until a 1 $\times$ 1 pixel is reached.

The images below show the first orthonormal coefficients, which have been rescaled by factor two. The images below use a colour map unlike the Figure 10: Original house image. The horizontal image shows the horizontal edges of the house. The vertical image shows the vertical images of the house. The diagonal image is less clear, but does show the high-frequency diagonal components.

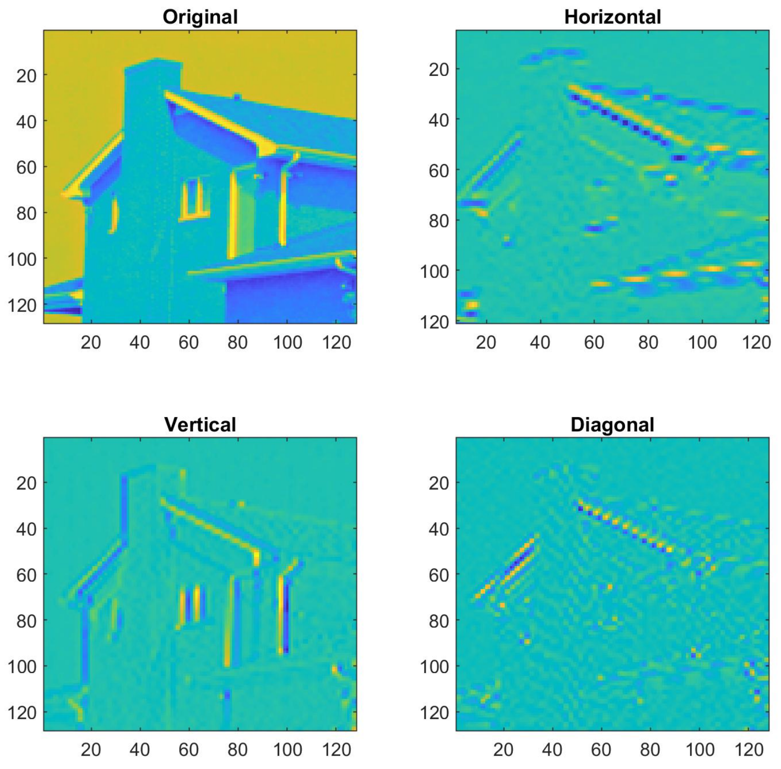

Figure 18: Top left: original image. Top right: horizontal image rescaled by factor two. Bottom left: vertical image rescaled by factor two. Bottom right: diagonal image rescaled by factor two

Thus, each of these matrices can be combined to produce an image that looks somewhat similar to the original image. The image below visualises this.

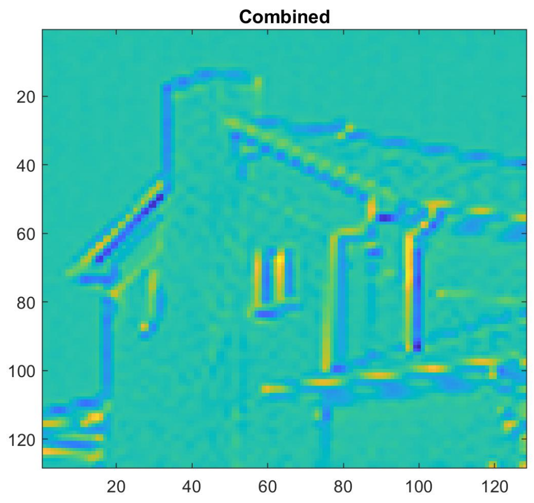

Figure 19: Combined image $(H_1 + V_1 + D_1)$

Reconstruct the image from the coefficients by calling the inverse wavelet transform. Check if your reconstructed image coincides with the original.

Reconstructing the original image from the above matrices is simple. The MATLAB function to do this is **ihaart2(a, h, v, d)**.

