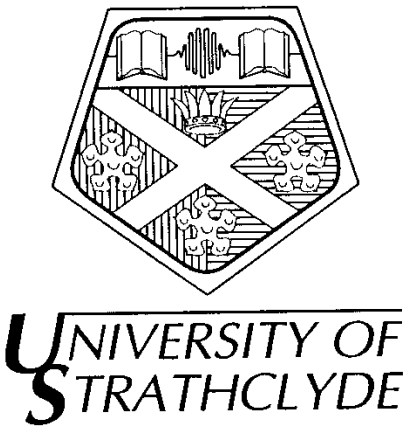


RCNDE Acoustics and Ultrasonics

Coursework

Written by

Euan Alexander Foster

May 2019

Email: e.foster@strath.ac.uk
Tel: 07539 450593

Contents

1 EXERCISE 3 DATA PROCESSING

1.1 ABSTRACT

Three separate problems were given and solved for. The first problem was to calculate frequency dependent attenuation for a given A-Scan as a function of frequency. The second problem was to calculate the reflection coefficient of an adhesive layer as a function of frequency. Lastly, the third problem was to calculate the thickness of a component from a given B-Scan.

Each problem agreed with the majority of the theory published in literature, with the only exception being the thickness measurement from a B-Scan were experimental differences were noted from one data point to the next.

1.2 INTRODUCTION

For exercise 3, there were three unique problems presented. For the first problem, an ultrasonic A-Scan from a 7.8 mm thick Perspex plate was given. The A-Scan was performed via pulse-echo immersion testing with a 2.25 MHz transducer. From the data given, this allowed for the attenuation as a function of frequency to be calculated.

The reflection coefficient of an adhesive layer was calculated for the second problem. Two different A-Scans were given for this to be performed – one of an adhesive free region and one of a bonded region. The A-Scans were obtained via pulse-echo immersion testing with a 10 MHz transducer.

For the final problem, a B-Scan containing a series of A-Scans of a bearing raceway were given. The data was obtained via immersion pulse echo testing with a 5 MHz transducer. The thickest point was noted to be 55 mm. From the data given, the thickness at each point in the B-Scan was calculated.

1.2.1 EXERCISE 3A – THEORY & RESULTS - FREQUENCY DEPENDENT ATTENTUATION OF PERSPEX FROM AMPLTUDE SPECTRUM

The A-Scan of the 7.8 mm Perspex block is given in Figure 1. As can be seen the voltage time trace contains no noise, so no filtering was performed. An automatic MATLAB function was developed to isolate each wave packet in time. Details of the

function are presented in APPENDIX A – FN_WAVEPACKET - WAVE PACKET ISOLATION.

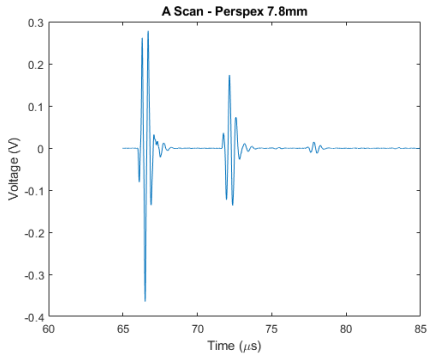


Figure 1 - A-Scan Time Trace

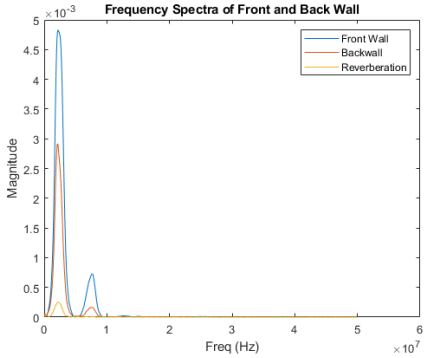


Figure 2 - Frequency Spectra of Wave Packets

With each of the wave packets isolated, the Fast Fourier Transform could be taken to reveal its' frequency spectra. The wave packets were padded with zeros to improve the resolution of the frequency content [1]. Figure 2 shows the frequency spectra of each wave packet within the A-Scan. As can be seen, the magnitude associated with each frequency reduces from the front wall echo to the back wall echo due to attenuation. Furthermore, the frequency content was restricted to ±6 dB of the centre frequency of 2.25 MHz. This was the assumed bandwidth of the transducer, as the magnitude of the maximum frequency has reduced by approximately one half.

With each frequency spectra known and truncated to the bandwidth of the transducer, the attenuation as a function of frequency could be calculated in accordance with the exponential decay demonstrated in Eq.1 & 2. Where $A_0(\omega)$ is the frequency amplitude associated with the front wall is, $A(\omega)$ is the frequency amplitude associated with the back wall, α is the attenuation coefficient in Np/m and d is the distance travelled by the ultrasonic wave in meters.

$$\frac{A(\omega)}{A_0(\omega)} = e^{-\alpha(\omega)d} \tag{1}$$

$$\alpha(\omega) = -\frac{1}{d}\ln\left(\frac{A(\omega)}{A_0(\omega)}\right) \tag{2}$$

The resulting attenuation as a function of frequency is shown in Figure 3. As can be seen the attenuation increases significantly with increases in frequency. Details on the MATLAB code used to solve this problem are given in APPENDIX B – EXC 3A – ATTENTUATION OVER FREQUENCY.

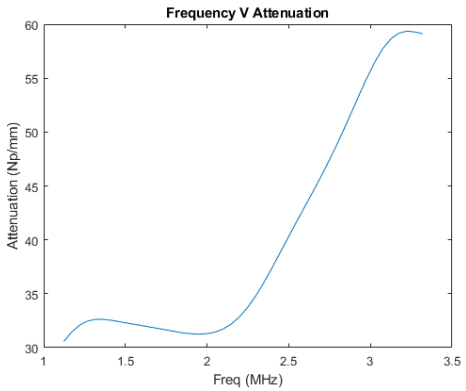


Figure 3 - Attenuation V Frequency

1.2.2 EXERCISE 3B – THEORY & RESULTS - FREQUENCY DEPENDENT REFLECTION COEFFICIENT FROM AN ADHESSIVE JOINT

In addition, to the description given in Section 1.2, a diagram to visually depict the location of each A-Scan is given in Figure 4. Position one shows the immersion pulse echo test being performed over a region with no adhesive bond, and vice versa for position two.

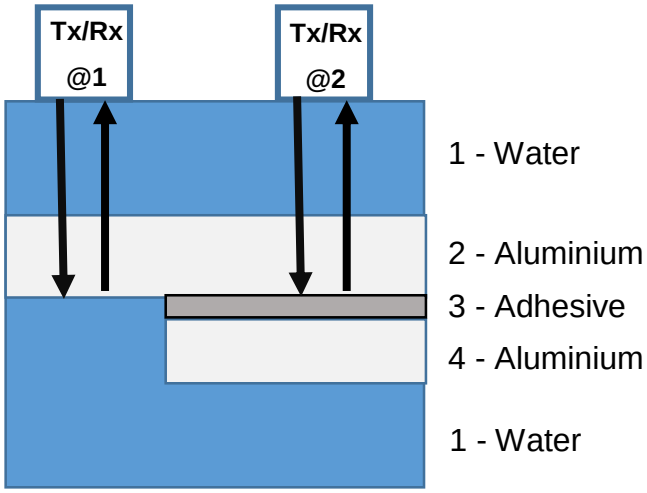


Figure 4 - Experimental Set Up

Moreover, Figure 5 shows the A-scan data given. Clear front and back wall echoes can be seen at approximately 57 µs and 60 µs respectively. As a result no filtering was formed on this data.

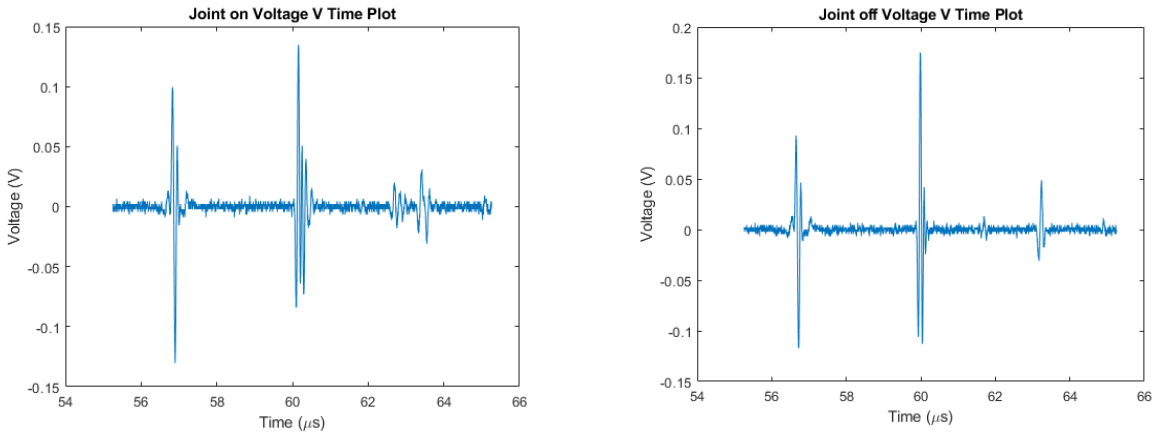


Figure 5 - A-Scan Data Adhesive Layer

By utilising the same wave packet isolation function detailed in Section 1.2.1, the back wall of each signal could isolated and it's frequency spectra compared. Figure 6 graphically shows this and differences in the back wall spectra are observed due to the adhesive layer.

