EE6133 Multirate Digital Signal Processing
Experiment # 1

Amizhthni P R K, EE21B015

October 19, 2023

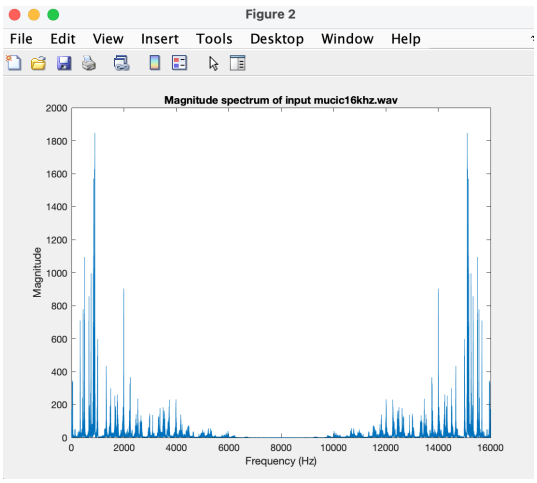
Contents

List of Figures

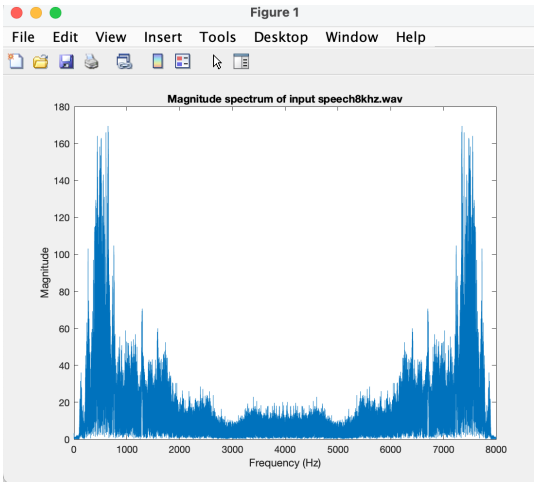
1 PART 1

We are presented with two audio samples `music16khz.wav` and `speech8khz.wav` which have been sampled at `16kHz` and `8kHz` respectively.

1.1 Magnitude Spectra of input signals



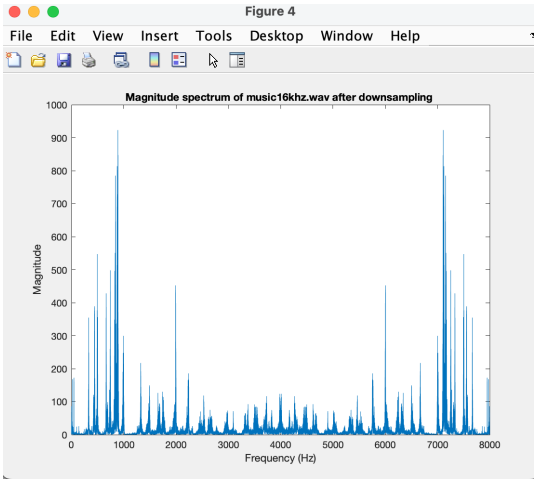
(a) Magnitude Spectrum of input `music16khz.wav`



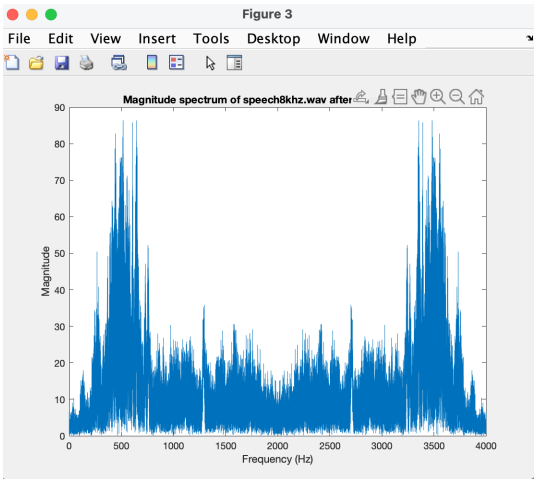
(b) Magnitude Spectrum of input `speech8khz.wav`