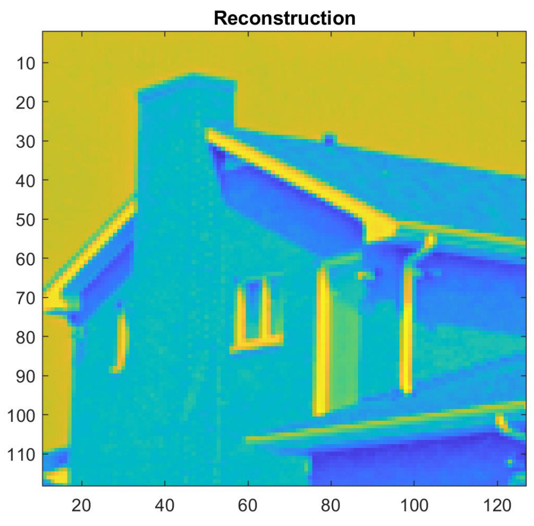

Figure 20: Reconstruction

This reconstruction appears to be very good. To test this further, the residual should calculated to associate a metric to this reconstruction.

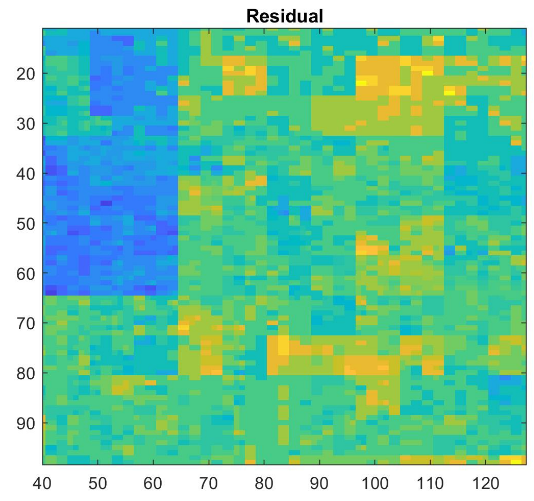

Figure 21: Residual (Reconstruction - Original)

There are no visible features in this residual. Furthermore, the values of the matrix were very small. In fact, out of 16384 elements of this 128×128 matrix, 4584 elements were zero (almost 28%!) Furthermore, the norm of the remaining elements of the residual matrix was 1.4922×10^{14}.

Write a function that implements thresholding for a given range (the different scales of your wavelet coefficients) and threshold parameter, and form a modified image by performing the inverse wavelet transform on the thresholded coefficients.

The included function **thresholdFunction.m** was relatively simple. It took the coefficients and applied MATLAB's **wthresh** function for each matrix. The thresholding function also included an additional parameter for hard or soft thresholding. This could be a necessary parameter, as some images could have many more edges than others. The effects of this will be demonstrated in the next part.

To calculate a value for the thresholding parameter a new function was created called **determine_threshold.m**. This function concatenated all matrix values and placed them in a sorted vector. If the user required a threshold value of 80%, then the function would multiply 0.8 by the size of the vector to find the smallest element within that subset.

Create a noisy version of your original image and perform denoising by thresholding of the wavelet coefficients. Investigate the effect of changing the range and the threshold parameter.

Additive white Gaussian noise with a standard deviation of 0.2 was added to Figure 10: Original house image. This image was then decomposed using the **haart2** function. The coefficients were then passed to the **determine_threshold** function to find an appropriate threshold value. This value, along with the coefficients were then passed to the function **thresholdFunction**. For this specific image a hard thresholding approach was selected. The newly-threshold coefficients were then passed to the inverse function **ihaart2** to get back the denoised image.

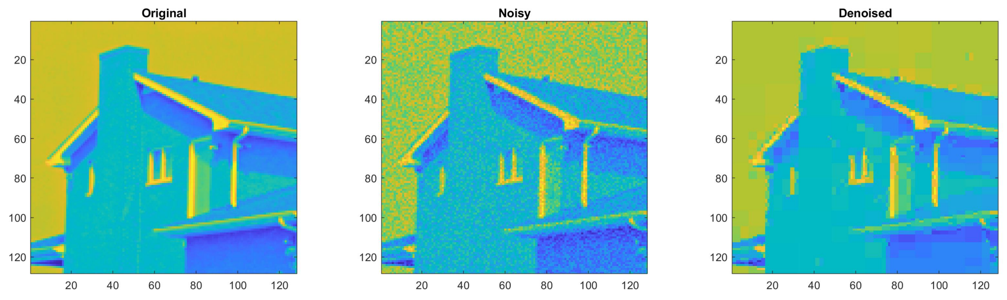

Figure 22: Left: Original image. Middle: Original image with additive Gaussian white noise $\sigma = 0.2$. Right: Reconstruction using Haar wavelet denoising

The images above shows the denoised reconstruction. For this reconstruction, 95% of the smallest coefficients were chosen for the threshold value, which translated to a value of 0.19 (at index 15563). This reconstruction used a hard thresholding parameter.

