ABSTRACT

Sound Navigation and Ranging (SONAR) in underwater is essential to communicate, detect enemy submarines, torpedoes and many other similar applications.

It is defined as ‘method or equipment for determining by underwater sound, the presence, location, or nature of objects in sea’. It is an underwater equivalent of radar which exploits electromagnetic (EM) waves for aerospace application, whereas SONAR uses sound waves for hydrospace application.

The basic principle behind towed array SONAR is beamforming. (Beamforming is one of the basic principles of Sonar)Beamforming is a signal processing technique used in sensor arrays for directional transmission or reception. It works on the principle that there is a delay when a wave front of sound reaches the sensors in the array and as a result there would be a phase difference between the element outputs.

This report is simulation of reception beamforming under different conditions and environments and aims to understand the basics of beamforming.

ABBREVIATIONS

SONAR = Sound Navigation and Ranging
RADAR = RAdio Detection And Ranging
SNR = Signal to Noise Ratio

SYMBOLS

Hz = Hertz
λ = Wavelength (unless mentioned otherwise, that of 2KHz)
x = the distance between two array elements (unless mentioned otherwise, is equal to $\lambda/2$)

CONTENTS

INTRODUCTION

SONAR[1] is a technique that uses sound propagation to navigate, communicate with, or detect objects or or under the surface of the water. Many other methods of detecting the presence of underwater targets in the sea have been investigated such as magnetic, optical signatures, electric field signatures, thermal detection (infrared) , Hydrodynamic changes (pressure) and has had a degree of success. Unfortunately, none of them has surpassed SONAR despite it possessing numerous disadvantages [1] .

There are two types of SONAR : Passive SONAR and Active SONAR

Passive sonar listens to sound radiated by a target using a hydrophone and detects signals against the background of noise. This noise can be self made noise or ambient noise. Self noise is generated inside the receiver and ambient noise may be a combination of sound generated by waves, turbines, marine life and many more.

Active sonar uses a projector to generate a pulse of sound whose echo is received after it gets reflected by the target. This echo contains both signal and noise, so the signal has to be detected against the background noise. The range of the target is calculated by detecting the power of the received signal and thus determining the transmission loss. The transmission loss is directly related to the distance travelled by the signal. In the case of the active sonar, half of the distance travelled by the signal to get attenuated to undergo a transmission loss detected is the range.

Beamforming is a signal processing technique that originates from the design of spatial filters into pencil shaped beams to strengthen signals in a specified direction and attenuate signals from other directions. Beamforming is applicable to either radiation or reception of energy. Here, we consider the reception part. A beamformer is a processor used along with an array to provide a versatile form of spatial filtering. It essentially performs spatial filtering to separate signals that have overlapping frequency content but generate from different spatial directions [2] .

[1] SONAR : SOund Navigation and Ranging

Underlying theory

Despite its major success in air to air detection, RADAR[2] isn't used in sub surface identification and detection. The main reason is the radio wave is an EM wave and the sea water is highly conductive and offers an attenuation of 1400 $\sqrt{f}$ dB/km. This essentially means that the sea water acts a short circuit to the EM energy [1].

Even if we were to use it, to get tangible results, the target should be in short range and shouldn't be completely submerged. Contemporary submarines are nuclear and has the capacity to go deep underwater for indefinite periods. Hence RADAR is useless in subsurface application and can only used along with SONAR without bringing much to the table. Hence SONAR is the dominating method/technology when it comes to underwater detection and ranging.

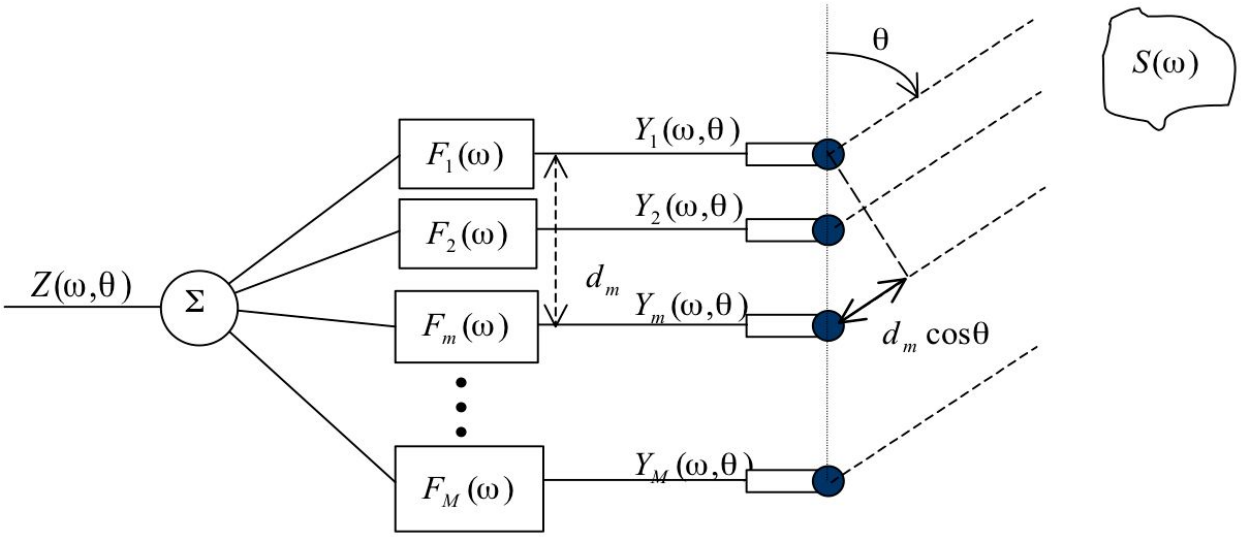
Beam forming or spatial filtering is the process by which an array of large number of spatially separated sensors discriminate the signal arriving from a specified direction from a combination of isotropic random noise called ambient noise and other directional signals. In the following simulations, we deal with 32 elements separated by a distance of $\lambda/2$, where λ is the wavelength of the frequency for which the beam former is designed.

The assumptions under which we perform the simulations are:
(a) the noise is isotropic
(b) ambient noise at the sensor-to-sensor output is uncorrelated,
(c) the signal at the sensor-to-sensor outputs are fully correlated [3].

The sensor spacing in the array is decided based on two important considerations namely, coherence of noise between sensors and formation of grating lobes in the visible region. As far as isotropic noise is concerned, the spatial coherence is zero at spacing in multiples of $\lambda/2$ and small at all points beyond $\lambda/2$. To avoid grating lobe, the spacing between sensors must be less than $\lambda/2$. Hence $\lambda/2$ is chosen as the distance between two elements in the array.

We also assume that the source is far away so as a result, the wave fronts are parallel straight lines. Since the source would be at an angle relative to the axis, the wavefront reaches each elements with varying delay. As a result, the output of each element will have a phase delay from each other.

[2] RADAR : RAdio Detection And Ranging



Using the above figure, we can calculate the corresponding delay of each elements. This will help us in determining to what degree we would have to delay the element outputs to obtain all the outputs of elements in co phase. Once the outputs are made co phase, they are added. For an M element array, the co-phase addition increases the signal power N^2 times and the uncorrelated noise power N times. Thus the SNR is enhanced by N times.

By changing the delays of the element's output, we can "steer" the setup to give the gain to signals coming from a certain direction. This is called beam steering and is one of the most attractive features of a beamformer. However, as one 'steers' a beamformer, the width of the main lobe goes on increasing because the effective length of the array decreases.

In our simulation, we create a matrix with the columns corresponding to the output of each elements for a source at a certain angle. The noise is then added. This is the output of the elements in the array. This is the basic setup. The array is then manipulated or used as input for other array manipulations to obtain the solution to the problem/objective posed.

Let the source signal be $s(k)$. The output of the elements are delayed versions of $s(k)$. So, for an element 'i', the output signal would be

$$y(k) = s[k - \tau_i(\theta)]$$

Using the fourier transform, we get

$$Y_i(w,\theta) = e^{-jw\tau_i(\theta)} S[w]$$

Where $\tau_i(\theta) = (d_m cos\theta / c) F_s$

Where,

d_m : the distance between the element considered and the element where the wavefront first strikes.

θ : the angle the rays make with the array axis

c : speed of sound in the water

F_s : The sampling frequency of the elements.

Now, we need to construct a steering vector.

$$d(w,\theta) = [1 \quad e^{jw\tau_2(\theta)} \quad e^{jw\tau_3(\theta)} \ldots\ldots\ldots\ldots \quad e^{jw\tau_M(\theta)}]$$

To obtain the element output, we multiply this matrix with the signal function.
We have,

$$\mathbf{Y}(w,\theta) = d(w,\theta)\, S[w]$$

The output signal is,

$$Z(w,\theta) = \sum_{i=1}^{M} F_i^*(w) Y_i(w,\theta) = F^H(w) \cdot Y(w,\theta)$$

F^H : the matrix containing complex weights.

Expression for (//complex radiation field) beam pattern of uniformly spaced linear array

The complex radiation field produced by a linear array of N passive receivers is given by

$$Z(w,\theta) = \frac{1}{M} \sum_{m=1}^{M} s(w) e^{-j(m-1)\frac{wd}{c}(cos\theta - cos\varphi)}$$

Objective 1 : Simulate a sine wave + noise in MATLAB and see the effect of SNR on both time domain and frequency domain

Description:

Here, a sine wave is created and simulated to replicate the conditions in beamforming and to see the output of a single element in the array. The noise is added by processing the SNR parameter. The changes are then observed by changing SNR.

The changes are observed in both time and frequency domain. The change in time domain is observed by plotting amplitude vs time. The change in frequency domain is observed by first taking fourier transform and then plotting absolute value vs frequency.