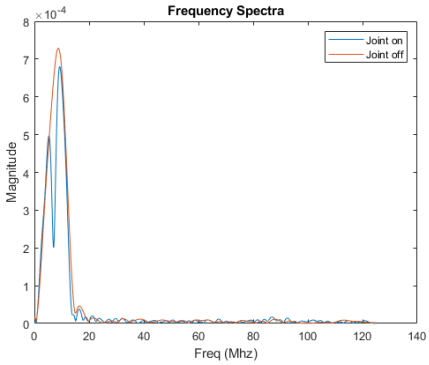


Figure 6 – Back Wall Frequency Spectra Comparison

The back wall reflection at position one and two in Figure 4 can be described by Eq. 3 & 4 respectively. Where I is the intensity of the ultrasonic beam; T_{12} is the dimensionless transmission coefficient from material 1, water, to material 2, aluminium; R_{12} is the dimensionless reflection coefficient from material 1 to material 2; R_{23} is the

dimensionless reflection coefficient from material 3 to material 2; T_{21} is the dimensionless transmission coefficient from material 2 to material 1; α is the attenuation coefficient in Np/m, and d is the thickness of material 2.

$$B(\omega)_{@1} = I(\omega)T_{12}R_{12}T_{21}e^{-2\alpha d} \quad (3)$$

$$B(\omega)_{@2} = I(\omega)T_{12}R_{23}T_{21}e^{-2\alpha d} \quad (4)$$

Eq. 4 can be divided by Eq. 3 to produce Eq. 5 in terms of R_{23}. R_{21} can be calculated via a ratio of acoustic impedances given in Eq. 6. The acoustic impedances are proportional to the materials' density and the speed of sound in the material. R_{21} is therefore assumed to not be a function of frequency.

$$R_{23} = \frac{B_{@2}}{B_{@1}}R_{21} \quad (5)$$

$$R_{21} = \frac{z_1 - z_2}{z_1 + z_2} = \frac{z_{aluminium} - z_{water}}{z_{aluminium} + z_{water}} \quad (6)$$

The absolute value of the frequency spectra of back wall echoes at each position can be used in conjunction with Eq. 6 and allow for Eq. 5 to be solved. The output of which is the reflection coefficient as a function of frequency. This is shown in Figure 7. Details on the MATLAB code used to solve this problem are given in APPENDIX C – EXC 3B – REFLECTION COEFFICIENT OVER FREQUENCY.

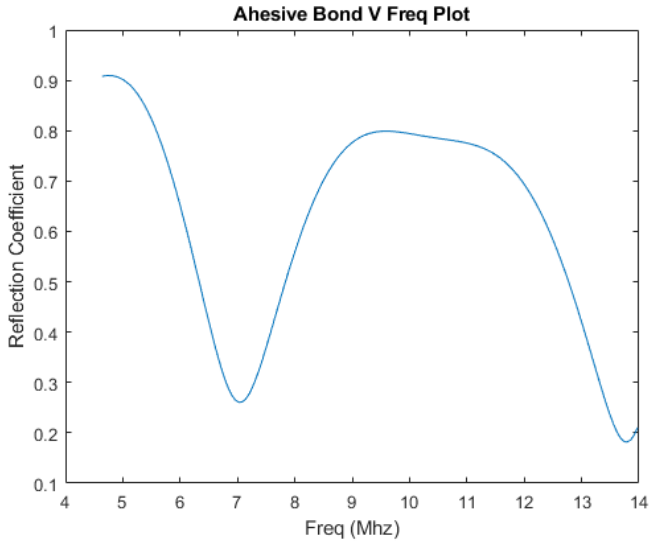
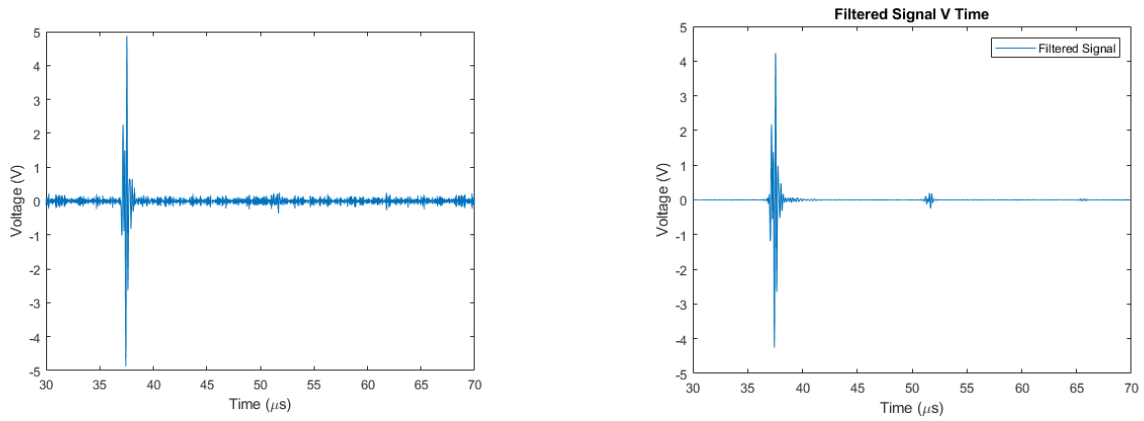


Figure 7 - Frequency V Reflection Coefficient

1.2.3 EXERCISE 3C – THEORY & RESULTS - AUTOMATED TIME-DOMAIN THICKNESS MEASUREMENT

As stated within Section 1.2, a B-Scan containing a series of A-Scans was given. An example of the raw data provided is shown in Figure 8A. As can be seen the data contains noise, and the back wall echo is submerged within the noise floor. As a result, low pass filtering was performed via a MATLAB function. This is detailed in APPENDIX D – FN_FREQLOWPASS – LOW PASS FREQUENCY FILTERING. Like before, no detail over the bandwidth of the 5 MHz transducer is given, so the function assumes it is 6 dB either side of the centre frequency. All filtering is performed in the frequency domain with a low pass Hanning window.

Figure 8B shows the same A-Scan as in Figure 8A, but after filtering has been performed. It can be seen that the back wall is now temporally distinct post filtering.



A B

Figure 8 - Unfiltered and Filtered A-Scan Example

With the signal noise reduced, the signal was time gated to isolate the front and back wall echoes. The front wall echo was assumed to occur from 30-45 µs, and the back wall echo was assumed to occur from 45-60 µs. Once the wave packets were isolated in time, a Hilbert transform was taken to calculate the envelope of the signal [2]. This allowed for the maximum amplitude in the signal to be identified and from this a time difference from the front to the back wall could be calculated. Figure 9 shows the signal envelope, and where the maximum of the front and back wall is located in time.

$$D = c\Delta t \tag{7}$$

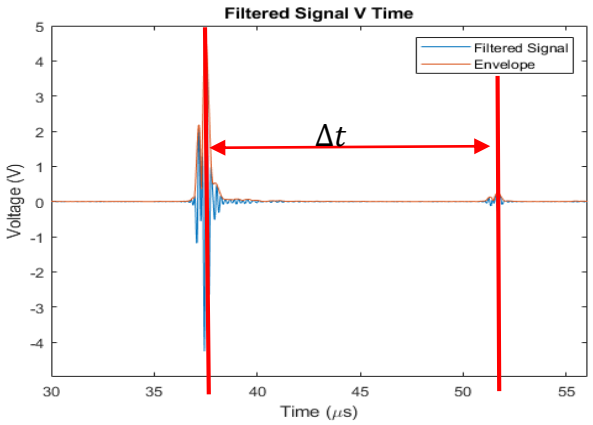


Figure 9 - Signal Envelope V Time

With the time difference known, a thickness from each A-Scan can be implied if the wave speed is known from Eq. 7. To calibrate the wave speed, the maximum thickness of 55 mm and the maximum time difference is utilised. This gives a wave speed of approximately 5950 m/s.

This technique can be applied to all the data given in the B-Scan and produce a total thickness plot for the component. Figure 10 shows the component thickness as a function of transducer position. Figure 10A shows a thickness plot with no averaging applied while Figure 10B shows a thickness plot with a 3 point moving average applied. The moving average was applied to smooth out repeatability errors from the differences in A-Scan data. It is thought that these errors can be attributed to coupling and positioning errors of the transducer itself. The thickness with the transducer at a position of 125 mm from the datum was calculated to be 47.2 mm. Details on the MATLAB code used to solve this problem are given in APPENDIX E – EXC 3C – THICKESS MAPPING.