When choosing a soft thresholding parameter, the user must be careful. Below shows the same reconstruction as Figure 15, however, soft thresholding was used.

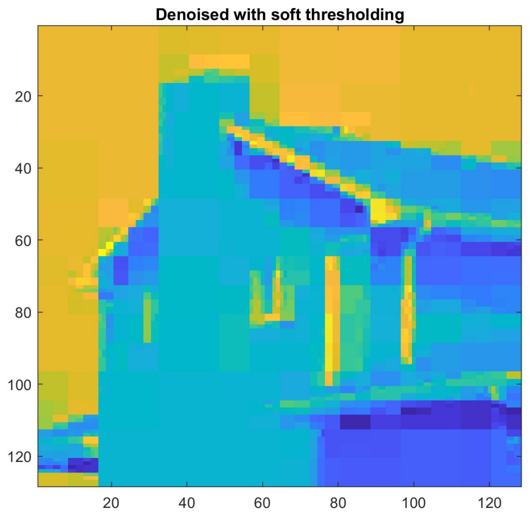

Figure 23: Result using ’s’ instead of ’h’ in MATLAB’s **wthresh** function

The reconstruction is still somewhat good and better than the original. However, the high-frequency elements such as the edges of the chimney and cladding could not be preserved entirely.

So far, all shown denoising approaches have used the full range of scales for the coefficients. However, this might not be necessary. As the matrix of coefficient values half in size at every level, noise will not be preserved as much. Therefore, denoising at these levels will not be as necessary. Below shows a reconstruction with thresholding at the first four levels.

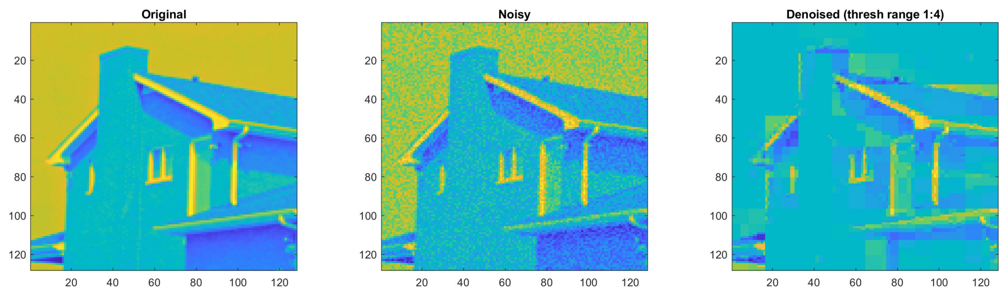

Figure 24: Left: Original image. Middle: Original image with additive Gaussian white noise $\sigma = 0.2$. Right: Reconstruction using Haar wavelet denoising from ranges one to four

The denoising algorithm still works and much of the noise has appeared to be reduced. However, the colour map shows that some of the values around the diagonal edges are inconsistent with the other parts of the image. More investigation is required to understand why this is.

2.5 Iterative soft-thresholding for X-ray tomography

The Live Script Q5.MLX provides a clear demonstration of this question.

Iterative soft-thresholding builds on the last section. This reconstruction algorithm works by solving the following:

$$\frac{1}{2}||A\mathbf{f} - \mathbf{g}||_2^2 + \alpha||W\mathbf{f}||_1 \rightarrow \min \tag{3}$$

As demonstrated previously, this algorithm will only use soft thresholding. A better reconstruction can be found through an iterative approach.

Below shows the original noisy image corrupted with white Gaussian noise with standard deviation of $\sigma = 0.1$.