PLOT :

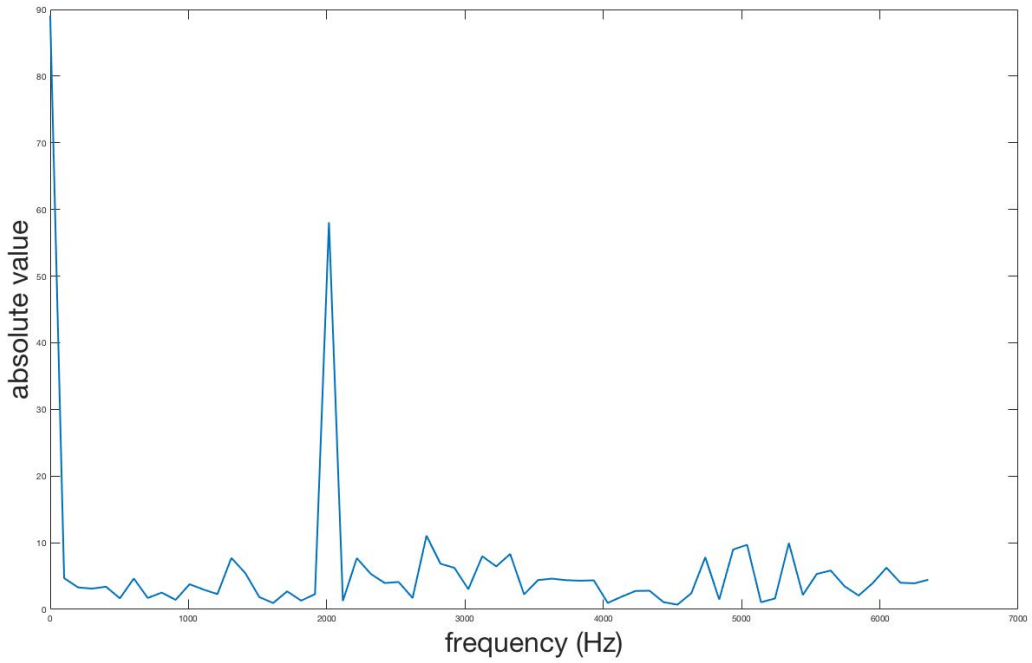


Figure 1.1. Fourier transform when SNR = -3

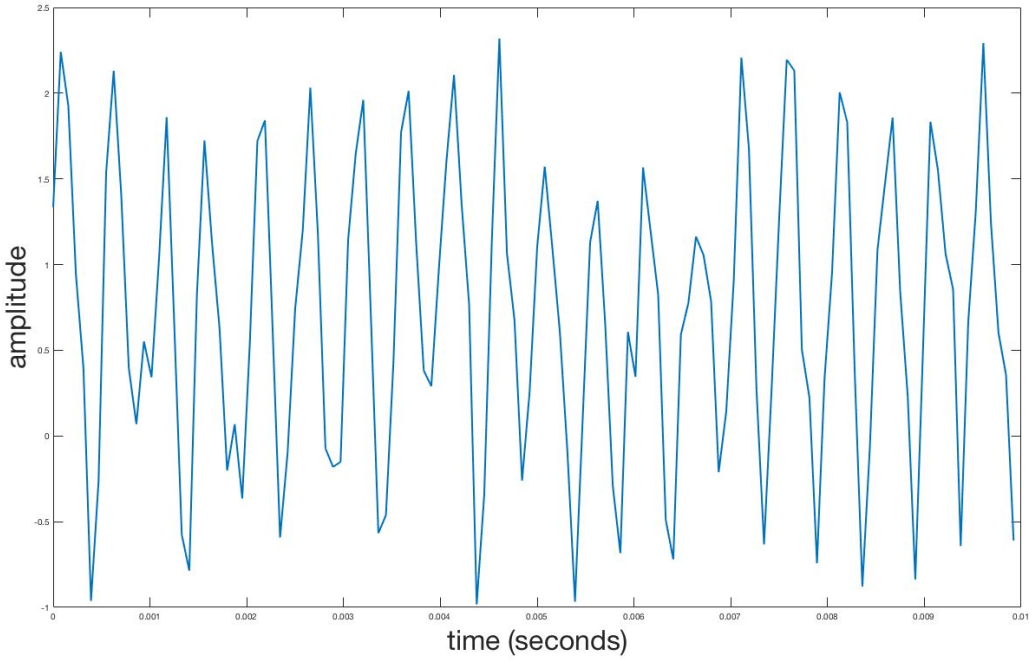


Figure 1.2. Element output signal when SNR = -3

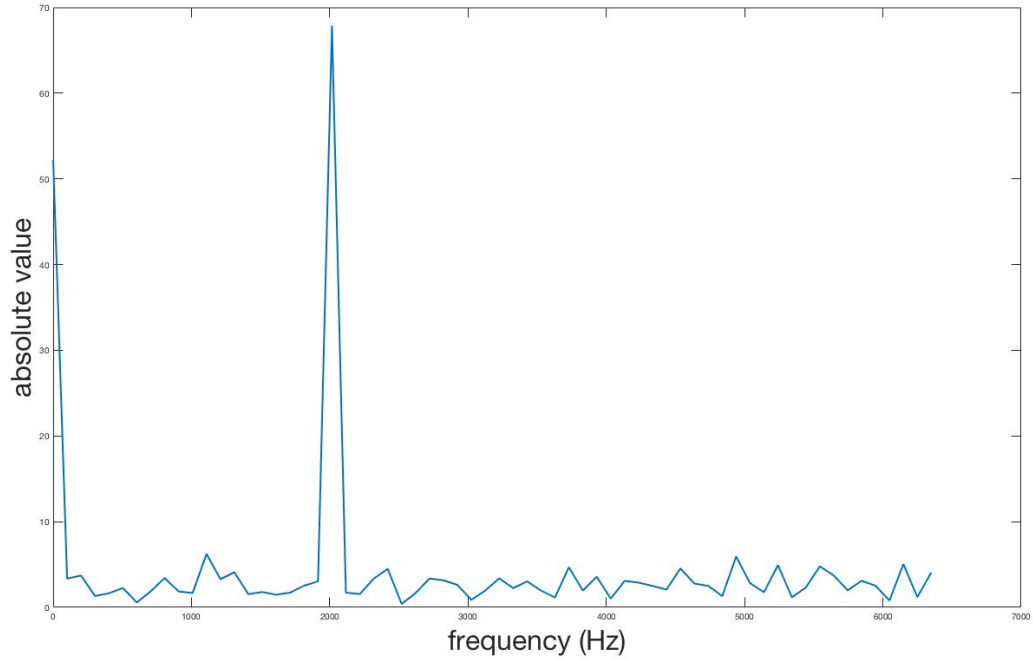


Figure 1.3. Fourier transform when SNR = 1

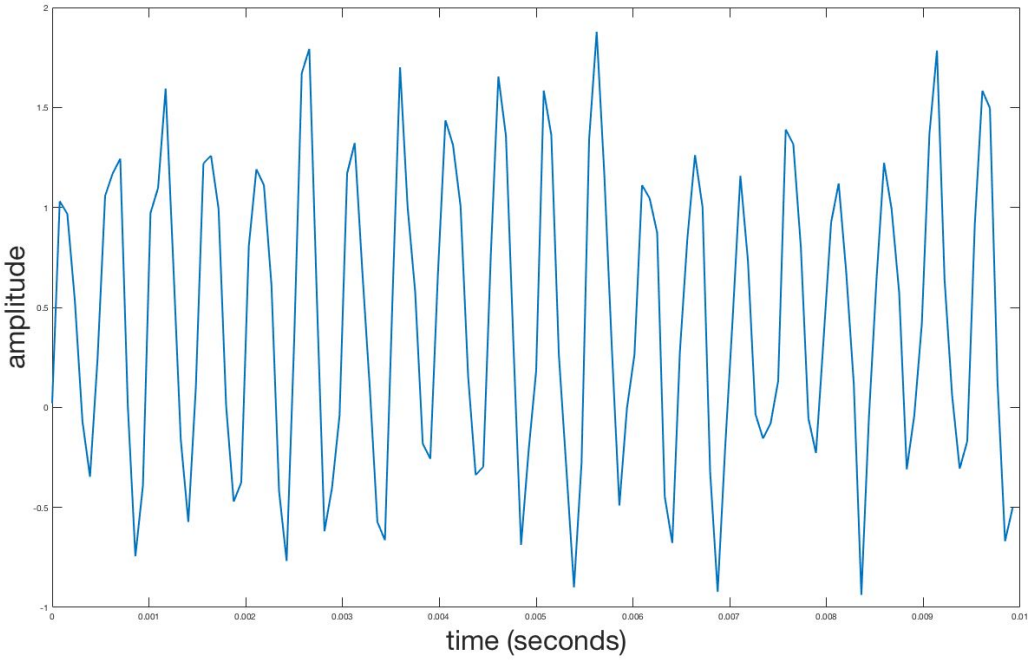


Figure 1.4. Element output signal when SNR = 1

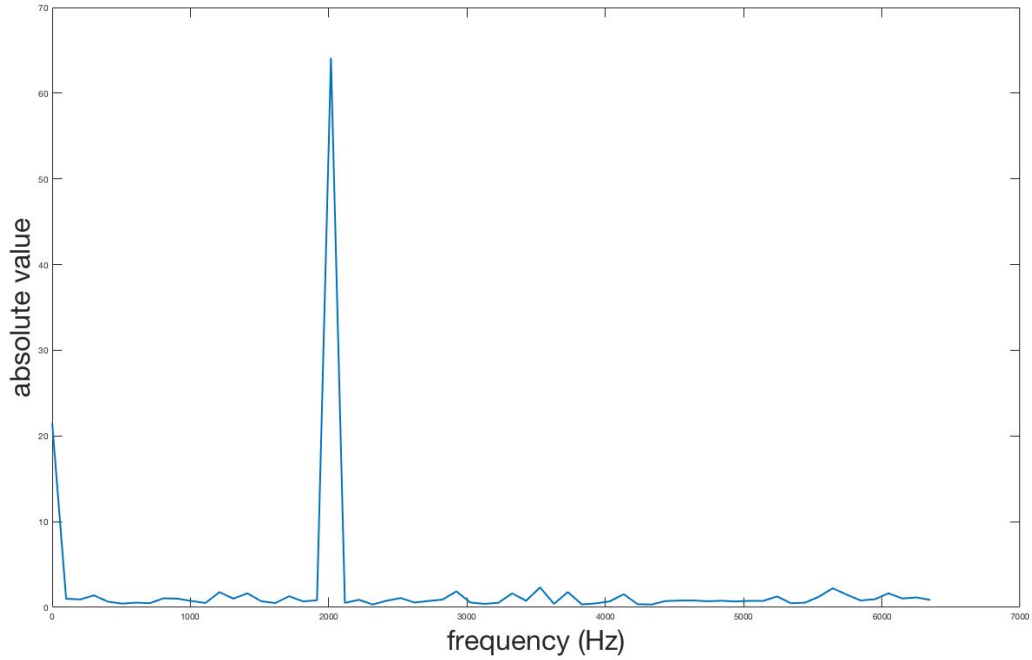


Figure 1.5. Fourier transform when SNR = 10

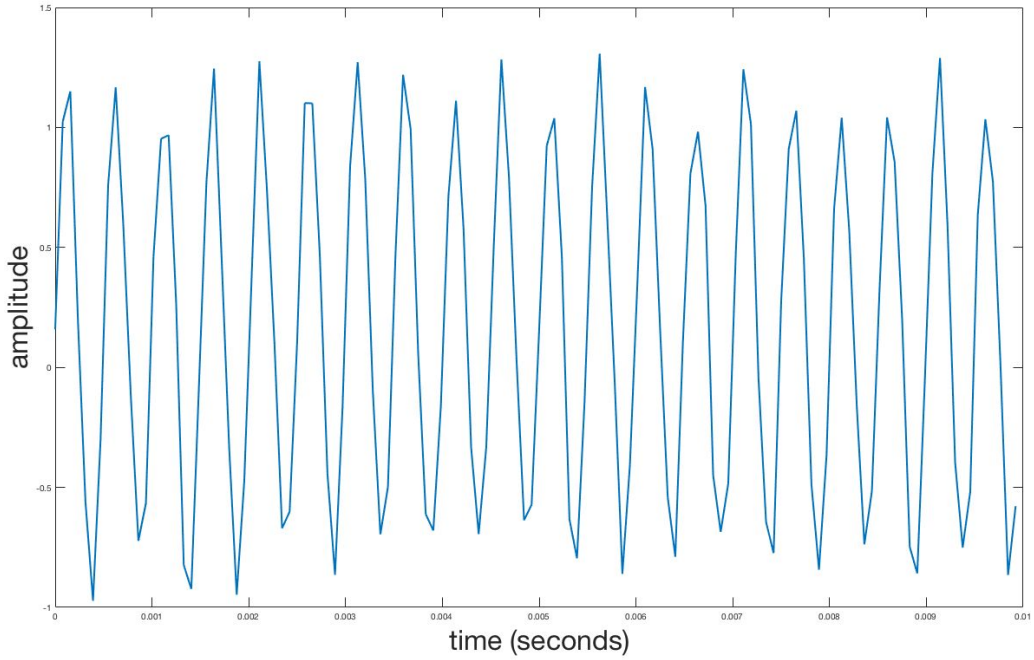


Figure 1.6. Element output signal when SNR = 10

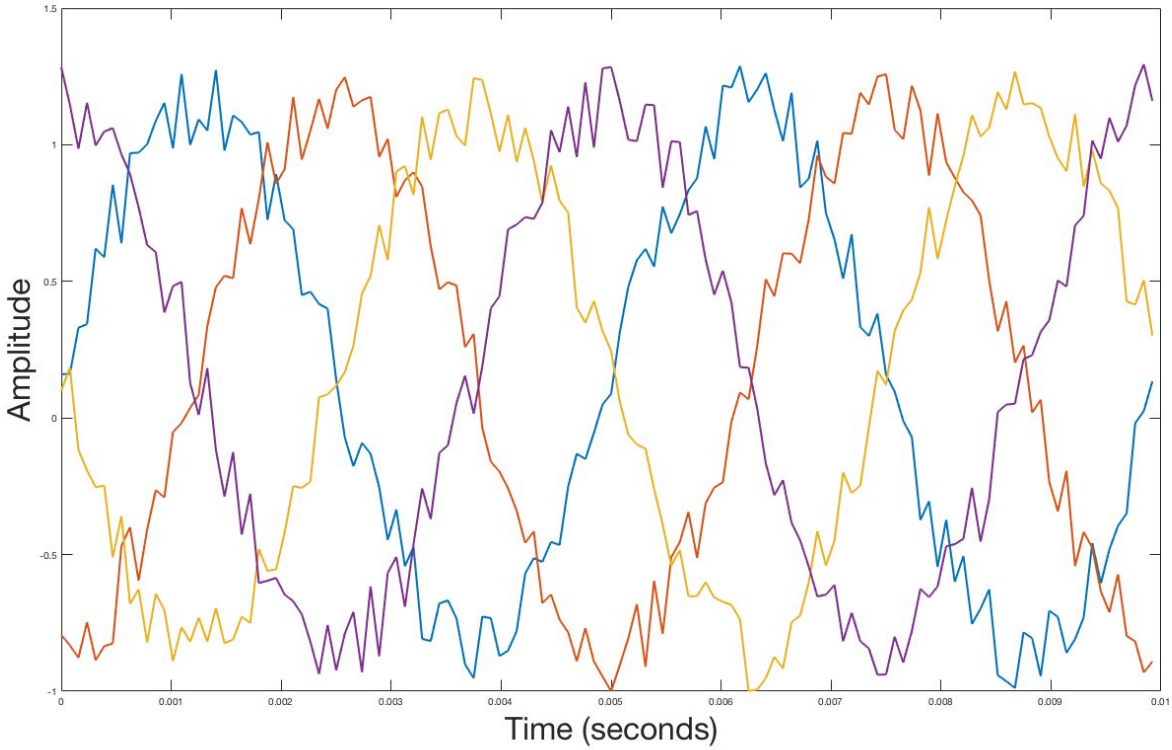
Observations :

From figures 1.2,1.4 and 1.6, we can see that with decreasing SNR, the amplitude-time graph has varying amplitude without a certain pattern. From figures 1.1,1.3 and 1.5, the components corresponding to signals other than source signals exist with increasing amplitude.

Objective 2 : Simulate the input to a 4 element array.

Here, the outputs of a 4 element array is simulated. The location of the source signal is far and therefore the wave fronts are parallel and arrives at the same angle at each element. As a result there would be a phase difference in the output of each element even for the same source signal due to difference in distance travelled. The sine wave is created as output from each element and corresponding phase difference is brought about. The noise is added by processing the SNR parameter.

PLOT :



Observation :