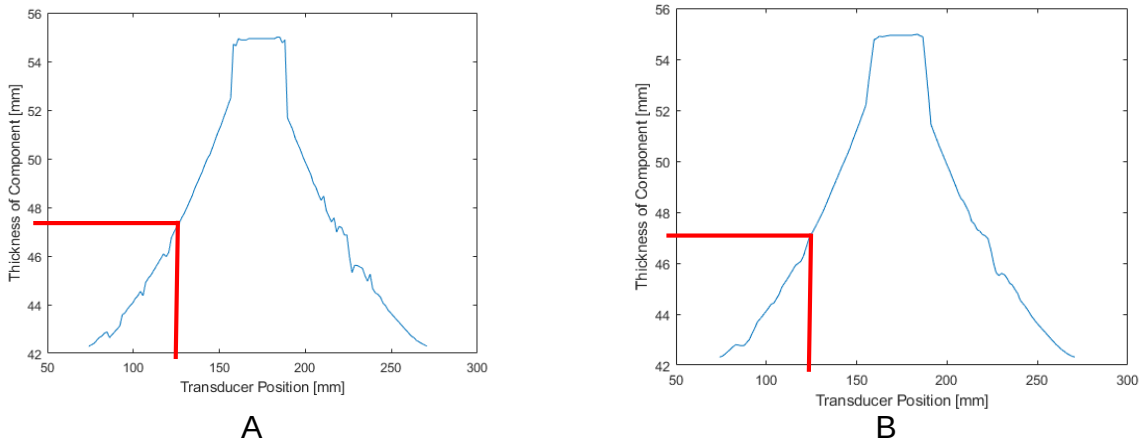


Figure 10 - Thickness Plot of Component

1.4 CONCLUSION

For Exercise 3A, the attenuation as a function of frequency was calculated for a 7.8 mm Perspex block. It was shown that the attenuation linearly increased with frequency around the transducers bandwidth of 1.25-3.25 MHz from 30-60 Np/mm.

For Exercise 3B, the reflection coefficient as a function of frequency was determined. The reflection coefficient was noted to be nonlinear, but the author is not aware of what may cause this non linearity that was observed.

For Exercise 3C, the thickness of the component is measured by looking at signal packets within the voltage-time trace. Experimental errors were noted from A-Scan to A-Scan within the B-Scan, as the thickest point varied. Due to experimental errors, a moving average was performed to accommodate the variance observed.

2 EXERCISE 5 – ARRAY DESIGN

2.1 ABSTRACT

Simulated FMC data was given and a Total Focusing Imaging Method was implemented on the data. For the samples given, all voids/pores were able to be resolved and in addition cracks were able to be sized. Various array parameters were also optimised to give the best cost to imaging performance.

2.2 INTRODUCTION

For exercise 5, an ultrasonic array algorithm that was capable of simulating Full Matrix Capture (FMC) data based on a given number of elements, element pitch, element width & centre frequency for a given sample was given. The sample could be varied over three samples – ‘Test Points’, ‘Test Crack’ and ‘Test Numbers’ – and then randomised to produce a unique sample. In addition, the simulated FMC data could be simulated with and without noise being present.

From the FMC data, an imaging algorithm was to be implemented that utilised optimised array parameters to inspect a region of 40x80mm. The following constraints were placed on the array design: - maximum number of elements: 128; minimum element width: 0.1 mm; maximum element width: 4.0 mm; minimum gap between elements: 0.05 mm, and centre frequency from 1-10 MHz. It was also stated that the -40 dB bandwidth of the array is 112 %, and that the speed of sound in the material was 5000 m/s. Defects such as cracks needed to be sized, whilst pores/voids were only to be resolved.

2.3 IMAGING ALGORITHM – THEORY AND RESULTS - TOTAL FOCUSING METHOD

From the simulated FMC data, a series of A-Scans from the array were given. As previously stated the data could be simulated with or without noise being present. Owing to this, band pass filtering was performed utilising the bandwidth outlined in Section 2.2. This was implemented within a function and details of which are given in APPENDIX G – EXC 5 – FN_FREQ_BANDPASS – BANDPASS FREQUENCY FILTERING .

Various imaging algorithms could be implemented from the FMC data given ranging from Plane B Scan to Total Focusing Method (TFM). The TFM imaging algorithm documented by Holmes et al. [3] was elected to be used as it allows for all points regardless of depth to be synthetically focused on.

Eq. 8 describes how the TFM imaging algorithm is implemented. It involves applying a time delay to each signal by calculating the distance from the transmit element to the imaging point and from the imaging point to the receive element. This is repeated each point and for every array element with the result being summed on each iteration, making it computationally demanding.

$$I(\boldsymbol{r}) = \sum h_{tx,rx} \frac{\left(\sqrt{(x_{tx} - x) + z^2} + \sqrt{(x_{rx} - x) + z^2}\right)}{c} \tag{8}$$

A MATLAB code capable of performing the TFM imaging algorithm was developed for processing the FMC data – see APPENDIX I – EXC 5 – IMAGING for more details. The sample was discretised into a grid of 100x100mm with each point on the image being 0.1 mm apart. Example images of the ‘Test Point’ sample with and without noise are given in Figure 11 & Figure 12. Another image is given in

with filtering applied to data with noise present.

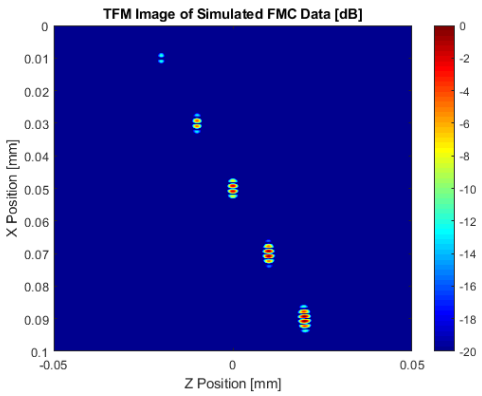


Figure 11 - TFM Image With No Noise

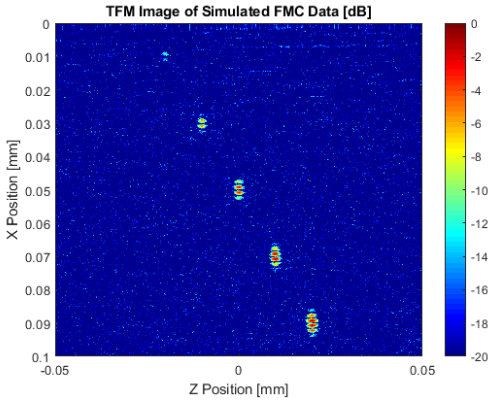


Figure 12 - TFM Image With Noise

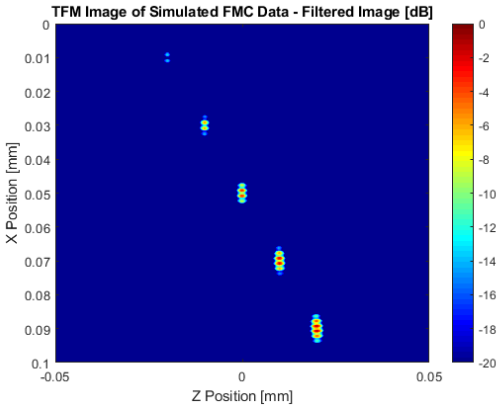


Figure 13 - TFM Image with Noise Post Filtering

As mentioned previously, the simulated FMC data could be performed on the same sample with and without noise being present. From these two cases, the noise present could be found by subtraction of the image without noise from the image with noise. Each element in the noise free image was then divided by each in the noise. The RMS value of this division could then be taken and be used the generate a Signal to Noise Ratio (SNR). This is shown mathematically in Eq. 9 & Eq. 10.

$$FMC_{Noise} = FMC_{TestPoints+Noise} - FMC_{TestPoints} \tag{9}$$

$$SNR = 20\log_{10}\left(RMS\left(\frac{FMC_{TestPoints}}{FMC_{Noise}}\right)\right) \tag{10}$$

For the sample ‘Test Points’ with noise, with and without filtering, for an example transducer, the SNR was 19.88 & -23.9 dB respectively. The increase in SNR showed

that the filtering greatly improved the image quality. The example transducer was assumed to have a centre frequency of 2 MHz, 64 elements with a width of 0.9 mm and a pitch of 1 mm.

For all realistic samples given – 'Test Points' & 'Test Crack'– the TFM code was conducted using optimised array parameters discussed in Section 2.4. As can be seen all defects are distinct and the crack has been sized to be approximately 5 mm in length.

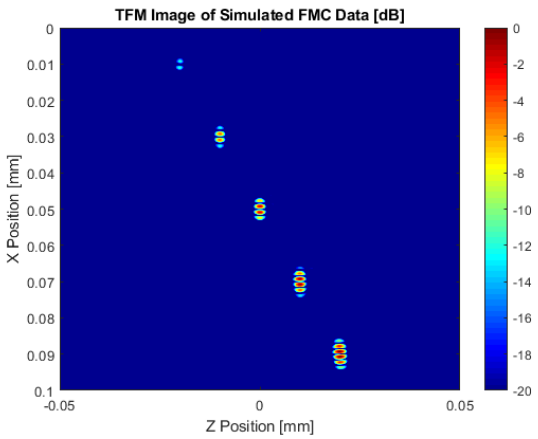


Figure 15 - Idealised Test Points

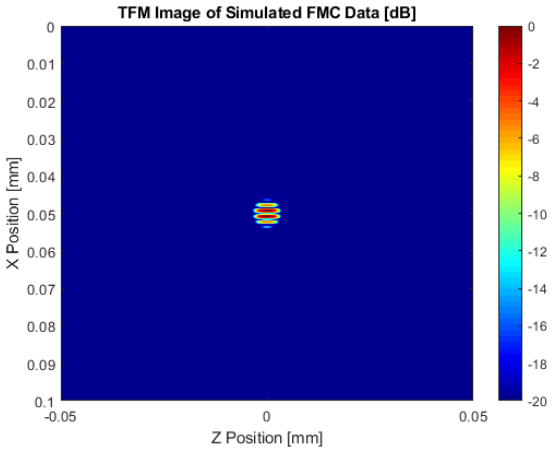


Figure 15 - Idealised Test Crack

2.4 ARRAY OPTIMISATION – THEORY AND RESULTS

For the optimisation of the array, element width, cost of manufacture, SNR and focusing power were considered.