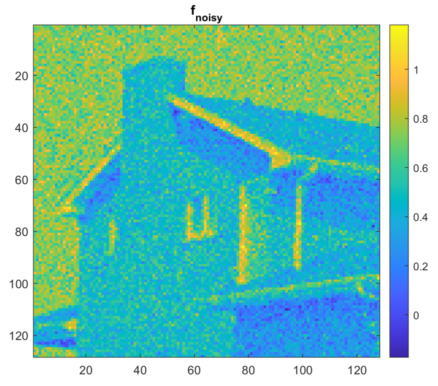

Figure 25: Noisy image

Applying a soft-thresholding algorithm with five iterations produced the following:

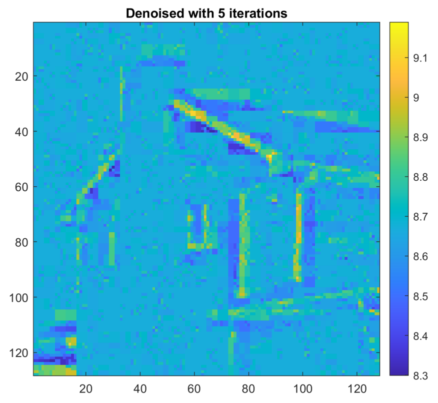

Figure 26: Denoised

3 Part B: Inpainting in Sinogram Space

The folder PROJECTB contains a collection of routines for the following problem.

Image manipulation tools such as Photoshop's "Content-Aware Fill" make use of image inpainting techniques to recover missing pixels. These algorithms can be applied to medical imaging to gain better insight into the captured photography. In many instances, information can be sparse or incomplete.

Similarly to the previous denoising problems, this approach will use a regularised inversion problem. Consider two variables f: the reconstructed image and g: the image with missing data.

$$f_{\text{recon}} = \text{arg min}\left[\phi(f,g) = \frac{1}{2}||g - I_\omega f||^2 + \alpha\Phi(f)\right] \tag{4}$$

This investigation will look into two incomplete types of sinogram information: limited angle and undersampled.

3.1 Limited Angle Sinograms

A limited angle sinogram is generated by creating a 128 × 128 Phantom. The Radon transform of this Phantom is taken. An equally sized matrix of ones, referred to as a mask is created. Values from (20, 70) to (160, 120) are set to zero. This mask is then multiplied to the sinogram to produce a limited angle sinogram. The inverse Radon transform of this can be taken to see the effects on the filtered back-projection.

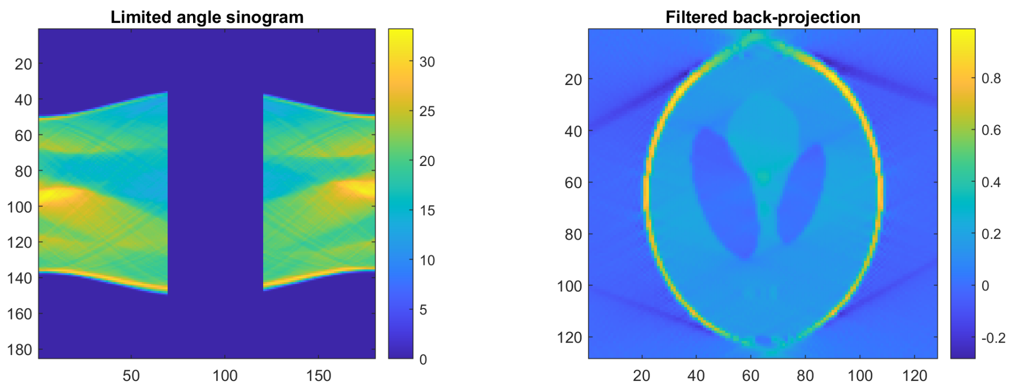

Figure 27: Effects of a limited angle sinogram. Left: limited angle sinogram. Right: filtered back-projection of the limited angle sinogram

This back-projection slightly resembles the Shepp-Logan Phantom from Figure 1, but it could be improved. By implementing Equation 4, the limited angle sinogram could be filled in with non-zero values and could yield a better back-projection. It is important to note that this inpainting algorithm uses information only from the limited angle sinogram, unlike the demo example from the lecture notes.

The image below shows the Laplacian used in the calculation for the inpaint.