The outputs from each element differs in phase. They vary from the ideal or source signal because they are corrupted due to the presence of noise in the working environment.

Objective 3 : Beamforming in the frequency domain. (Narrowband beamformer)

The outputs of each element in the array differ in phase. They are made in phase again by bringing about an artificial delay corresponding to the array position. In this code, this artificial delay is brought on the property of Fourier transform that

$$x(t - t_0) \leftrightarrow e^{-jwt_o} X(w)$$

So the input is Fourier transformed. Then a weight vector is defined for each angle ranging from 0 to 180 degrees and the two matrices are multiplied. Then the absolute value vs angle is plotted.

PLOT :

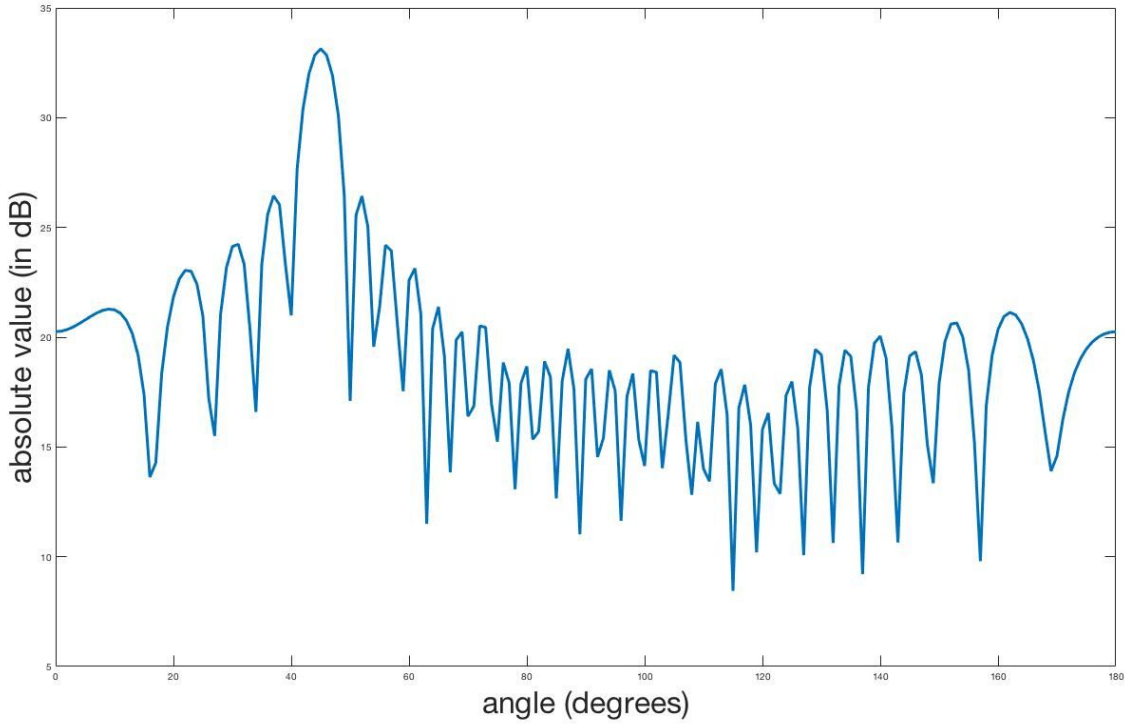


Figure 3.1 : beamforming at angle 45

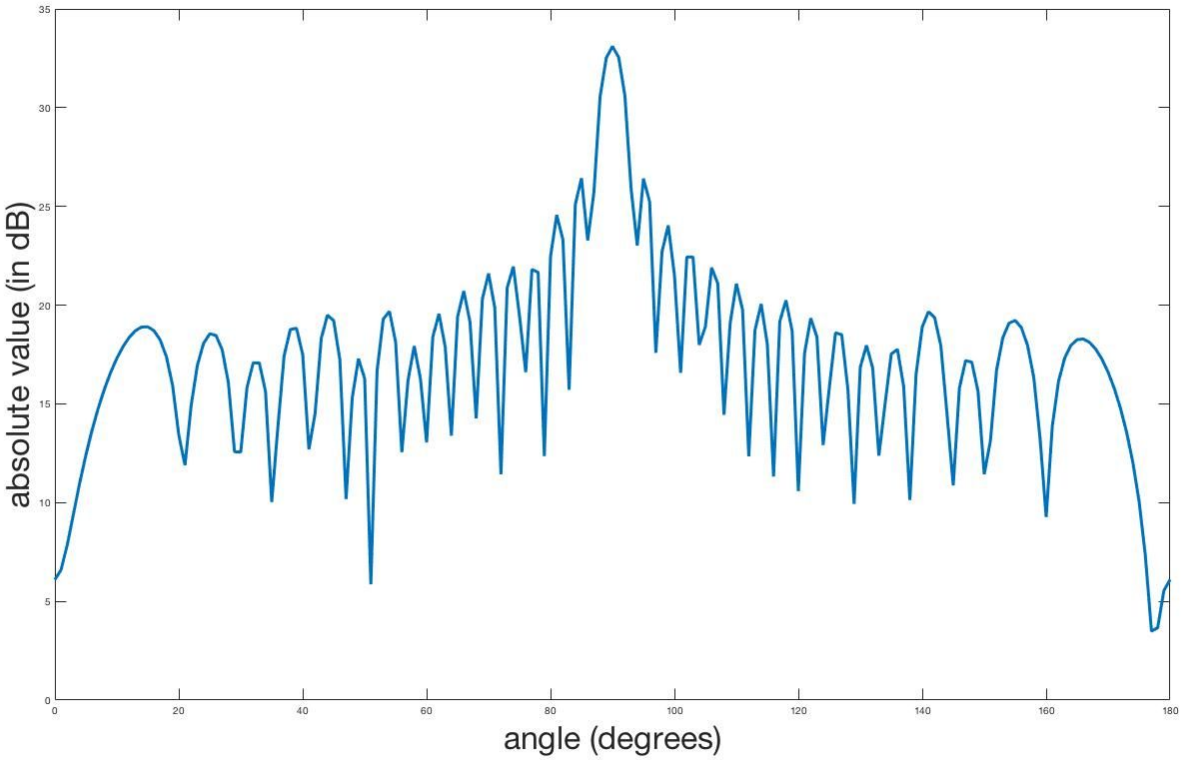


Figure 3.2 : Beamforming at angle 90

Observation :

It is seen that for the angle equal to the source angle, a maxima is given and for every other angles, an absolute value much smaller than the peak value is given.

Objective 4 : Simulate beam pattern by shifting theta

In this code, we explore the property of the beamformer in hand. The beam former is essentially a spatial filter. That is, it attenuates the signals which doesn't come from a specific direction (which we decide when designing) and provides a gain for signals coming in the direction.

We show this by plotting the gain vs angle.