From Huygen's principle of super position, it can be noted that anything under two sources per wavelength does not allow for the beam to be steered effectively. This is shown in Figure 16 & Figure 17. From this, the element width across the 1-10 MHz frequency range could be calculated in accordance with Eq. 11. Where f is the frequency in hertz, c is the wave speed in m/s, and λ is the wavelength in meters. It was further assumed that the minimum gap between elements was ideal, as it gave the widest element for best focusing performance on reception. Using this a pitch for each frequency could be determined in line with Eq. 12.

$$Element\ Width = \frac{\lambda}{2} = \frac{c}{2f} \tag{11}$$

$$Pitch = Element\ Width + Min\ Gap \tag{12}$$

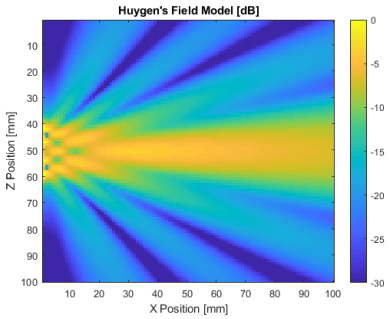


Figure 16 - Huygen's Model with 2 Sources/Wavelength

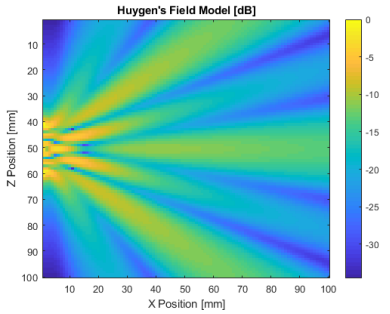


Figure 17 - Huygen's Model with < 2 Sources/Wavelength

With a pitch and element width of the array now calculated as a function frequency, functions of cost, SNR and field intensity vs the number of elements and frequency were implemented.

Figure 19 shows a cost factor based on fictitious data. This assumes that the cost of manufacture increases with the number of elements in a linear fashion, but achieves this in an inverse fashion. The cost factor is based on 8-128 elements with 8 elements being assumed to be the cheapest so has the maximum weighting of one assigned to it. The opposite is true for 128 elements and a weighting of 0.5 is assigned to it.

Likewise, a field intensity factor is shown in Figure 19 for the number of elements and normalised. This was based off Huygen's principle and was performed via a MATLAB function. Details of this function are provided in APPENDIX F – EXC 5 –

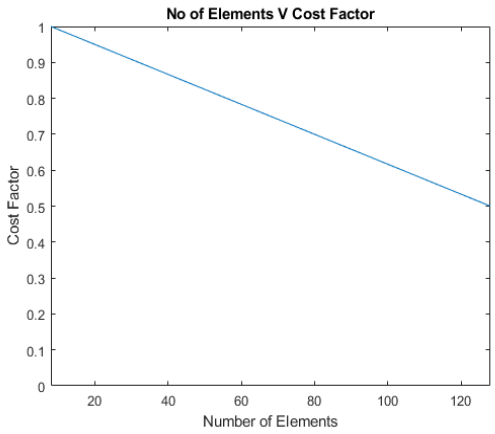


Figure 19 - Cost V Number of Elements

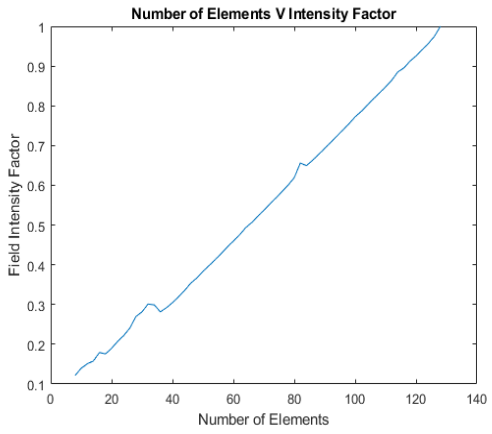


Figure 19 - Field Intensity Factor V Number of Elements

FN_HUYGENS – FIELD INTENSITY FACTOR. The function varies the number of elements and sums the central field intensity.

As described earlier, the SNR could be calculated for standardised images. This allowed for the SNR to calculated for noisy images as a function of frequency. The data produced a highly non-linear curve which could normalised against the maximum SNR. This is shown in Figure 20 and described by Eq. 13.

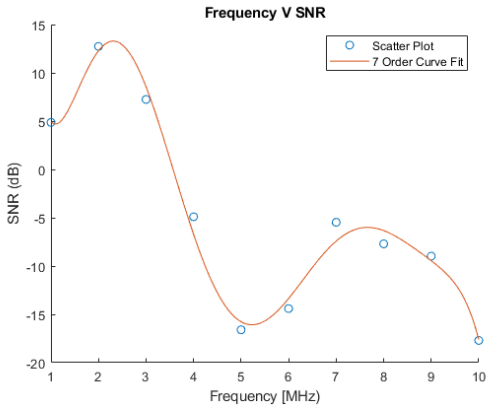


Figure 20 - Frequency V SNR

$$SNR = -0.0033f^7 + 0.13f^6 - 2.2f^5 + 18f^4 - 83f^3 + 1.9x10^2f^2 - 2x10^2f + 83 \quad (13)$$

With the element width and SNR calculated as a function of frequency, and the beam intensity and cost as a function of a number of elements, these normalised parameters could be multiplied to give a performance factor ranging from a minimum value of 0 to 1. This is shown in Eq. 14 & Figure 21.

$$Perforance\ Factor = Element\ Width\ x\ Cost\ Factor\ x\ SNR\ Factor\ x\ Field\ Intensity\ Factor \quad (14)$$

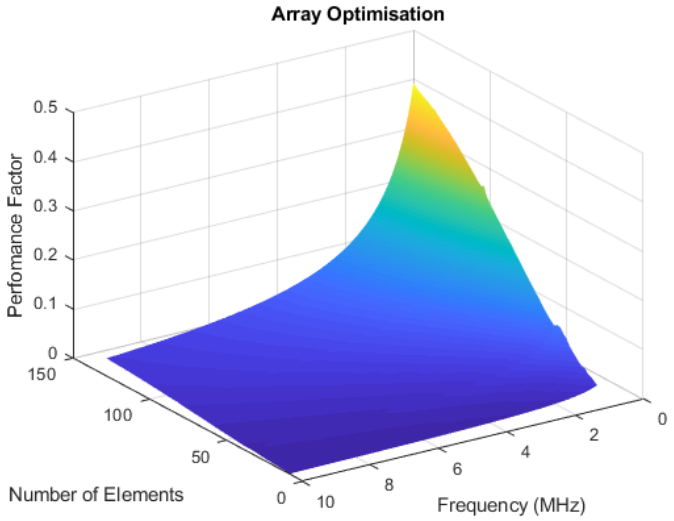


Figure 21 - Array Optimisation

From the optimisation, the following parameters for best image quality were determined:

- No of Elements: 128
- Centre Frequency: 1 MHz
- Element Width: 2.45 mm
- Element Pitch: 2.50 mm

For an image based on a random sample gave the image present in Figure 22. Details of the MATLAB code used to perform the optimisation are given in APPENDIX H – EXC 5 – ARRAY OPTIMISATION.

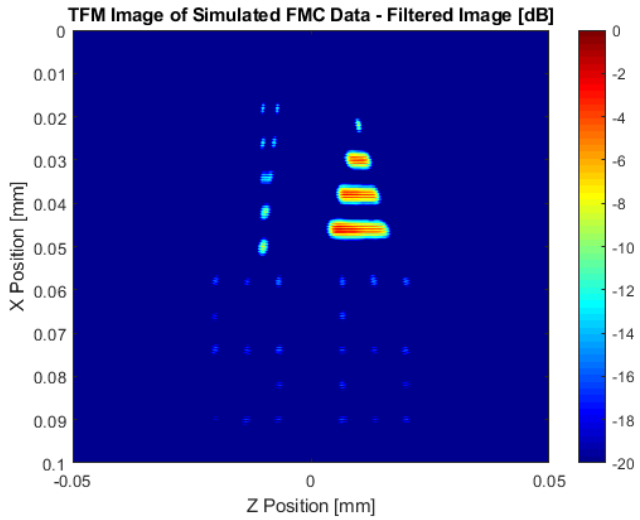


Figure 22 - Randomised Image

2.5 CONCLUSION

A TFM imaging algorithm was implemented on simulated FMC data in line with well published guidelines. Band pass data filtering around the transducers' bandwidth was employed on the FMC data resulting in greatly increased SNR in the images produced (~38 dB). All voids/pores were able to be resolved and in addition cracks were able to be sized.

In conjunction, an optimisation code was also developed to and focused on SNR, Element Width, Cost of Manufacture & Field Intensity. These parameters were then normalised and multiplied together to produce a performance factor. The highest performance factor was assumed to have best array performance. Justification for this is given in Section 2.4. The resulting array with the best performance was given with 128 elements, a centre frequency of 1 MHz and an element with and pitch of 2.45 mm and 2.5 mm. The author acknowledges that the cost of manufacture data is fictitious and this optimisation would be more realistic if appropriate data were used.