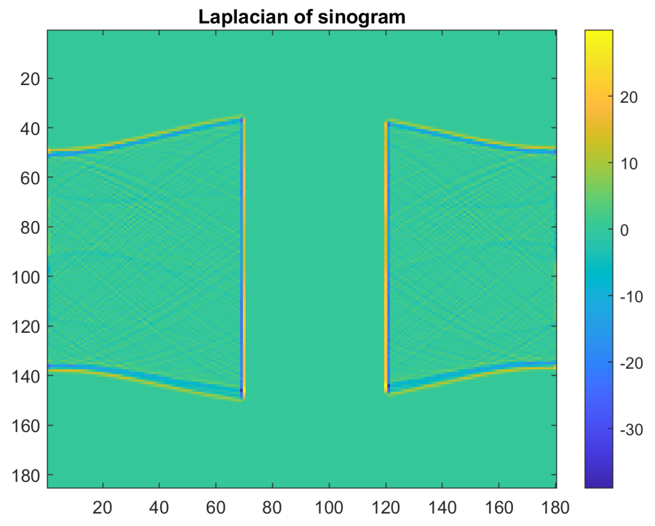

Figure 28: Laplacian of the limited angle sinogram

The final results are shown below. The Phantom image has significantly improved, such as the shape of the exterior oval. The also colour begins to resemble Figure 1.

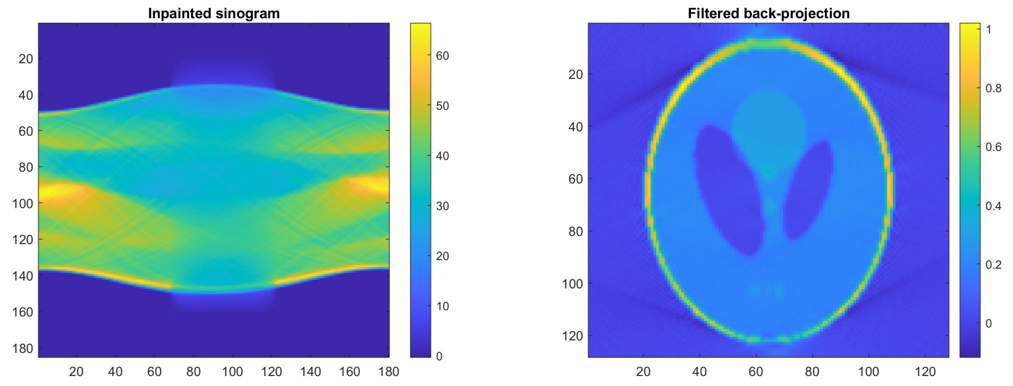

Figure 29: Left: inpainted sinogram. Right: back-projection of sinogram

According to MATLAB, the PCG took 100 iterations converging to the tolerance 1×10^{-6}. The regularisation term α was set at 1×10^{-2}.

However, the inpainted pixels of the sinogram do have some blur and this can be attributed to Hessian.

The paper "Limited angle CT reconstruction by simultaneous spatial and Radon domain regularization based on TV and data-driven tight frame" by Wenkun Zhang et al. suggests a TV denoising approach in this case. Using notes from earlier lectures, this was applied to sinogram and produced the following.

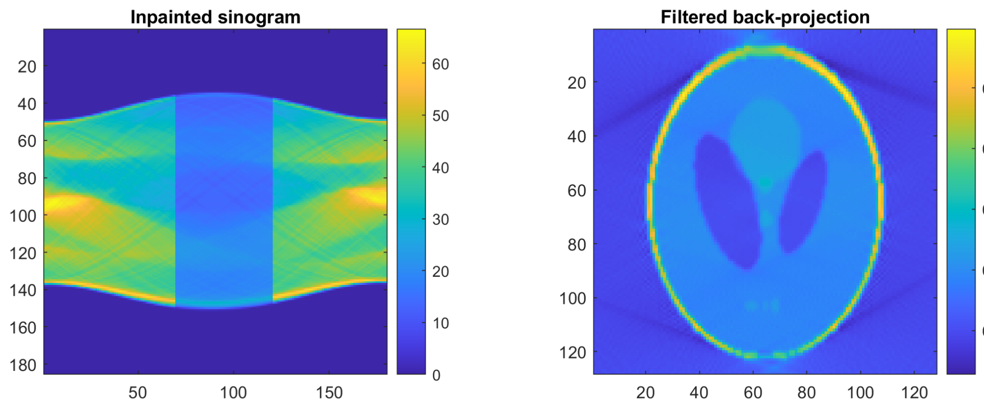

Figure 30: Left: inpainted sinogram with Total Variation deblurring. Right: back-projection of sinogram

The inpaint looks far better and the edges have become stronger in the image space of the sinogram. However, this has not made much difference to the filtered back-projection and more investigation is required as to why this is.