PLOTS :

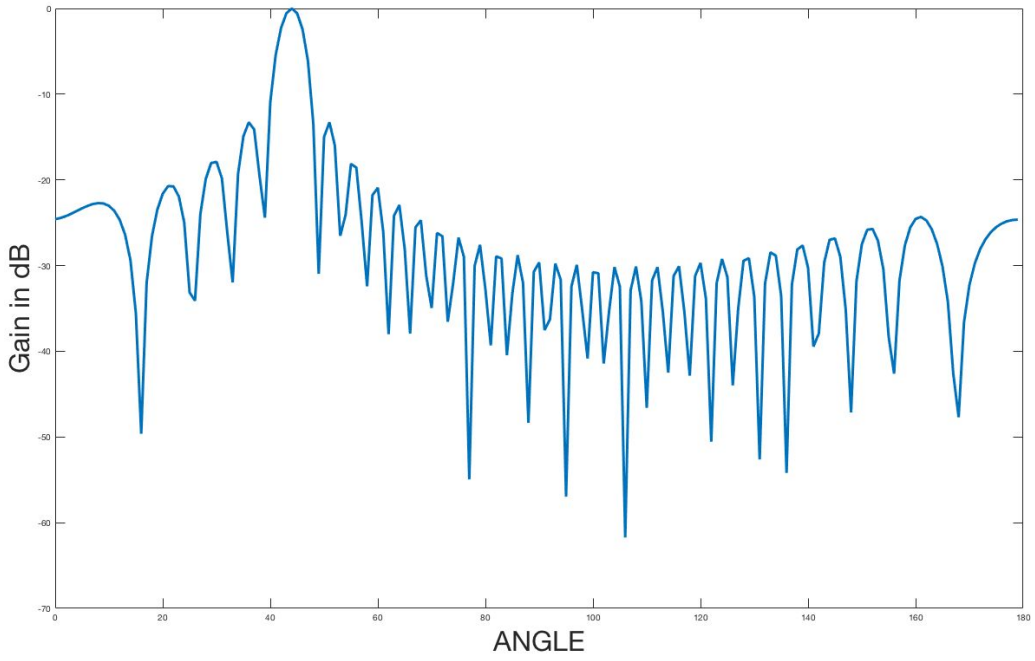


Figure 4.1 : Beam pattern for beam former at angle 45

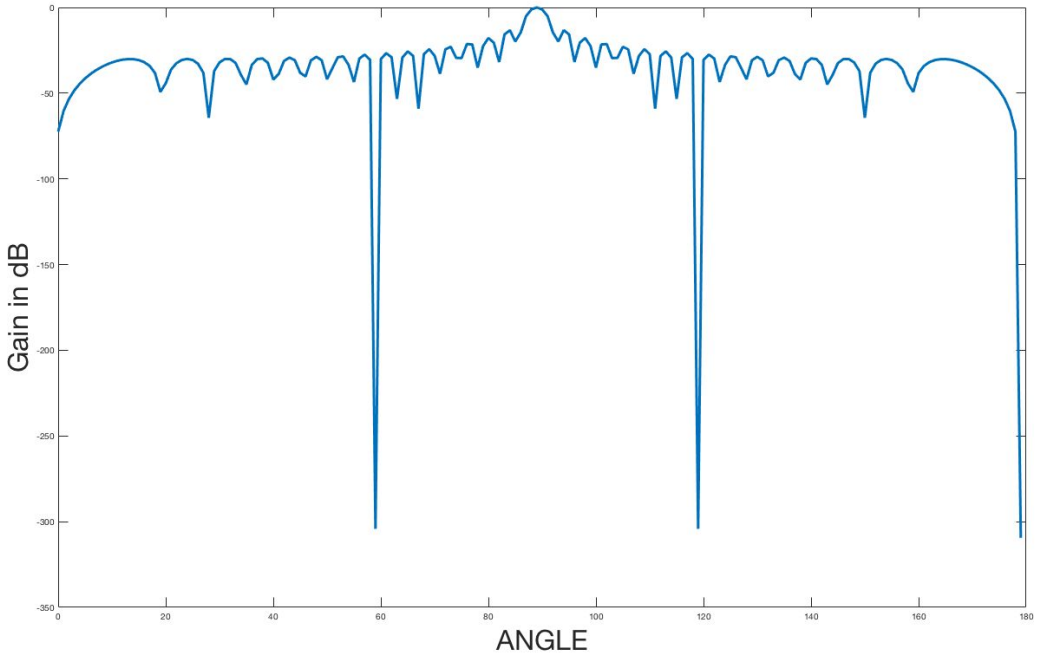


Figure 4.2 : Beam pattern for beamformer at 90 degrees

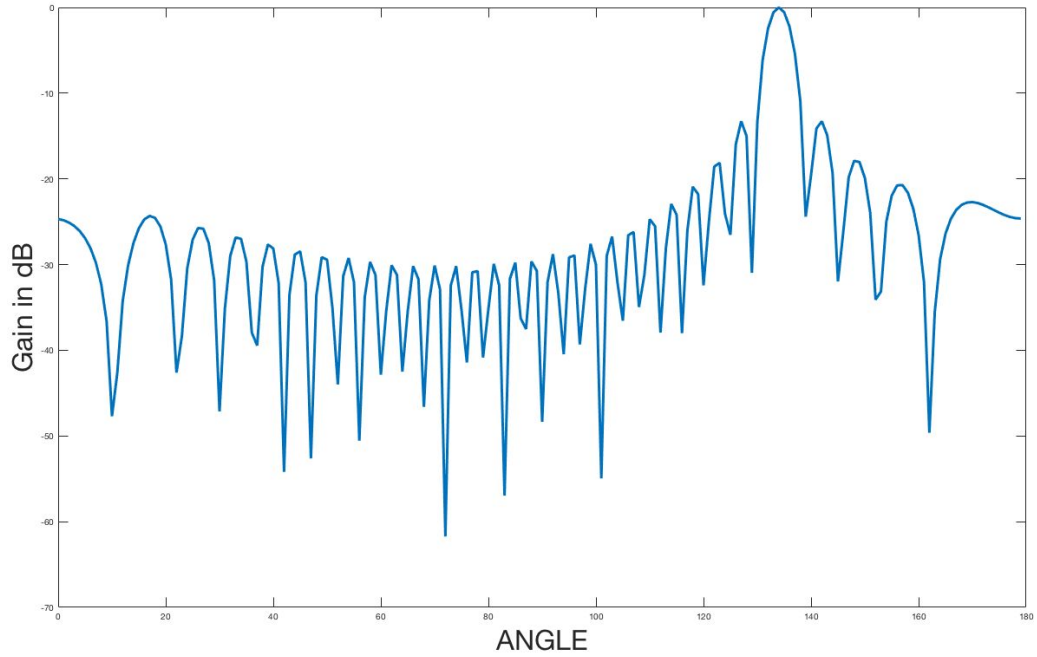


Figure 4.3 : Beam pattern for beamformer at 135 degrees

Observation :

We can see that when we change the designed angle, the gain and attenuation correspondingly move to the designed angle.

OBJECTIVE 5 : SIMULATE BEAM PATTERN FOR DIFFERENT FREQUENCIES ON AN ARRAY DESIGNED FOR A CERTAIN FREQUENCY

Designing an array for a frequency is done by placing the elements in an array at interspace distance of half the wavelength of the signal in scrutiny. With increasing interspace distance, there is an increasing probability of end fire anomaly. End fire anomaly is the phenomenon when beam pattern is plotted, another maxima other than the original maxima arises.

The array we designed is designed for the frequency of 2kHz. So, to see how change in frequency affects the beam pattern, what we have to do is change the weight vector to corresponding values of the frequency and plot absolute value vs angle.

PLOT :