3 REFERENCES

[1] "Fast Fourier transform - MATLAB fft - MathWorks United Kingdom." [Online]. Available: https://uk.mathworks.com/help/matlab/ref/fft.html. [Accessed: 14-May-2019].
[2] "Signal envelope - MATLAB envelope - MathWorks United Kingdom." [Online]. Available: https://uk.mathworks.com/help/signal/ref/envelope.html. [Accessed: 19-May-2019].
[3] C. Holmes, B. W. Drinkwater, and P. D. Wilcox, "Post-processing of the full matrix of ultrasonic transmit–receive array data for non-destructive evaluation," *NDT E Int.*, vol. 38, no. 8, pp. 701–711, Dec. 2005.

4 APPENDIX A – FN_WAVEPACKET - WAVE PACKET ISOLATION

```
function  [voltage_wave, time_wave] = fn_wave_packet(voltage, time,
v_threshold)
%USAGE
%   [voltage_wave, time_wave] = fn_wave_packet(voltage, time, v_threshold)
%AUTHOR
%   Euan Foster (2019)
%SUMMARY
%   Identifies and isolates wave packets for a given voltage time trace and
%   noise threshold. The isolated waves are padded with zero values after
%   their voltage and time traces to make good frequency resolution
%OUTPUTS
%   Outputs a voltage and time 2D array that in which each row contains
%   the time and voltage of the wave packet identified
%INPUTS
%   voltage - sampled voltage values
%   time    - sampled time values
%   v_threshold - voltage threshold of noise floor
%NOTES
%This code was to work for 3C also. However, when it gets to column 80 the
%backwall is too faint to be automatically detected by this function.

%If you have any ideas on how to do this I would like to hear :)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[up1,lo1] = envelope(voltage);                              %Hilbert transform
of wave signal
v_binary = up1;                                             %Establishing a
binary wave array that is the same size as the voltage and time array
v_binary(v_binary>=v_threshold) = 1;                        %Populating the
binary wave
v_binary(v_binary<v_threshold) = 0;                         %Populating the
binary wave
v_binary = medfilt1(v_binary,75,'zeropad');                 %Filtering out
binary wave windows of insignificant size
m_binary = diff(v_binary);                                  %Absolute value of
the column wise gradient
m_binary(size(m_binary,1):size(time,1),:) = 0;              %Making m_binary
the same length as voltage/time

[rows, columns] = find(m_binary==1);                        %Identifying where
the gradient change occurs - 2 gradient values at start and end of wave
max_packets = sum(columns(:,:)==mode(columns));             %Calculating the
max number of wave packets obsered in a wave

ind_start = zeros(max_packets,size(voltage,2));             %Establishing an
array to store all the start indexes of the wave packets
ind_end = zeros(max_packets,size(voltage,2));               %Establishing an
array to store all the end of the wave packets

%Populating the ind_start & ind_end
for ii = 1:size(voltage,2)
    x = find(m_binary(:,ii)==1);
    y = find(m_binary(:,ii)==-1);
    lenx = length(x);
    leny = length(y);
    ind_start(1:lenx,ii) = x;
```

```
    ind_end(1:leny,ii) = y;
end

fft_len = 2^nextpow2(size(time,1));
%Establishing a large number
voltage_wave = zeros(fft_len,max_packets,size(voltage,2));
%Setting a 2D voltage array at all points to zero
time_wave =zeros(fft_len,max_packets,size(voltage,2));
%Setting a 2D time array at all points to zero

%Populating voltage and time array for each wave packet identified
for ii = 1:size(voltage,2)
    for jj = 1:max_packets

        if ind_start(jj,ii)~= 0 && ind_end(jj,ii)~= 0
            vol = voltage(ind_start(jj,ii):ind_end(jj,ii),ii);
            len = length(vol);

            voltage_wave(1:len,jj,ii) =
voltage(ind_start(jj,ii):ind_end(jj,ii),ii);
            time_wave(1:len,jj,ii) =
time(ind_start(jj,ii):ind_end(jj,ii),1);
        end
    end
end


end
```

5 APPENDIX B – EXC 3A – ATTENTUATION OVER FREQUENCY

```
%% Initialising and loading in data

clear; %clear all variables from memory
close all; %close all windows
clc; %clear command window

disp('Frequency dependent attenuation of perspex'); %display the title

%file name
fname = '7_8mm_thick_perspex.mat';
sample_thickness = 7.8e-3;

%load file
load(fname);

%% Plotting A-Scan Data
%Plotting the data
figure(01)
plot(time * 1e6, voltage);
xlabel('Time (\mus)');
ylabel('Voltage (V)');
title('A-Scan - Perspex 7.8mm')
%% Automatically Detecting Each Wave Packet within the A-Scan
%PROGRAM
v_threshold = 1.5e-3;                  %Setting a noise floor threshold
[voltage_wave, time_wave] = fn_wave_packet(voltage,time,v_threshold);
%isolating the waves
%Transposing matrixes from column to row vectors for FFT
voltage_wave = voltage_wave.';
time_wave = time_wave.';
%calculating the sampling frequency
sampling_freq = 1/(time(2)- time(1));

%% Calculating Frequency Content of Wave
%FFT of isolated waves
n = length(voltage_wave);
f = sampling_freq*(0:(n/2))/n;
Y = fft(voltage_wave,n,2);
P = abs(Y/n);

%% Plotting frequency Spectra of Each Wave Packet
%Plotting Spectra of wave packets over half the calculated FFT
figure(02)
plot(f,P(:,1:n/2+1))
xlabel('Freq (Hz)');
ylabel('Magnitude');
title('Frequency Spectra of Front and Back Wall')
legend('Front Wall','Backwall','Reverberation')

%% Calculating attenuation of as a function of Frequency and Distance
%Calculating attentuation over full frequency range
d = 2*sample_thickness;
A_omega = P(2,:)./P(1,:);
alpha = (log(A_omega)*-1)/d;
```

```
%Truncating attenuation for frequency range of interest
%Assumes a 12db drop in first echo
%This is equivalent to a full width at half maximum technique
i = find(P(1,:) == max(P(1,:)),1,'first');
freq_mag_drop = f(1,i)/(10^(6/20));
j = find(f>=f(1,i)-freq_mag_drop,1,'first');
k = find(f>=f(1,i)+freq_mag_drop,1,'first');

%Plotting attenuation as a function of relevant frequency content
figure(03)
plot(f(1,j:k)/1e6,alpha(1,j:k))
xlabel('Freq (MHz)');
ylabel('Attenuation (Np/mm)');
title('Frequency V Attenuation')
```

6 APPENDIX C – EXC 3B – REFLECTION COEFFICIENT OVER FREQUENCY

```
%% Initialising and loading in data for Joint off Case

clear; %clear all variables from memory
close all; %close all windows
clc; %clear command window

disp('FREQUENCY-DEPENDENT REFLECTION COEFFICIENT FROM AN ADHESIVE JOINT');
%display the title

%file name & loading file
fname = 'joint_off_adhesive.mat';
load(fname);

%% Plotting A-Scan of Joint off Case
%Plotting the data for no joint
figure(01)
plot(time * 1e6, voltage);
title('Joint off Voltage V Time Plot')
xlabel('Time (\mus)');
ylabel('Voltage (V)');

%% Isolating Each Wave Packet in the A-Scan
v_threshold = 1e-2;                 %Setting a noise floor threshold

[voltage_wave, time_wave] = fn_wave_packet(voltage,time,v_threshold);
%isolating the waves
%Transposing matrixes from column to row vectors for FFT
voltage_wave = voltage_wave.';
time_wave = time_wave.';

%Storing joint off wave and time arrays
voltage_wave_jointoff = voltage_wave;
time_wave_jointoff = time_wave;
sampling_freq_jointoff = 1/(time(2)-time(1));

%% Loading in data for joint on case
%file name & loading file
fname = 'joint_on_adhesive.mat';
load(fname);

%% Plotting A-Scan of Joint on Case
[voltage_wave, time_wave] = fn_wave_packet(voltage,time,v_threshold);
%isolating the waves
%Transposing matrixes from column to row vectors for FFT
voltage_wave = voltage_wave.';
time_wave = time_wave.';

%storing joint on wave and time arrays
voltage_wave_jointon = voltage_wave;
time_wave_jointon = time_wave;
sampling_freq_jointon = 1/(time(2)-time(1));
```

```
%% Plotting A-Scan of Joint on Case
figure(02)
plot(time * 1e6, voltage);
title('Joint on Voltage V Time Plot')
xlabel('Time (\mus)');
ylabel('Voltage (V)');

%% Calculating FFT of Isolated Waves for Both Cases
%FFT of isolated waves
%Joint off
n = length(voltage_wave_jointoff);
f_jointoff = sampling_freq_jointoff*(0:(n/2))/n;
Y_jointoff = fft(voltage_wave_jointoff,n,2);
P_jointoff = abs(Y_jointoff/n);
P_jointoff_B = P_jointoff(2,1:n/2+1);           %Truncating FFT to half
length and only storing backwall

%Joint on
n = length(voltage_wave_jointon);
f_jointon = sampling_freq_jointon*(0:(n/2))/n;
Y_jointon = fft(voltage_wave_jointon,n,2);
P_jointon = abs(Y_jointon/n);
P_jointon_B = P_jointon(2,1:n/2+1);

%% Plotting Frequency Spectra of Both Cases
figure(03)
plot(f_jointon / 1e6,P_jointon_B,f_jointon / 1e6, P_jointoff_B);
title('Frequency Spectra')
xlabel('Freq (Mhz)');
ylabel('Magnitude');
legend('Joint on','Joint off')

%% Truncating Frequency Spectra to region of Interest
%Assumes a 12db drop in first echo
%This is equivalent to a full width at half maximum technique
i = find(P_jointon_B(1,:) == max(P_jointon_B(1,:)),1,'first');
freq_mag_drop = f_jointon(1,i)/(10^(6/20));
j = find(f_jointon>=f_jointon(1,i)-freq_mag_drop,1,'first');
k = find(f_jointon>=f_jointon(1,i)+freq_mag_drop,1,'first');
P_jointon_B = P_jointon_B(1,j:k);
P_jointoff_B = P_jointoff_B(1,j:k);

%% Calculating Reflection Coefficient as a Fucntion of Frequency
%Taken from Course Notes
%Density of Materials
rho_alu = 2700;
rho_water = 1000;
%Longitudinal Velocity
v_alu = 6320;
v_water = 1500;

%Acoustic Impedance
z_alu = rho_alu*v_alu;
z_water = rho_water*v_water;

ref_alu_water = (z_alu-z_water)/(z_alu+z_water);
```

```
ref_alu_adhesive = (P_jointon_B./P_jointoff_B)*ref_alu_water;

figure(04)
plot(f_jointon(1,j:k) / 1e6, ref_alu_adhesive);
title('Ahesive Bond V Freq Plot')
xlabel('Freq (Mhz)');
ylabel('Reflection Coefficient');
```