Figure 1: Input Magnitude Spectra

1.2 Magnitude Spectra of input signals after downsampling



(a) Magnitude Spectrum of input `music16khz.wav` after downsampling



(b) Magnitude Spectrum of input `speech8khz.wav` after downsampling

Figure 2: Input Magnitude Spectra after downsampling

1.3 Matlab code for Part 1

```
%Reading the music files

[speech,Fs_speech]= audioread('/Applications/speech8khz.wav');
[music,Fs_music]= audioread('/Applications/music16khz.wav');

%Plotting the magnitude spectrum of the audio files
Fs_speech
Fs_music
NFFT = length(speech);
Y = fft(speech,NFFT);
magnitudeY = abs(Y);
freq=(0:NFFT-1) * (Fs_speech) / NFFT;
figure(1);
plot(freq,magnitudeY)
title("Magnitude spectrum of input speech8khz.wav");
xlabel('Frequency (Hz)');
ylabel('Magnitude');


NFFT1 = length(music);
Y1 = fft(music,NFFT1);
magnitudeY1 = abs(Y1);
freq1=(0:NFFT1-1) * (Fs_music) / NFFT1;
figure(2);
plot(freq1,magnitudeY1)
title("Magnitude spectrum of input mucic16khz.wav");
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%Plotting the magnitude spectrum of the audio files after downsampling them

x1=downsample(speech,2);
NFFT2 = length(x1);
Y2 = fft(x1,NFFT2);
magnitudeY2 = abs(Y2);
freq2=(0:NFFT2-1) * (Fs_speech/2) / NFFT2;
figure(3);
plot(freq2,magnitudeY2)
title("Magnitude spectrum of speech8khz.wav after downsampling ");
xlabel('Frequency (Hz)');
ylabel('Magnitude');

x2=downsample(music,2);
NFFT3 = length(x2);
Y3 = fft(x2,NFFT3);
magnitudeY3 = abs(Y3);
freq3=(0:NFFT3-1) * (Fs_music/2) / NFFT3;
figure(4);
plot(freq3,magnitudeY3)
```

```
title("Magnitude spectrum of music16khz.wav after downsampling ");
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%Listening to the audio samples
sound(speech,Fs_speech)
pause(10)
sound(music,Fs_music)
pause(10)
sound(x1,Fs_speech/2)
pause(10)
sound(x2,Fs_music/2)
pause(10)


```

Listing 1: Code snippet used in the experiment for Part 1

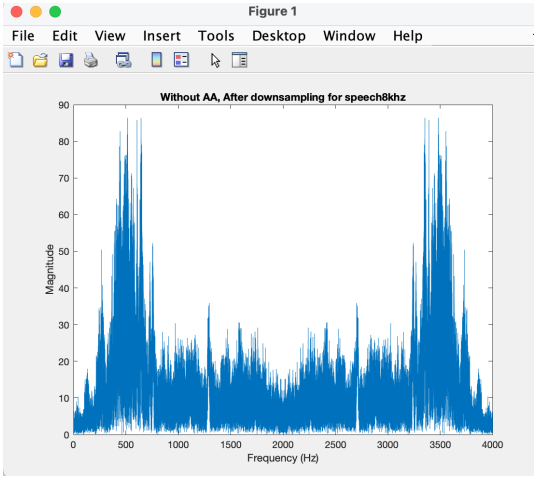
1.4 Observations

- In the plot for the magnitude spectrum of `music16khz.wav`, the frequency of 16KHz corresponds to 2π radians, since the sampling rate is 16000 Hz.
- For the same audio post downsampling, the frequency corresponding to 2π radians becomes 8000Hz.
- Clearly aliasing occurs in both the signals during downsampling. We observe a lot of unnecessary samples especially around π radians in both the audio inputs.
- This is even more so in `speech8khz.wav` since there are more samples such that $\omega > \pi$. This distorts the signal further.