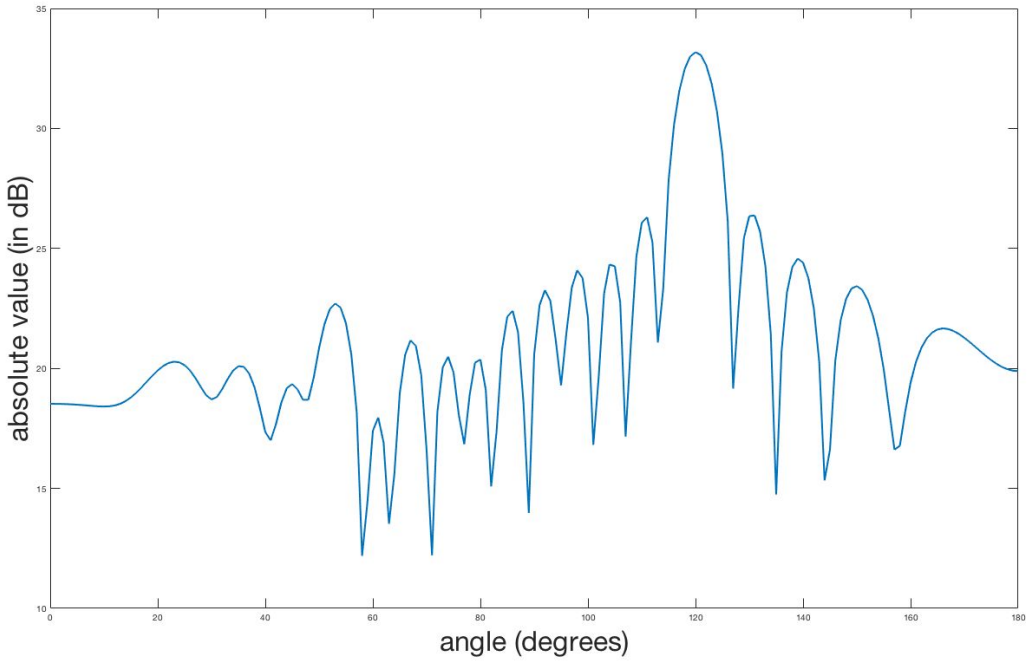


Figure 5.1 : Beamformed for 1200Hz

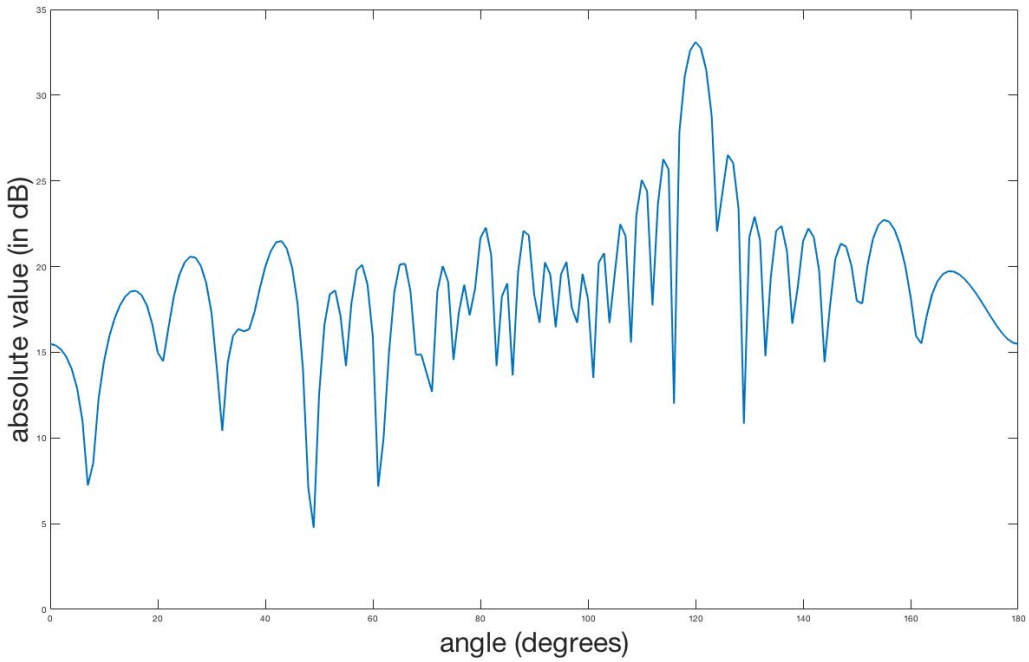


Figure 5.2 : Beamformed for 2000 Hz

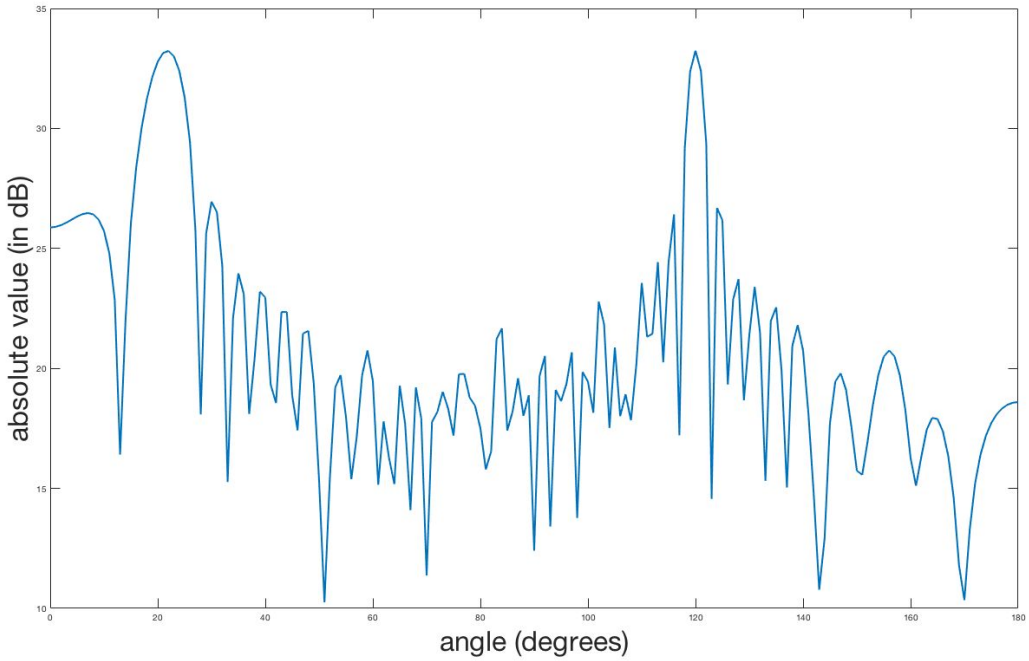


Figure 5.3 : Beamformed for 2800 Hz

Observation :

We observe that as frequency goes below the value of the designed frequency, we still get a working beam pattern with no end fire anomaly. But as we increase the frequency beyond the designed frequency, another maxima arises. This poses the problem of ambiguity.

OBJECTIVE 6 : EFFECT OF SNR ON BEAM PATTERN

SNR or signal to noise ratio plays an important role in beamforming. The beamformer is a spatial filter. That is, it gives a gain for signal coming from a certain angle/direction. This is an added advantage because along with attenuating other signals, it enables us to still find the source location even if the SNR is low.

In this code, we intend to see how change in SNR affects the relative side lobe levels. And to what extent we can find the source angle without ambiguity arising. The array is broadside beamformed.

PLOT :

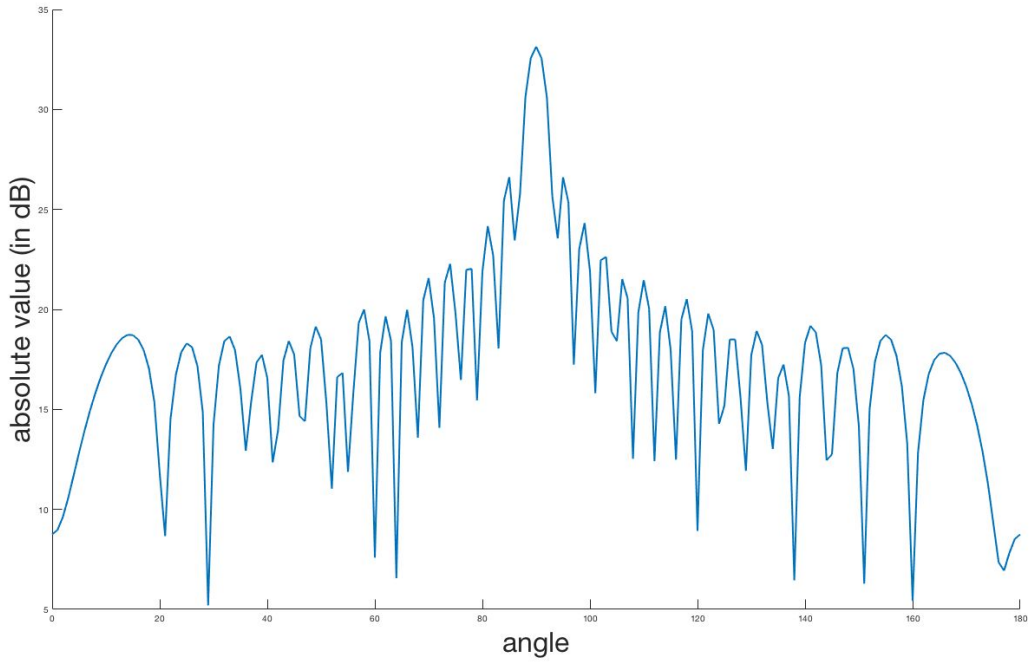


Figure 6.1 : Beamforming for SNR = 1

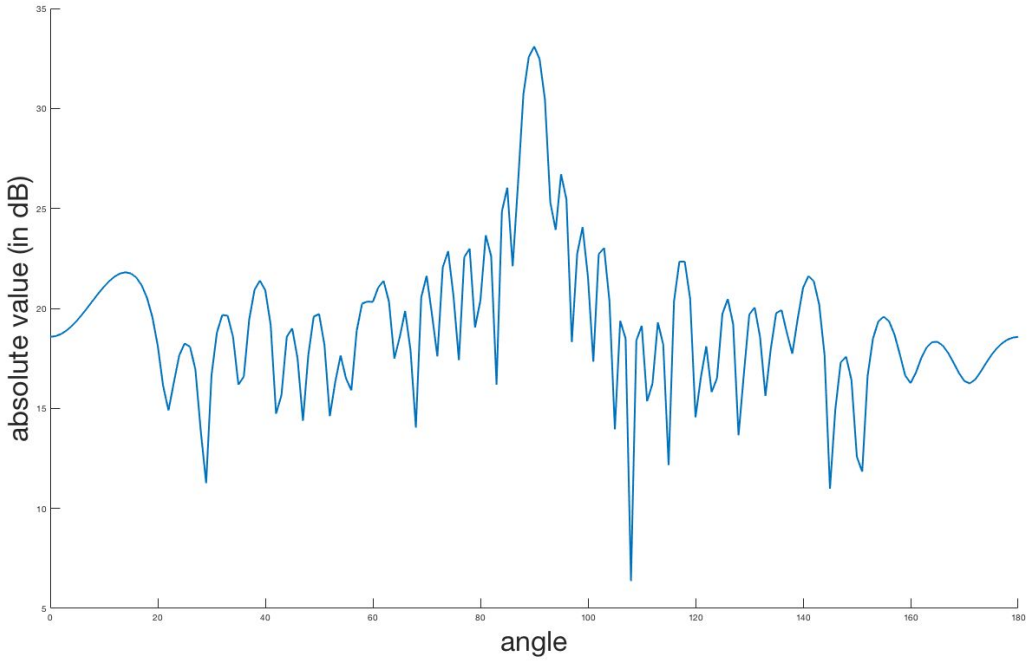


Figure 6.2 : Beamforming for SNR = -10

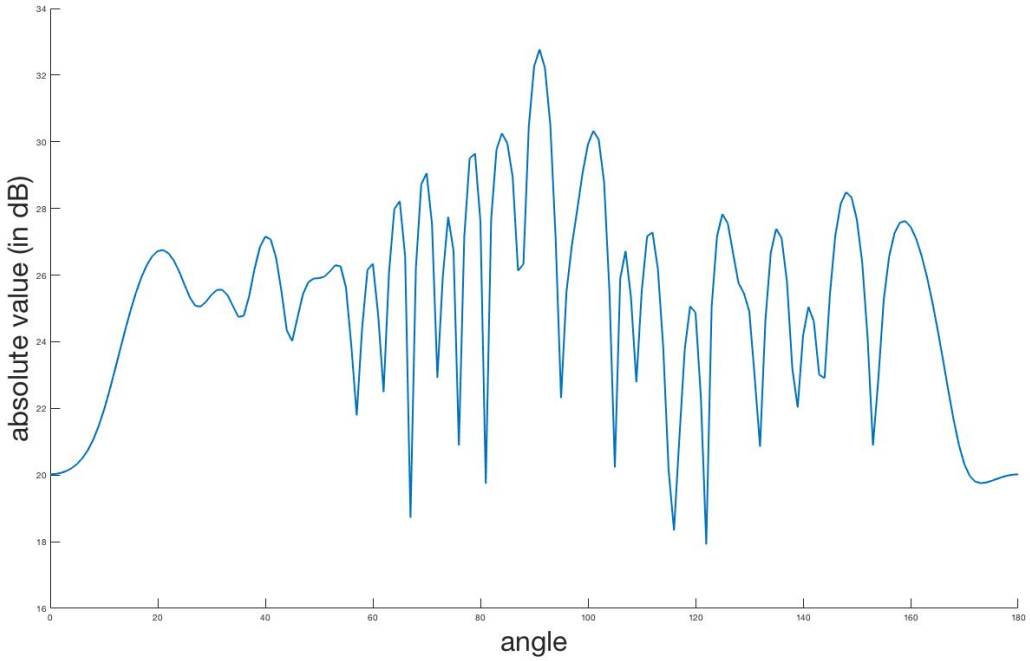


Figure 6.3 :Beamforming for SNR = -30

OBSERVATION :

We see that as SNR decreases, the relative side lobe levels rise and eventually will reach a level where we cannot differentiate between the side lobe and the main lobe.Smaller the value of SNR, more difficult it is to determine the source signal location.

OBJECTIVE 7 : TO GET THE ORIGINAL SIGNAL FROM THE BEAM FORMED OUTPUT

Once the element signal outputs are fourier transformed and beamformed, it is essential to bring back the signal to the time domain. Since we deal with discrete series here, a problem arises because of cyclic delay.The last few samples would be error filled. We overcome this problem by using overlap-and-save method.

Here, two blocks are taken from the input matrix and then fourier transform is taken separately. And after beamforming, we take the inverse fourier transform. After taking the inverse fourier of the two blocks, we concatenate each other after removing the last few parts of both the blocks.