7 APPENDIX D – FN_FREQLOWPASS – LOW PASS FREQUENCY FILTERING

```
function  [filtered_voltage] = fn_freq_lowpassfilter(voltage, time,
f_centre)
%USAGE
%   [filtered_voltage] = fn_freq_lowpassfilter(voltage, time, f_centre)
%AUTHOR
%   Euan Foster (2019)
%SUMMARY
%   Performs a low pass filter on signal 6dB about the centre frquency of
%   the transducer. Returns a filtered signal in the time domain
%OUTPUTS
%   Outputs a filtered signal in the time domain with only the low
%   frequencies of interest present at the same sample length as the
%   original signal
%INPUTS
%   voltage - sampled voltage values
%   time    - sampled time values
%   f_centre - centre frequency of transducer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%performing the FFT of the voltage array.
n = 2^nextpow2(size(voltage,1));
Y = fft(voltage,n,1);
f = 1/(time(2)-time(1))*(0:(n/2-1))/n;
f = f';
P = abs(Y/n);
P = P(1:n/2,:);

%Calulating a window to filter the data with a low pass filter
%Can easily be adjusted for other windowing fucntions
%Assumes a 6 dB drop about the centre frequency
freq_mag_drop = f_centre/(10^(6/20));
j = find(f>=f_centre,1,'first');
k = find(f>=f_centre+freq_mag_drop,1,'first');
window = fn_hanning_lo_pass(n/2,j/(n/2),k/(n/2));

%checks
%plotting half of the first column of the abs spectra voltage values
% figure(03)
% clf
% yyaxis left
% plot(f,P(:,1))
% yyaxis right
% plot(f,window)
% legend('Freq Magnitude','Window');
% title('Half Frequency Spectra and Window');

%Doubling the window to match mirroring effect of the fft
window =[window; flip(window)];
%caulculating the filtered singal over the full spectra content
filtered_spectra = Y.* window;

%checks
%plotting the full first column of the abs voltage spectra and window
```

```
% figure(03)
% clf
% yyaxis left
% plot(abs(Y(:,1)/n));
% hold on
% yyaxis right
% plot(window);
% legend('Freq Magnitude','Window');
% title('Full Frequency Spectra and Window');

%checks
%plotting the full first column of the abs filtered vthe moltage spectra
and window
% figure(04)
% clf
% yyaxis left
% plot(abs(filtered_spectra(:,1)/n));
% hold on
% yyaxis right
% hold on
% plot(window);
% title('Filtered Frequency Spectra and Window');
% legend('Freq Magnitude','Window');

%converting back to the time domain on full Frequency Spectra
filtered_voltage = real(ifft(filtered_spectra,n));
filtered_voltage = filtered_voltage(1:length(time),:);
end
```

8 APPENDIX E – EXC 3C – THICKESS MAPPING

```
%% Initialising and loading in data

clear; %clear all variables from memory
close all; %close all windows
clc; %clear command window

disp('AUTOMATED TIME-DOMAIN THICKNESS MEASUREMENT'); %display the title

fname = 'bearing_casing_bscan.mat';
load(fname);
max_thickness = 55e-3;

%% Plotting first column of data/first A scan
%plotting first column of data
figure(01)
plot(time*1e6,voltage(:,1))
xlabel('Time (\mus)');
ylabel('Voltage (V)');

%% Filtering data based on centre frequency of Transducer
%Asking user for Centre Frequency of Transducer
prompt = {'What is the centre frequency of the transducer used in Mhz?'};
title = 'Frequency Input';
dims = [1 35];
definput = {'5'};
response = inputdlg(prompt,title,dims,definput);
f_centre = str2double(response(1,1))*1e6;
clear title

%Low pass filtering of original signal
filtered_voltage = fn_freq_lowpassfilter(voltage, time, f_centre);

%% Plotting First Column of filtered data
%plotting the first column of the filtered voltage signal in time domain
figure(02)
plot(time*1e6,filtered_voltage(:,1))
xlabel('Time (\mus)');
ylabel('Voltage (V)');
title('Filtered Signal V Time');
legend('Filtered Signal');

%% Automatic Front Wall and Backwall Detection
%Attempted tp use the fn_wave_packet function to auto detect front and
%backwall. This worked till column 80 of the voltage array and then the
%back wall was too faint to detect.

%If you ideas on how to do this, please let me know: e.foster@strath.ac.uk

%function will run but wont return backwalls for columns 80-95ish
%v_threshold =5e-3;
%[voltage_wave, time_wave] =
fn_wave_packet(filtered_voltage,time,v_threshold); %isolating the waves

%% Time Gating Signal about 45-60 micro seconds & Rectifying Signals
```

```
x = find(time==45e-6);
y = find(time==60e-6);

voltage_frontwall = filtered_voltage(1:x,:);
voltage_backwall = filtered_voltage(x+1:y,:);
time_frontwall = time(1:x,:);
time_backwall = time(x+1:y,:);

voltage_frontwall = abs(voltage_frontwall);
voltage_backwall = abs(voltage_backwall);

%% Calulating Envelope & Time of Flight between wavepackets
[up1,lo1] = envelope(voltage_frontwall);
[up2,lo2] = envelope(voltage_backwall);

time_peak_frontwall = zeros(size(voltage,2),1);
time_peak_backwall= zeros(size(voltage,2),1);

for ii = 1:size(voltage,2)

    x = find(voltage_frontwall(:,ii)==max(voltage_frontwall(:,ii)));
    y = find(voltage_backwall(:,ii)==max(voltage_backwall(:,ii)));

    time_peak_frontwall(ii,1) = time_frontwall(x,1);
    time_peak_backwall(ii,1) = time_backwall(y,1);
end

%% Calibrating and calculating thickness

delta_t = time_peak_backwall - time_peak_frontwall;
delta_t = delta_t./2;
speed = max_thickness/max(delta_t);

thickness = speed.*delta_t;
thickness = movmean(thickness,3);

figure(03)
plot(pos*1e3,thickness*1e3);
xlabel('Transducer Position [mm]')
ylabel('Thickness of Component [mm]')
```

9 APPENDIX F – EXC 5 – FN_HUYGENS – FIELD INTENSITY FACTOR

```
function central_field_intensity = fn_huygens(no_elements)
%USAGE
%   central_field_intensity = fn_huygens(no_elements);
%SUMMARY
%   Produces a huygens field and returns a value that is summed 10mm about
%   the centre of the field
%AUTHOR
%   Euan Foster (2019)
%INPUTS
%   no_elements - number of elements
%OUTPUT
%   central_field_intensity - the amplitude of the field summed 10mm about
%   the centre of the field

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%wave parameters
velocity = 5e3;
frequency = 2e6;
lambda = velocity/frequency;

%transducer details
pitch = lambda/2;
transducer_width = 64*pitch + pitch;
source_x_positions = linspace(-transducer_width/2
,transducer_width/2,no_elements);

grid_size = round(transducer_width/10e-3)*10e-3 + 100e-3;
grid_pts = grid_size*1000;

%set up output grid
x = linspace(-grid_size/2, grid_size/2, grid_pts);
y = linspace(0,grid_size,grid_pts);

%set up sources for transducer

%prepare output matrix
p = zeros(length(y),length(x)); %TO BE ENTERED

[A,B] = meshgrid(x,y);
c = cat(2,A',B');
grid_coor = reshape(c,[],2);

transducer_coor = zeros(length(source_x_positions),2);
transducer_coor(:,1) = source_x_positions;

r = pdist2(transducer_coor(:,:), grid_coor(:,:));
r_value = zeros(length(y),length(x),length(source_x_positions));

for ii = 1:length(source_x_positions)
   r_value(:,:,ii) = reshape(r(ii,:), [length(y),length(x)]);
end
```

```
k = 2*pi/lambda;

p = (1/sqrt(r_value)).*exp(1i*k*r_value);
p = sum(p,3);
p = abs(p);

% plot field
% figure()
% clf
% imagesc(y,x,p)
% title('Ultrasonic Field Intensity from Huygens Principle');
% caxis ([ 0 90 ])

%Calculating central field intensity
ii = size(p,1)/2;
jj = ii - 5;
kk = ii + 5;
central_field_intensity = sum(sum(p(jj:kk,:)));

end
```