3.2 Undersampled Sinograms

An undersampled sinogram can be caused by missing projection samples. To illustrate this, a mask with equals slots of zeros and ones was applied to Phantom's sinogram.

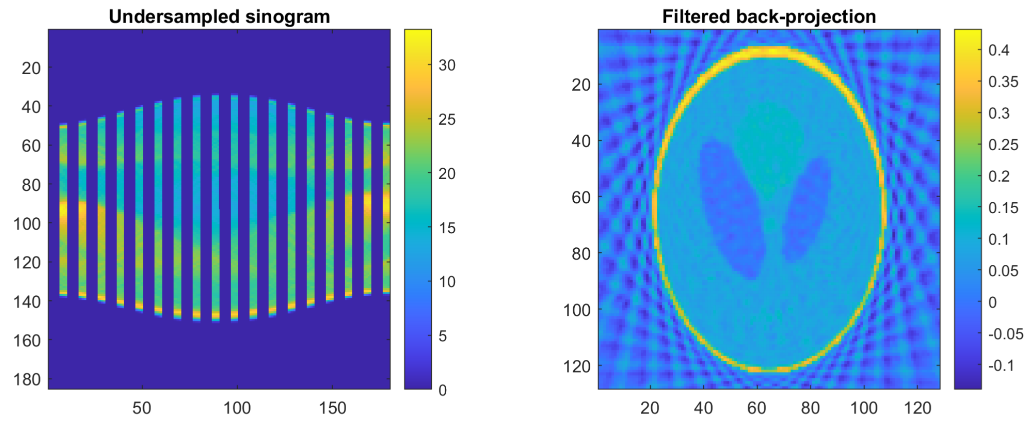

Figure 31: Left: undersampled sinogram. Right: back-projection of sinogram

The exact routines from the limited angle inpainting algorithm was applied to the undersampled sinogram. This produced the following:

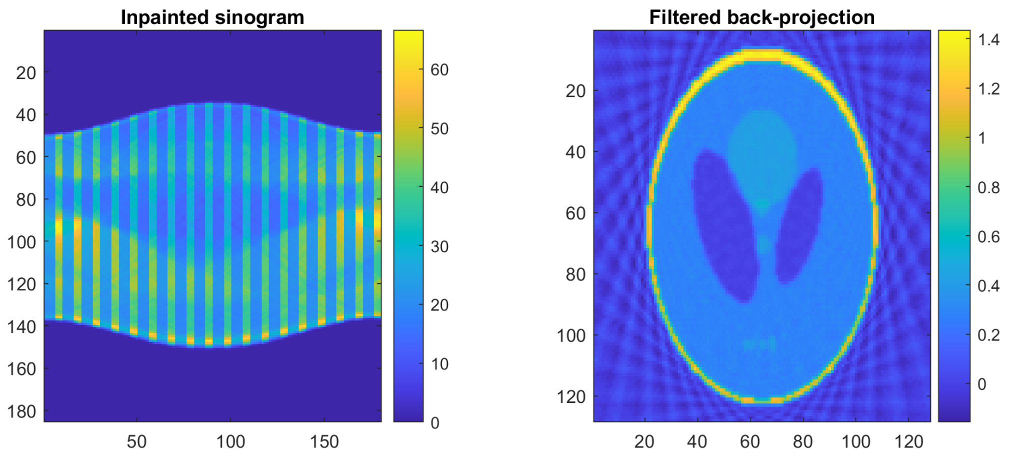

Figure 32: Left: inpainted and deblurred undersampled sinogram. Right: back-projection of sinogram

The inpaint applied to the sinogram in addition to the deblurring produced a good result. It does not quite match the original sinogram of a Phantom, but it is a good start. The filtered back-projection of this also looks much improved. Much of those noise of the inner-oval has decreased, but the lines outside this are still present.