PLOT :

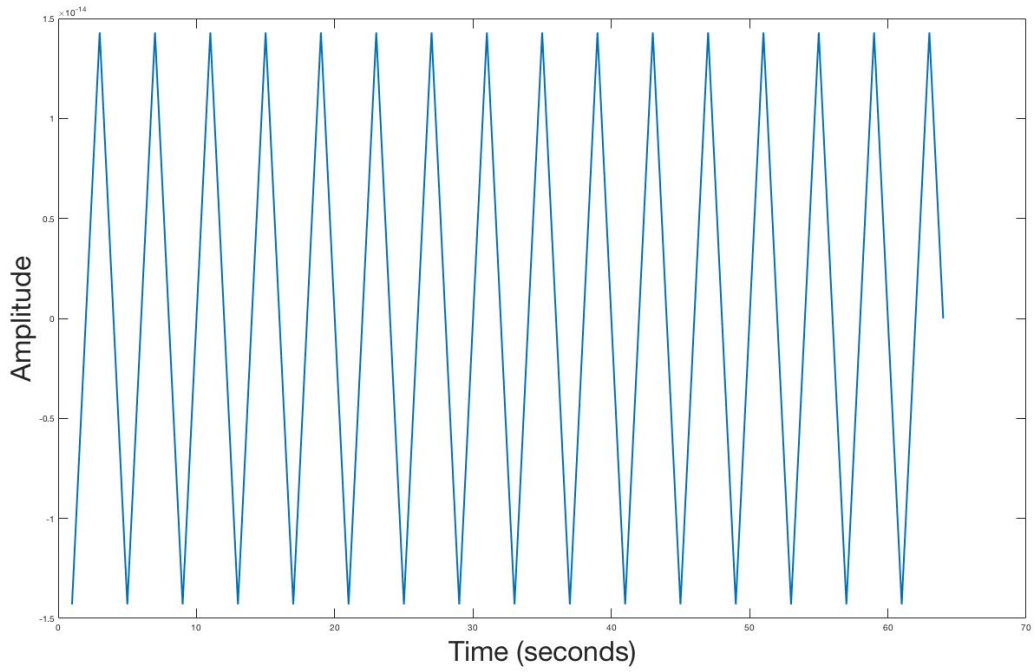


Figure 7.1 The first block of time signal

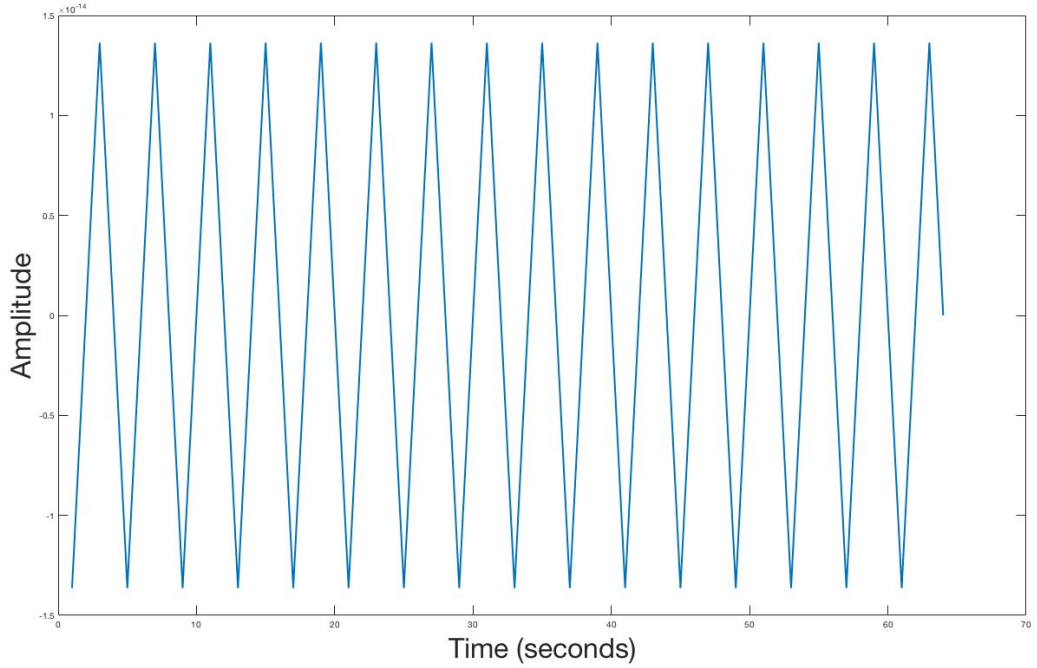


Figure 7.2 The second block of time signal

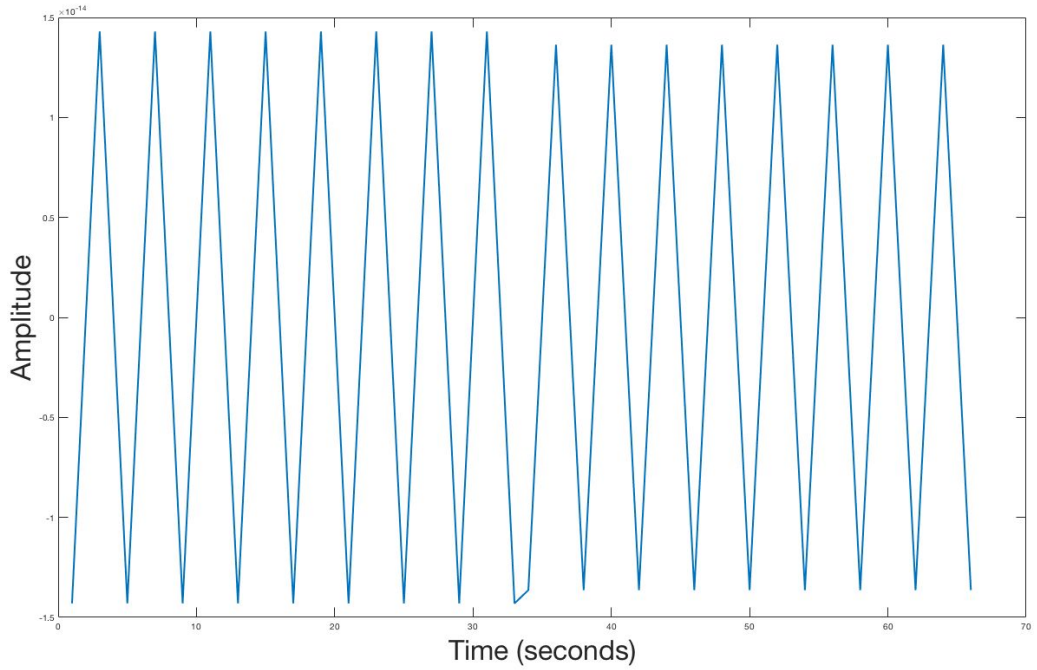


Figure 7.3 The second block of time signal

Observation :

Figure 7.1 is the first block. We take a fixed number of samples and treat it like a new signal and this is called a block. Similarly Figure 7.2 shows the second block. Figure 7.3 is the final result. The one we get after concatenating the two blocks after removing a few samples from both of them. We can see that this is a good approximation of the initial signal.

OBJECTIVE 8 : SIMULATE BROADBAND BEAMFORMING

The aim is to beam form the incoming signal for different frequencies and to find out the direction of source signal from the plot. Here, we deal with a single frequency source coming from a single source. But here, we are unaware of the frequency of the source along with the angle.

In this case, we have to beam form for each frequency and sweep for angles from 0 to 180. Then the absolute values of each frequency for each angle are summed up and plotted in an abs value vs angle graph.We only consider the nine frequencies for the sake of simplicity.

PLOTS :

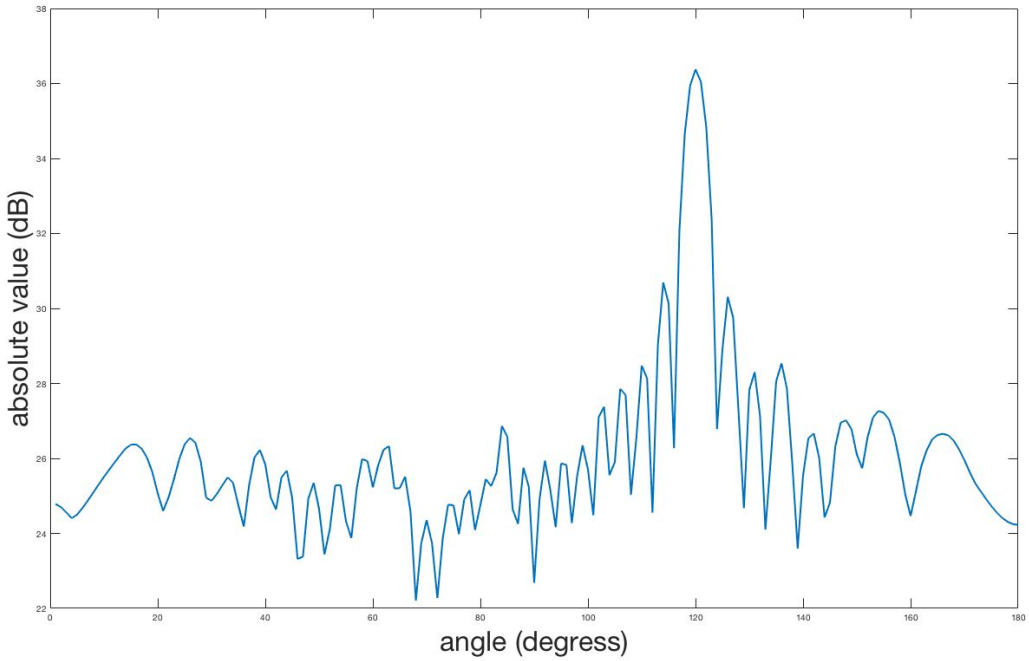


Figure 8. Beamformed output for chosen frequencies

OBSERVATION :

Here we will observe that for the plot is similar for beamforming for one frequency because for every frequency other than the signal frequency, the absolute values are very low. A peak appears corresponding to the angle where the source is and absolute values at every other angle is very low.

OBJECTIVE 9 : Apply the chebyshev windowing in beamforming

Traditional beamforming (ie, rectangular windowing) brings along with it significant disadvantages. The first-order, second-order and third-order sidelobes are respectively only 13.5, 18 and 21 dB below the peak level of the main lobe. Strong signals will be detected through the sidelobes of adjacent beams as well as, correctly, in the main lobe of the beam at the bearing of the signal. The resultant bearing ambiguity and additional, false, signals complicate all further processes [1] .

The aim, therefore, should be to produce the narrowest possible main lobe consistent with some reasonable level of sidelobes. One such method is chebyshev window. Here, we multiply the amplitude of each element's output with shading coefficients obtained from the chebwin function in matlab.