10 APPENDIX G – EXC 5 – FN_FREQ_BANDPASS – BANDPASS FREQUENCY FILTERING

```
function  [filtered_voltage] = fn_freq_bandpassfilter(voltage, time,
f_centre)
%USAGE
%   [filtered_voltage] = fn_freq_bandpassfilter(voltage, time, f_centre)
%AUTHOR
%   Euan Foster (2019)
%SUMMARY
%   Performs a bandpass pass filter on signal 6dB about the centre frquency
of
%   the transducer. Returns a filtered signal in the time domain
%OUTPUTS
%   Outputs a filtered signal in the time domain with only the bandpass
%   frequencies of interest present at the same sample length as the
%   original signal
%INPUTS
%   voltage - sampled voltage values
%   time    - sampled time values
%   f_centre - centre frequency of transducer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%performing the FFT of the voltage array.
n = 2^nextpow2(size(voltage,1));
Y = fft(voltage,n,1);
f = 1/(time(2)-time(1))*(0:(n/2-1))/n;
f = f';
P = abs(Y/n);
P = P(1:n/2,:);

%Calulating a window to filter the data with a band pass filter
j = find(f>=f_centre,1,'first');
k = find(f>=f_centre*1.56,1,'first');
l = find(f>=f_centre*0.44,1,'first');
window = fn_hanning_band_pass(n/2,1/(n/2),j/(n/2),j/(n/2),k/(n/2));

%checks
%plotting half of the first column of the abs spectra voltage values
% figure(03)
% clf
% yyaxis left
% plot(f,P(:,1))
% yyaxis right
% plot(f,window)
% legend('Freq Magnitude','Window');
% title('Half Frequency Spectra and Window');

%Doubling the window to match mirroring effect of the fft
window =[window; flip(window)];
%caulculating the filtered singal over the full spectra content
filtered_spectra = Y.* window;

%checks
%plotting the full first column of the abs voltage spectra and window
% figure(03)
```

```
% clf
% yyaxis left
% plot(abs(Y(:,1)/n));
% hold on
% yyaxis right
% plot(window);
% legend('Freq Magnitude','Window');
% title('Full Frequency Spectra and Window');

%checks
%plotting the full first column of the abs filtered voltage spectra and
window
% figure(04)
% clf
% yyaxis left
% plot(abs(filtered_spectra(:,1)/n));
% hold on
% yyaxis right
% hold on
% plot(window);
% title('Filtered Frequency Spectra and Window');
% legend('Freq Magnitude','Window');

%converting back to the time domain on full Frequency Spectra
filtered_voltage = real(ifft(filtered_spectra,n));
filtered_voltage = filtered_voltage(1:length(time),:);
end
```

11 APPENDIX H – EXC 5 – ARRAY OPTIMISATION

```
close all
clear all

%% Assigning a no elements v cost to assign a performance factor
X = [8 128];
Y1 = [0.5 1];
Y2 = [1,0.5];

m1 = (Y1(2)-Y1(1))/(X(2)-X(1));
m2 = (Y2(2)-Y2(1))/(X(2)-X(1));

c1 = Y1(1) - (m1*X(1));
c2 = Y2(1) - (m2*X(1));

x = 8:2:128;
y1 = m1.*x + c1;
y2 = m2.*x + c2;

figure
plot(x,y1)
xlim([8 128])
ylim([0 1])
xlabel('Number of Elements')
ylabel('Assumed Normalised Cost of Manufacture')
title('No of Elements V Cost of Manufacture')

figure
plot(x,y2)
xlim([8 128])
ylim([0 1])
xlabel('Number of Elements')
ylabel('Cost Factor')
title('No of Elements V Cost Factor')

%% Calculating a no of elements v field intensity function to assign a
performance factor
elements = 8:2:128;
field_intensity = zeros(length(elements),1);

for ii = 1:length(elements)
   field_intensity(ii) = fn_huygens(elements(ii));
end

%Transposing and normalising the data
field_intensity = field_intensity';
field_intensity = field_intensity./max(field_intensity);

figure
plot(elements,field_intensity)
xlabel('Number of Elements')
ylabel('Field Intensity Factor')
title('Number of Elements V Intensity Factor')

%% Specifying a frequency V SNR function to assing a perfomance factor
```

```
frequency = [1 2 3 4 5 6 7 8 9 10];
SNR = [4.87 12.74 7.25 -4.9 -16.6 -14.4 -5.47 -7.7 -8.97 -17.7];

figure
scatter(frequency,SNR)
xlabel('Frequency [MHz)')
ylabel('SNR (dB)')
title('Frequency V SNR')
legend('Scatter Plot','7 Order Curve Fit')



%% Refining parameters so that only array values are returns within the
contrains on the question
%Defining Parameters Given in Question
velocity = 5000;
min_gap = 0.05e-3;
min_element_width = 0.1e-3;
max_element_width = 4e-3;

%Calculating Pitch Range
%Assumes you want widest element for best focusing so min_gap = gap
min_pitch = min_element_width+min_gap;
max_pitch = max_element_width+min_gap;

%Calculating Wavelength Range
%From Huygens having a pitch of wavelength/2 is a good assumption
min_lambda = min_pitch*2;
max_lambda = max_pitch*2;

%Calculating Frequency Range
max_frequency = velocity/min_lambda;
min_frequency = velocity/max_lambda;

%Checking if Frequency Range is within question Parameters
if max_frequency > 10e6
    max_frequency = 10e6;
end
if min_frequency < 1e6
    min_frequency = 1e6;
end

%Refining Frequency range to multiples of 0.05Mhz
freq_step = 0.05e6;
max_frequency = round(max_frequency/freq_step)*freq_step;
min_frequency = round(min_frequency/freq_step)*freq_step;

%% Calculating the element width in the array
%assumes that the pitch is Lambda/2

%Calculating element width over applicable freq range
frequency = min_frequency:freq_step:max_frequency;
frequency = frequency';
lambda = velocity./frequency;
pitch = lambda./2;
frequency = frequency/1e6;
```

```
SNR = (-0.0033.*frequency.^7) + (0.13.*frequency.^6)...
    - (2.2.*frequency.^5) + (18.*frequency.^4) - (83.*frequency.^3)...
    + (1.9e2.*frequency.^2) + (2e2.*frequency) + 83;
SNR = SNR/max(SNR);

figure
plot(frequency,SNR)
xlabel('Frequency (MHz)')
ylabel('Normalised SNR')
title('Frequency v Normalised SNR')

%Converting to mm
pitch = pitch*1e3;
min_gap = min_gap*1e3;
element_width = pitch-min_gap;

elements = 8:2:128;

performance_factor = zeros(length(element_width),length(elements));
element_width_factor = element_width./max(element_width);

for ii = 1:length(elements)
    performance_factor(:,ii) =
element_width_factor.*y2(ii)*field_intensity(ii)*SNR(ii);
end

surf(elements,frequency,performance_factor)
xlabel('Number of Elements')
ylabel('Frequency (MHz)')
zlabel('Perfomance Factor')
title('Array Optimisation')

[row,col] = find(performance_factor == max(max(performance_factor)));

fprintf('The optimum array performance is given with %d elements, with a
centre frequency of %e MHz, an element width of %f mm and a gap of %g
mm\n', elements(1,col), frequency(row,1),element_width(row,1),min_gap);
```