PLOTS :

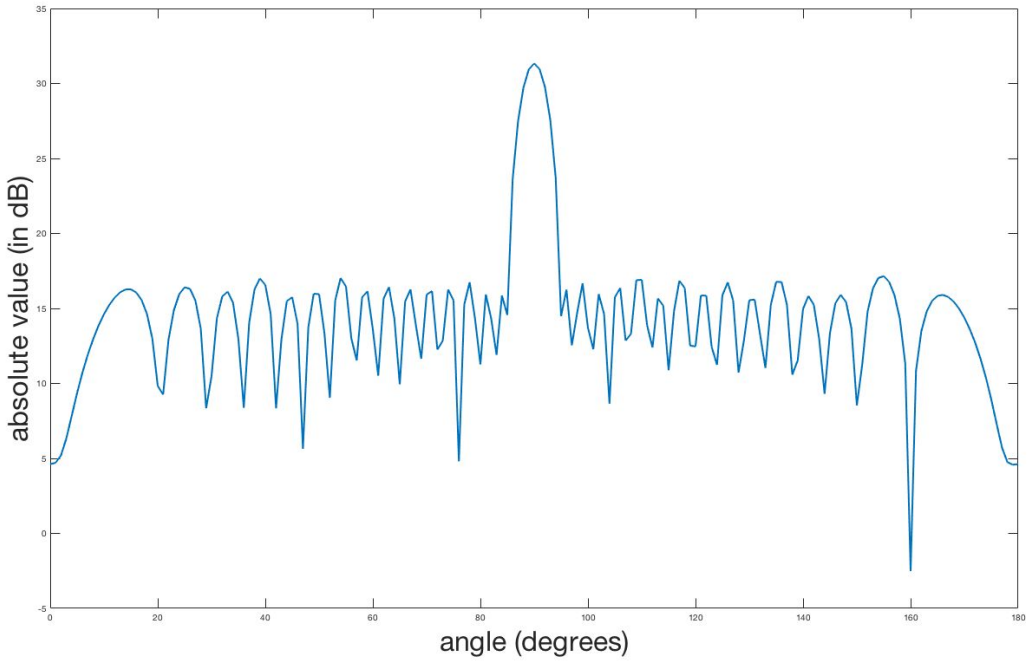


Figure 9 : Beamformed for broadside using chebyshev window

Observation :

The side lobes all fall to a same level. But this reduction in the side lobe seems to be at the expense of the widening of the main lobe ie the width of the main lobe increased. This is to be used in places where it is imperative that the side lobes be decreased and increase in main lobe width can be tolerated.

APPENDIX A

MATLAB CODE :

OBJECTIVE 1

```
%% Basic setup
clc;
close all;

%% Initialising variables
f               = 2000;                 %the main frequency
Fs              = 12800;                %sampling frequency
Ts              = 1/Fs;                 %sampling interval
N               = 128;                  %number of intervals

SNR             = 10;                   %signal to noise ratio
SNR_weight = 10^(-1*SNR*0.05);          %SNR multiplying factor

t         = (0:N-1)*Ts;                 %time matrix
new_mat    = zeros(1,N);                %initialising noise included signal

%% creating the sine wave
y = sin(2*pi*f*t);                      %signal vector

%% adding the noise
new_mat = y + SNR_weight*rand(1,N);   %noise included signal vector

%% plotting the wave in the time domain
time_axis = linspace(0,(N-1)*Ts,N);        %creating the time axis

figure(1)
plot(time_axis,new_mat,'linewidth',2);     %plotting the signal in time domain
xlabel('time (seconds)','FontSize',32)
ylabel('amplitude','FontSize',32)

%% taking the fourier transform
NFFT = N;                               %number of frequency samples
```

```
fend = (NFFT-1)*Fs/NFFT;                %finding the end frequency
w_axis = linspace(0,fend/2,NFFT/2);     %creating the spacing

Fourier = fft(new_mat,NFFT);            %taking the fourier transform

%% plotting the frequency domain
w = 1:NFFT/2;
figure(2)
plot(w_axis,abs(Fourier(1,w)),'linewidth',2);          %plotting the fourier matrix
xlabel('frequency (Hz)','FontSize',32)
ylabel('absolute value','FontSize',32)
```

OBJECTIVE 2

```
%% basic setup
clc;
close all;

%% Initialising variables
f          = 200;                       %the main frequency
Fs         = 12800;                     %sampling frequency
Ts         = 1/Fs;                      %sampling interval
N          = 128;                       %number of intervals

m          = 4;                         %number of sensors
angle      = 60;                        %incoming angle
c          = 1500;                      %speed of the sound signal
lambda     = c/f;                       %wavelength of incoming signal
x          = lambda/2;                  %sensor interspacing
d          = x*cosd(angle)/c;           %unit delay

SNR        = 10;                        %signal to noise ratio
SNR_weight = 10^(-1*SNR*0.05);          %SNR noise weight

t          = (0:N-1)*Ts;                %time matrix
matrix     = zeros(N,m);                %initialising noise included signal

%% bringing about the natural delay
y = sin(2*pi*f*t);                      %generating the ideal sine wave
```

```
for i = 1:m
   matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
new_mat = zeros(N,m);                    %initialising the noise matrix
new_mat = matrix + SNR_weight*rand(N,m);   %creating the impure matrix

%% plotting the wave in the time domain
time_axis = linspace(0,(N-1)*Ts,N);       %creating the time axis

figure(1)
plot(time_axis,new_mat,'linewidth',2);    %plotting the signal in time domain
xlabel('Time (seconds)','FontSize',32)
ylabel('Amplitude','FontSize',32)
```

OBJECTIVE 3

```
%% basic setup
clc;
close all;

%% initialising the variables
f          = 2000;                    %the main frequency
Fs         = 12800;                   %sampling frequency
Ts         = 1/Fs;                    %sampling interval
N          = 128;                     %number of intervals

m              = 32;                  %number of sensors
angle          = 135;                 %incoming angle
c              = 1500;                %speed of the sound signal
lambda         = c/f;                 %wavelength of incoming signal
x              = lambda/2;            %sensor interspacing
d              = x*cosd(angle)/c;     %unit delay

SNR            = 1;                   %signal to noise ratio
```

```
SNR_weight = 10^(-1*SNR*0.05);        %SNR noise weight

t               = (0:N-1)*Ts;              %time matrix
matrix          = zeros(N,m);              %initialising noise included signal

%% bringing about the natural delay
y = sin(2*pi*f*t);                         %generating the ideal sine wave

for i = 1:m
   matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
new_mat = zeros(N,m);                      %initialising the noise matrix
new_mat = matrix + SNR_weight*rand(N,m);   %creating the impure matrix

%% taking the fourier transform
NFFT = N;                                  %number of frequency samples
fend = (NFFT-1)*Fs/NFFT;                   %finding the end frequency
w_axis = linspace(0,fend,NFFT);            %creating the spacing

Fourier = fft(new_mat,NFFT);               %taking the fourier transform

%% Choosing the frequency row
index = f/(Fs/NFFT)+1;
f_mat=zeros(1,m);
f_mat(1,:)=Fourier(index,:);

%% bringing the delay in frequency region

angle_matrix= zeros(1,181);
delay_column = zeros(m,1);

for test_angle = 0:180
   test_d = x*cosd(test_angle)/c;          %the unit delay for test angle

   for i = 1:m
      delay_column(i,1) = exp(1*1i*2*pi*f*(i-1)*test_d);
   end
```

```
    angle_matrix(1,test_angle+1) = abs(f_mat*delay_column);
end

%% plotting the beam formed output
angle_axis = linspace(0,180,181);          %setting up the angle axis

figure(3)
plot(angle_axis,10*log10(angle_matrix),'linewidth',3);    %plotting the angle matrix
xlabel('angle (degrees)','FontSize',32)
ylabel('absolute value (in dB)','FontSize',32)
```

OBJECTIVE 4

```
%% basic setup
clc;
close all;

%% initialising the variables
f           = 2000;                        %the main frequency

m           = 32;                          %number of sensors
angle       = 135;                         %incoming angle
c           = 1500;                        %speed of the sound signal
lambda      = c/f;                          %wavelength of incoming signal
x           = lambda/2;                    %sensor interspacing
d           = x*cosd(angle)/c;              %unit delay

matrix      = zeros(1,m);                  %initialising signal

%% bringing about the natural delay

for i = 1:m
    matrix(1,i) = (1/m)*exp(-1*1i*2*pi*f*(i-1)*d);
end

%% bringing the delay in frequency region
delay_column = zeros(m,1);
```

```
angle_matrix= zeros(1,181);


for test_angle = 1:180
   test_d = x*cosd(test_angle)/c;          %the unit delay for test angle

   for i = 1:m
      delay_column(i,1) = exp(1*1i*2*pi*f*(i-1)*test_d);
   end

   angle_matrix(1,test_angle) = abs(matrix*delay_column);
end

%% plotting the response
angle_axis = linspace(0,180,181);          %setting up the axis values for displaying angle_matrix

figure(2)
plot(angle_axis,20*log10(angle_matrix),'linewidth',3); %plotting the angle matrix
xlabel('ANGLE','FontSize',32)
ylabel('Gain in dB','FontSize',32)
```

OBJECTIVE 5

```
%% basic setup
clc;
close all;

%% initialising the variables
f            = 2800;                %the incoming frequency
f_1          = 2000;                %the frequency for which the array is designed
Fs           = 12800;               %sampling frequency
Ts           = 1/Fs;                %sampling interval
N            = 128;                 %number of intervals

m            = 32;                  %number of sensors
angle        = 120;                 %incoming angle
c            = 1500;                %speed of the sound signal
```

```
lambda      = c/f_1;                    %wavelength of incoming signal
x           = lambda/2;                 %sensor interspacing
d           = x*cosd(angle)/c;          %unit delay

SNR         = -10;                      %signal to noise ratio
SNR_weight = 10^(-1*SNR*0.05);          %SNR noise weight

t           = (0:N-1)*Ts;               %time matrix
matrix      = zeros(N,m);               %initialising noise included signal

%% bringing about the natural delay
y = sin(2*pi*f*t);                      %generating the ideal sine wave

for i = 1:m
   matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
new_mat = zeros(N,m);                   %initialising the noise matrix
new_mat = matrix + SNR_weight*rand(N,m);   %creating the impure matrix

%% taking the fourier transform
NFFT = N;                               %number of frequency samples
fend = (NFFT-1)*Fs/NFFT;                %finding the end frequency
w_axis = linspace(0,fend,NFFT);         %creating the spacing

Fourier = fft(new_mat,NFFT);            %taking the fourier transform

%% Choosing the frequency row
index = f/(Fs/NFFT)+1;
f_mat=zeros(1,m);
f_mat(1,:)=Fourier(index,:);

%% bringing the delay in frequency region

angle_matrix= zeros(1,181);
delay_column = zeros(m,1);

for test_angle = 0:180
```

```
    test_d = x*cosd(test_angle)/c;            %the unit delay for test angle

    for i = 1:m
        delay_column(i,1) = exp(1*1i*2*pi*f*(i-1)*test_d);
    end

    angle_matrix(1,test_angle+1) = abs(f_mat*delay_column);
end

%% plotting the beam formed output
angle_axis = linspace(0,180,181);          %setting up the axis values for displaying angle_matrix

figure(3)
plot(angle_axis,10*log10(angle_matrix),'linewidth',2);    %plotting the angle matrix
xlabel('angle (degrees)','FontSize',32)
ylabel('absolute value (in dB)','FontSize',32)
```

OBJECTIVE 6

```
%% basic setup
clc;
close all;

%% initialising the variables
f           = 2000;                 %the main frequency
Fs          = 12800;                %sampling frequency
Ts          = 1/Fs;                 %sampling interval
N           = 128;                  %number of intervals

m           = 32;                   %number of sensors
angle       = 90;                   %incoming angle
c           = 1500;                 %speed of the sound signal
lambda      = c/f;                  %wavelength of incoming signal
x           = lambda/2;             %sensor interspacing
d           = x*cosd(angle)/c;      %unit delay

SNR         = -30;                  %signal to noise ratio
```

```
SNR_weight = 10^(-1*SNR*0.05);        %SNR noise weight

t                = (0:N-1)*Ts;                %time matrix
matrix           = zeros(N,m);                %initialising noise included signal

%% bringing about the natural delay
y = sin(2*pi*f*t);                            %generating the ideal sine wave

for i = 1:m
   matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
new_mat = zeros(N,m);                         %initialising the noise matrix
new_mat = matrix + SNR_weight*rand(N,m);%creating the impure matrix

%% taking the fourier transform
NFFT = N;                                     %number of frequency samples
fend = (NFFT-1)*Fs/NFFT;                      %finding the end frequency
w_axis = linspace(0,fend,NFFT);               %creating the spacing

Fourier = fft(new_mat,NFFT);                  %taking the fourier transform

%% Choosing the frequency row
index = f/(Fs/NFFT)+1;
f_mat=zeros(1,m);
f_mat(1,:)=Fourier(index,:);

%% bringing the delay in frequency region
angle_matrix= zeros(1,181);
delay_column = zeros(m,1);

for test_angle = 0:180
   test_d = x*cosd(test_angle)/c;             %the unit delay for test angle

   for i = 1:m
      delay_column(i,1) = exp(1*1i*2*pi*f*(i-1)*test_d);
   end
```

```
    angle_matrix(1,test_angle+1) = abs(f_mat*delay_column);
end

%% plotting the beam formed output
angle_axis = linspace(0,180,181);          %setting up the axis values for displaying angle_matrix

hold on
plot(angle_axis,10*log10(angle_matrix),'linewidth',2);  %plotting the angle matrix
xlabel('angle','FontSize',32)
ylabel('absolute value (in dB)','FontSize',32)
```

OBJECTIVE 7

```
%% basic setup
clc;
clear all;
close all;

%% initialising variables
f1        = 1000;                    %the lower limit
f2        = 2000;                    %the upper limit
angle     = 60;                      %angle ranges from 0 to 180
f         = 200;                     %frequency of the analog wave
Fs        = 800;                     %sampling frequency
Ts        = 1/Fs;                    %sampling time
N         = 256;                     %number of samples
c         = 1500;                    %speed of sound in water
m         = 32;                      %number of elements
SNR       = 55;                      %signal to noise ratio

lambda = c/f;                        %wavelength
x         = lambda/2;                %interspace distance
d         = x*cosd(angle)/c;         %quantum delay

t         = (0:N-1)*Ts ;             %creating the time
matrix    = zeros(N,m);              %delayed pure signals

L         = 64;                      %segment length
N_L       = N/L;                     %number of such segments
```

```
%% bringing about the natural delay
y = sin(2*pi*f*t);                          %generating the ideal sine wave

for i = 1:m
    matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

tend = (N-1)*Ts;                            %the final value in linspace
xaxis=linspace(0,tend,N);

new_mat    = matrix;

%% Here is where the blocking happens

%initialising the blocks
segment_matrix_1 = zeros(L,m);
segment_matrix_2 = zeros(L,m);

%transferring the data value for the blocks
for i = 1:L
     segment_matrix_1(i,:) = new_mat(i,:);
     segment_matrix_2(i,:) = new_mat(L+i,:);

end

%% fourier transform

%initialising the fourier
Fourier_1 = zeros(L,m);
Fourier_2 = zeros(L,m);

NFFT  = L;
fend  = (NFFT-1)*Fs/NFFT;
waxis = linspace(0,fend,NFFT);          %creating the spacing

%setting up the fouriers
Fourier_1 = fft(segment_matrix_1,NFFT);
Fourier_2 = fft(segment_matrix_2,NFFT);
```

```
%% beamforming
%the angle matrix stores the magnitude of response for each angle

angle_matrix_1 = zeros(1,181);        %initialising the angle matrix
angle_matrix_2 = zeros(1,181);

f_mat_1 =  zeros(1,m);                %initialising the matrix for the chosen frequency
f_mat_2 =  zeros(1,m);

index       = (f/(Fs/NFFT))+1;        %finding the index value of f
delay_column = zeros(m,1);            %initialising the delay column

f_mat_1(1,:) =  Fourier_1(index,:);   %extracting the values for the frequency
f_mat_2(1,:) =  Fourier_2(index,:);

for test_angle = 0:180
   test_d = x*cosd(test_angle)/c;     %quantum delay for test angle

   for i = 1:m                        %setting up the delay column
      delay_column(i,1,:) = exp(-1*1i*2*pi*f*(i-1)*test_d);        %steering vector
   end

   angle_matrix_1(1,test_angle+1,:) = f_mat_1*delay_column; %storing
   angle_matrix_2(1,test_angle+1,:) = f_mat_2*delay_column;

end

%% Constructing the artificial fourier

%initialising the artificial fourier
art_fourier_1 = zeros(1,L);
art_fourier_2 = zeros(1,L);

%setting up the artificial fourier
art_fourier_1(1,index) = angle_matrix_1(1,angle+1);
art_fourier_2(1,index) = angle_matrix_2(1,angle+1);

art_fourier_1(1,index + NFFT/2) = conj(angle_matrix_1(1,angle+1));
```

```
art_fourier_2(1,index + NFFT/2) = conj(angle_matrix_2(1,angle+1));

%% taking the inverse fourier

inv_fourier_1 = ifft(art_fourier_1);
inv_fourier_2 = ifft(art_fourier_2);

%% plotting the inverse fourier

figure(4)
plot((inv_fourier_1),'linewidth',2)
xlabel('Time (seconds)','FontSize',32)
ylabel('Amplitude','FontSize',32)


figure(5)
plot((inv_fourier_2),'linewidth',2);
xlabel('Time (seconds)','FontSize',32)
ylabel('Amplitude','FontSize',32)

%% concatenating

trunc_sample_num  = round((m-1)*d*Fs);
remain_sample_num = L - trunc_sample_num;

concat_matrix = zeros(1,64);

for i = 1:remain_sample_num
   concat_matrix(1,i)    = inv_fourier_1(1,i);
end

for i = 1:remain_sample_num
   concat_matrix(1,i + remain_sample_num) = inv_fourier_2(1,i);
end

figure(6)
plot(concat_matrix,'linewidth',2);
xlabel('Time (seconds)','FontSize',32)
ylabel('Amplitude','FontSize',32)
```

OBJECTIVE 8

```
%% basic setup
clc;
close all;

%% initialising variables
angle   = 120;                          %input wave angle
f       = 2000;                         %input wave frequency
Fs      = 12800;                        %sampling frequency
Ts      = 1/Fs;                         %sampling interval
c       = 1500;                         %speed of sound in water
m       = 32;                           %number of element

SNR     = -2;                           %signal to noise ratio


N       = 256;                          %no of original samples
t       = (0:N-1)*Ts;                   %total time of input

lambda  = c/2000;                       %wavelength
x       = lambda/2;                     %interspace distance
d       = x*cosd(angle)/c;              %quantum delay
matrix  = zeros(N,m);                   %delayed pure signals

%% bringing the natural delay
y       = sin(2*pi*f*t);                %generating the ideal sine wave
tend    = (N-1)*Ts;                     %finding the time interspace
xaxis = linspace(0,tend,N);             %constructing the time axis

for i = 1:m                             %bringing about the delay
   matrix(:,i)=sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
```

```
SNR_weight = 10^(-1*SNR*0.05);                  %computing the noise weight
new_mat    = matrix + SNR_weight*rand(N,m);  %creating the impure matrix

%% taking the fourier transform

Fourier  = zeros(N,m);                    %initialising the fourier
NFFT     = N;                             %defining NFFT
fend     = (NFFT-1)*Fs/NFFT;              %end sample index
waxis    = linspace(0,fend,NFFT);         %creating the spacing

Fourier = fft(new_mat,NFFT);              %taking the fourier transform

figure(3)
plot(waxis,abs(Fourier));                 %plotting the fourier transform

%% establishing the delay thing
delay_column    = zeros(m,1);             %initialising the delay column
frequency_inter = Fs/N;                   %frequency index
f_mat           = zeros(1,m);             %initialising the bin row/matrix
angle_matrix    = zeros(N,181);           %initialising the angle matrix
frequency_matrix = zeros(180,9);

for sweep_angle = 1:180
   for f = 1000:250:3000
      index = 1 + f/(Fs/N);
      f_mat(1,:) = Fourier(index,:);

      for i = 1:m
         delay_column(i,1) = exp(1*1i*(i-1)*2*pi*f*(x/c)*(cosd(sweep_angle)));
      end

       frequency_matrix(sweep_angle,(f/250)-3) = abs(f_mat*delay_column);

    end
end
%% plotting the frequency vs abs
angle_axis = linspace(1,180,180);          %setting up the angle axis
sum_matrix = sum(frequency_matrix,2);  %summing column wise
```

```
plot(angle_axis,20*log10(sum_matrix),'linewidth',2);        %plotting the sum_matrix
xlabel('angle','FontSize',32)
ylabel('absolute value','FontSize',32)
```

OBJECTIVE 9

```
%% basic setup
clc;
close all;

%% initialising the variables
f          = 2000;                    %the main frequency
Fs         = 12800;                   %sampling frequency
Ts         = 1/Fs;                    %sampling interval
N          = 128;                     %number of intervals

m          = 32;                      %number of sensors
angle      = 90;                      %incoming angle
c          = 1500;                    %speed of the sound signal
lambda     = c/f;                     %wavelength of incoming signal
x          = lambda/2;                %sensor interspacing
d          = x*cosd(angle)/c;         %unit delay

SNR        = 4;                       %signal to noise ratio
SNR_weight = 10^(-1*SNR*0.05);        %SNR noise weight

t          = (0:N-1)*Ts;              %time matrix
matrix     = zeros(N,m);              %initialising noise included signal

%% bringing about the natural delay
y = sin(2*pi*f*t);                    %generating the ideal sine wave

for i = 1:m
   matrix(:,i) = sin(2*pi*f*(t-(i-1)*d));
end

%% adding the noise
new_mat = zeros(N,m);                 %initialising the noise matrix
```

```
new_mat = matrix + SNR_weight*rand(N,m);   %creating the impure matrix

%% taking the fourier transform
NFFT = N;                                  %number of frequency samples
fend = (NFFT-1)*Fs/NFFT;                    %finding the end frequency
w_axis = linspace(0,fend,NFFT);            %creating the spacing

Fourier = fft(new_mat,NFFT);               %taking the fourier transform

%% Choosing the frequency row
index = f/(Fs/NFFT)+1;
f_mat=zeros(1,m);
f_mat(1,:)=Fourier(index,:);

%% bringing the chebyshev window
cheb_window = chebwin(m,30);

for i = 1:m
   f_mat(1,i) = f_mat(1,i)*cheb_window(i,1);
end
%% bringing the delay in frequency region

angle_matrix= zeros(1,181);
delay_column = zeros(m,1);

for test_angle = 0:180
   test_d = x*cosd(test_angle)/c;          %the unit delay for test angle

   for i = 1:m
      delay_column(i,1) = exp(1*1i*2*pi*f*(i-1)*test_d);
   end

   angle_matrix(1,test_angle+1) = abs(f_mat*delay_column);
end

%% plotting the beam formed output
angle_axis = linspace(0,180,181);          %setting up the angle axis

plot(angle_axis,10*log10(angle_matrix),'linewidth',2);       %plotting the angle matrix
```

```
xlabel('angle (degrees)','FontSize',32)
ylabel('absolute value (in dB)','FontSize',32)
```

REFERENCE

1. Sonar for Practising Engineers by A.D. Waite
2. Matched Field Beamforming applied to Sonar Data by Tomas Lundström
3. Signals and systems by Hwei P. Hsu