12 APPENDIX I – EXC 5 – IMAGING

```
clear; close all; clc;


%% Calculating pseudo FMC data

%In this exercise, you will use a pre-written function,
fn_simulate_data_v2,
%to synthesise the FMC data from a specified ultrasonic array when
%looking at various samples.

%Your task is to specify an appropriate array, centre frequency and write
%the necessary code to convert the FMC data into an image. The
%ultimate goal is to obtain and analyse the image when
%fn_simulate_data is used with code = 'XXX' where 'XXX' is your initials.
%The sample in this case contains a number of cracks and point reflectors
%in various patterns, including 2 digits.

%When you formed an image of this sample, you should consider questions
%such as:
% - Can you resolve the most closely-spaced point reflectors? How do you
% assess this?
% - Can you estimate the lengths of the cracks from the image? If so, how?
% - Can you tell what digits are present?
% - What is the smallest number of elements needed in an array to obtain
% adequate images?

%The region of interest in all samples is from x = -20 to 20 mm and from
%z = 10 to 90 mm, but you will probably want to create a slightly larger
%image. The speed of sound in the target is 5000 m/s.

%The parameters 'no_elements', 'element_pitch', 'el_width' and
%'centre_freq' describe the array and should be in SI units (i.e. m, Hz).
%The array has to satisfy the following manufacturing limits:
% - maximum number of elements: 128
% - minimum element width: 0.1 mm
% - maximum element width: 4.0 mm
% - minimum gap between elements: 0.05 mm (i.e. element_pitch - el_width >=
%   0.05mm).
% - centre frequency must be in the range 1 to 10 MHz

%For the 'sample' parameter use either:
% - 'TEST POINTS' to return the FMC data from a sample with 5 point targets
%   across the region of interest with no noise present
% - 'TEST CRACK' to return the FMC data from a sample with a 5mm long crack
%   in the centre of the region of interest with no noise present
% - 'TEST NUMBERS' to return the FMC data for a sample containing point
%   targets representing the digits 1-9 in the region of interest with no
%   noise present
% - 'XXX' where 'XXX' is your initials to return the FMC data for the
%   blind trial sample containing various cracks, point reflectors and
noise.
%   Adding '+NOISE' to any of the first three sample strings (e.g.
%   'TEST POINTS+NOISE') will return the same FMC data with additional
%   noise at the same level as the noise for the blind trial sample.
```

```
%Your chosen array parameters go here:
no_elements = 64;
element_pitch = 1e-3;
element_width = 0.9e-3;
centre_freq = 2e6;

%set code to the appropriate code for the sample for which you wish to
simulate data

%Determining what initials the user has
prompt = {'What are your initials?'};
title = 'Initials?';
dims = [1 35];
definput = {'EAF'};
response = inputdlg(prompt,title,dims,definput);
clear title

%Determining what sample you have
prompt = {'What sample would you like to simulate?'};
title = 'Sample?';
list = {'TEST POINTS','TEST CRACK','TEST NUMBERS',...
'TEST POINTS+NOISE','TEST CRACK+NOISE','TEST NUMBERS+NOISE'};
list(1,end+1) = response;
[indx,tf] = listdlg('ListString',list,'PromptString',prompt,'name',title);
clear title

sample = char(list(1,indx));

%Call the encrypted function to simulate the FMC data
[time, time_data, element_positions] = fn_simulate_data(sample,
no_elements, element_pitch, element_width, centre_freq);

%The fn_simulate_data function returns the following parameters:
% - 'time' is an m-by-1 column vector representing the time-axis of every
%   A-scan in the FMC data
% - 'time_data' is a m-by-no_elements-by-no_elements 3D array containing
%   the FMC data itself. The dimensions represent time, transmitter number
%   and receiver number. Therefore time_data(:, 7, 3)) is the A-scan for
%   transmitter element 7 and receiver element 3.
% - 'element_positions' is a 1-by-no_elements row-vector of array element
%   x-coordinates, centred on x = 0. Therefore in the example case of
%   time_data(:, 7, 3), the transmitting element is at element_positions(7)
%   and the receiving element is at element_positions(3).

%Example of a simulated A-scan from the FMC data
figure(01);
transmitter_index = 7; %this is just an arbitrary choice as an example
receiver_index = 3; %this is just an arbitrary choice as an example
plot(time, time_data(:, transmitter_index, receiver_index));
title(sprintf('Time signal for transmitter %i to receiver %i',
transmitter_index, receiver_index));
xlabel('Time (s)');

%Now write your own code to convert this data into an image ...
%% Filtering the data
```

```
%Determining if the user wishes the data to be filtered
prompt = {'Would you like the FMC data to be filtered? (1=Yes, 0=No)'};
title = 'Filtering?';
dims = [1 35];
definput = {'1'};
response = inputdlg(prompt,title,dims,definput);
filtering = str2double(response);
clear title

%Deterimining the sample contains noise
Substring = 'NOISE';
Substring2 = char(list(1,length(list)));
noise_present = ~contains(sample, Substring);
initials_present = ~contains(sample, Substring2);

%Perfoming filtering if required
if noise_present == 0 && filtering == 1 || initials_present == 0 &&
filtering == 1
    filtered_voltage =
zeros(size(time_data,1),size(time_data,2),size(time_data,3));

    %Bandpass Filtering of original signal
    for ii = 1:size(time_data,3)
        filtered_voltage(:,:,ii) =
fn_freq_bandpassfilter(time_data(:,:,ii), time, centre_freq);
    end

    time_data = filtered_voltage;

    %Example of filtered data
    figure(02);
    transmitter_index = 7; %this is just an arbitrary choice as an example
    receiver_index = 3; %this is just an arbitrary choice as an example
    plot(time, time_data(:, transmitter_index, receiver_index));
    title(sprintf('Filtered Time signal for transmitter %i to receiver %i',
transmitter_index, receiver_index));
    xlabel('Time (s)');
end

%% Defining Waveproperties
velocity = 5e3;
wavelength = velocity/centre_freq;
sampling_freq = 1/(time(2)-time(1));

%% Defining Grid Spatial Properties
grid_size = 100e-3;
grid_pts = 1000;

x = linspace(-grid_size/2, grid_size/2, grid_pts);
z = linspace(0,grid_size,grid_pts);
[X,Z]=meshgrid(x,z);

%% Calculating TFM data
%Initialising Arrays
distance = zeros(length(z),length(x));
```

```
distance_T=zeros(length(z),length(x));
distance_R=zeros(length(z),length(x));
image = zeros(length(z),length(x));

%Determining what GPU the user has
prompt = {'Does your PC have a Nvidia GPU (1=Yes, 0=No)?'};
title = 'GPU Type?';
dims = [1 35];
definput = {'0'};
response = inputdlg(prompt,title,dims,definput);
pretend_cuda = str2double(response(1,1));
clear title

%Converting arrays to the GPU if applicable
if pretend_cuda == 1
    time_data = gpuArray(time_data);
    Z = gpuArray(Z);
    X = gpuArray(X);
    distance = gpuArray(distance);
    distance_T=gpuArray(distance_T);
    distance_R=gpuArray(distance_R);
    image = gpuArray(image);
end

%Initialising loop variables
time_taken = 0;
num_print_dots = 20;
num_2_print = round(linspace(1,num_print_dots,no_elements));

%Calculating Tx Ray Path
%note: Could maybe do this with arrays and not a for loop
for ii = 1:no_elements
    tic
    distance_T=sqrt(((X-element_positions(ii)).^2)+(Z.^2));
%Pythagoras from grid to tranducer positions

    %Calculating Rx Wave Path
    for jj = 1:no_elements
        distance_R=sqrt(((X-element_positions(jj)).^2)+(Z.^2));
%Pythagors from grid to each indivial trandcuse position
        distance = distance_T + distance_R;
%Summation of total distance from each transmit and receive
        T = distance/velocity;
%Computing time taken of each path
        test=min(max(round...
%Computing the closest index in of sampled time for the time of each path
            (T*sampling_freq),1),length(time));
        curr=time_data(:,ii,jj);
%Current FMC data set of concern
        image=image+curr(test);
%Summing Image Value on each iteration
    end

    %Not necessary but never done one of these before
    %Probably slows down the code somewhat
    %¯\_(?)_/¯
    time_taken = time_taken+toc;
```

```
    est_time = (no_elements - ii) * time_taken/ii;
    clc;
    fprintf('Imaging TFM \n');
    fprintf('Processing TX %d of %d\n',ii,no_elements);
    fprintf('%.2f seconds remaining\n',est_time);
    dots = repmat('.',[1,num_2_print(ii)]);
    spaces = repmat(' ',[1,num_print_dots-num_2_print(ii)]);
    fprintf('[%s%s]\n'...
        ,dots,spaces);
end

%% Refining and Plotting FMC Image
image_original = gather(image);
image = abs(image)/max(max(abs(image)));
%Normalising the image data to the max
image = 20*log10(image);                                    %Converting
to dB scale
image = gather(image);                                      %Gathering
from GPU

%Ploting image
figure(03)
clf
imagesc(x,z,image)
colorbar;
colormap('jet');
caxis([-20 0]);
title('TFM Image of Simulated FMC Data');

%Appliying Median Filtering to image
%Very basic image processing
if noise_present == 0 || initials_present == 0
    J = medfilt2(image,[10 10]);
    figure(04)
    clf
    imagesc(x,z,J)
    colorbar;
    colormap('jet');
    caxis([-20 0]);
    title('TFM Image of Simulated FMC Data - Filtered Image');
end

%% Calculating Image Metrics
%Performed on the following logic
%Get the signal - that's your "true" noiseless image.
%Get the noise - that's your actual noisy image minus the "true" noiseless
image.
%Power Defintion
%Divide them element by element, then take the mean over the whole image
%Voltage Definition
%Divide them element by element, then take the RMS over the whole image

%The above is idealised and assumes you know the 'true' noiseless signal
%How do you do this for real life case when all you have is noisey signal?
%contact e.foster@strath.ac.uk if you know. I would also like to know :)

if noise_present == 0
```

```
    %Determing if metric case has been used & loading in noiseless image
    if no_elements == 64 && element_width == 0.9e-3 && element_pitch == 1e-
3 && centre_freq == 10e6

        if strcmp(sample,'TEST POINTS+NOISE') == 1

            fname = 'TESTPOINTS10MHZ.mat';
            load(fname);

        elseif strcmp(sample,'TEST CRACK+NOISE') == 1

            fname = 'TESTCRACK10MHZ.mat';
            image_noiseless = load(fname);

        elseif strcmp(sample,'TEST NUMBERS') == 1

            fname = 'TESTNUMBERS10MHZ.mat';
            image_noiseless = load(fname);

        end

        %Based off voltage definition
        noise = image_original - image_noiseless;
        SNR = round(20*log10(rms(rms(image_noiseless/noise))),2);

        fprintf('Signal to noise ratio of image is %d \n', SNR);

    end
end
```