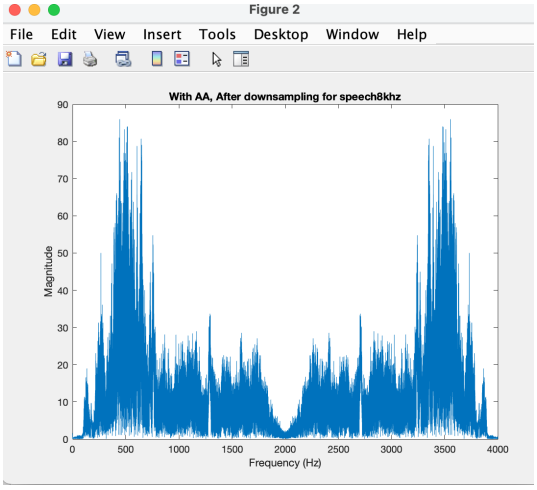
2 PART 2

We are presented with two audio samples `music16khz.wav` and `speech8khz.wav` which have been sampled at `16kHz` and `8kHz` respectively. We first construct a Low Pass Filter with a cut off frequency $\omega_c = 0.5\pi$ and use this as an Anti Aliasing Filter and then downsample our signal.

2.1 Magnitude Spectra comparison plots - With and Without Anti-Aliasing Filtering

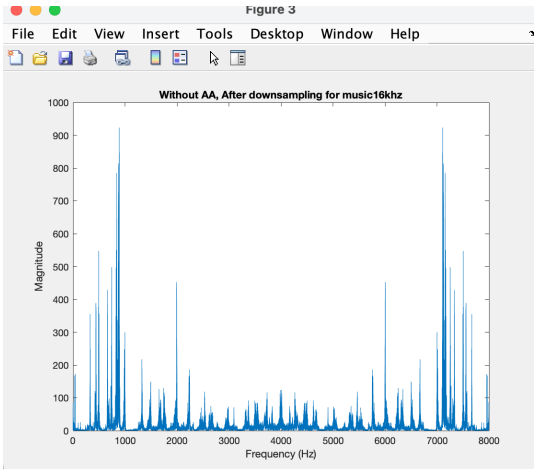


(a) Magnitude Spectrum of input `speech8khz.wav` after downsampling without the Anti Aliasing Filter

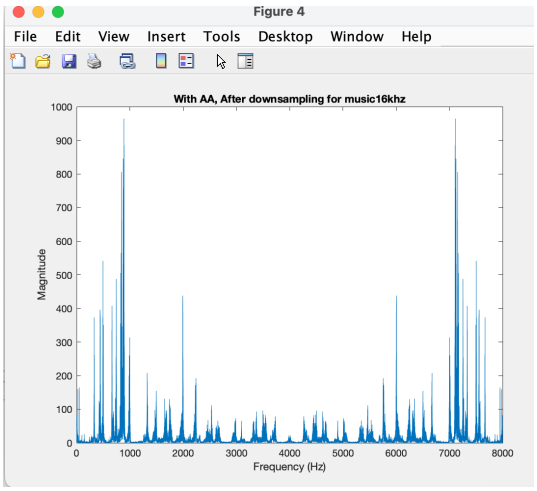


(b) Magnitude Spectrum of input `speech8khz.wav` after downsampling with the Anti Aliasing Filter

Figure 3: Comparing the working of the Anti Aliasing Filter for the input `speech8khz.wav`



(a) Magnitude Spectrum of input `music16khz.wav` after downsampling without the Anti Aliasing Filter



(b) Magnitude Spectrum of input `music16khz.wav` after downsampling with the Anti Aliasing Filter

Figure 4: Comparing the working of the Anti Aliasing Filter for the input `music16khz.wav`

2.2 Filter Design

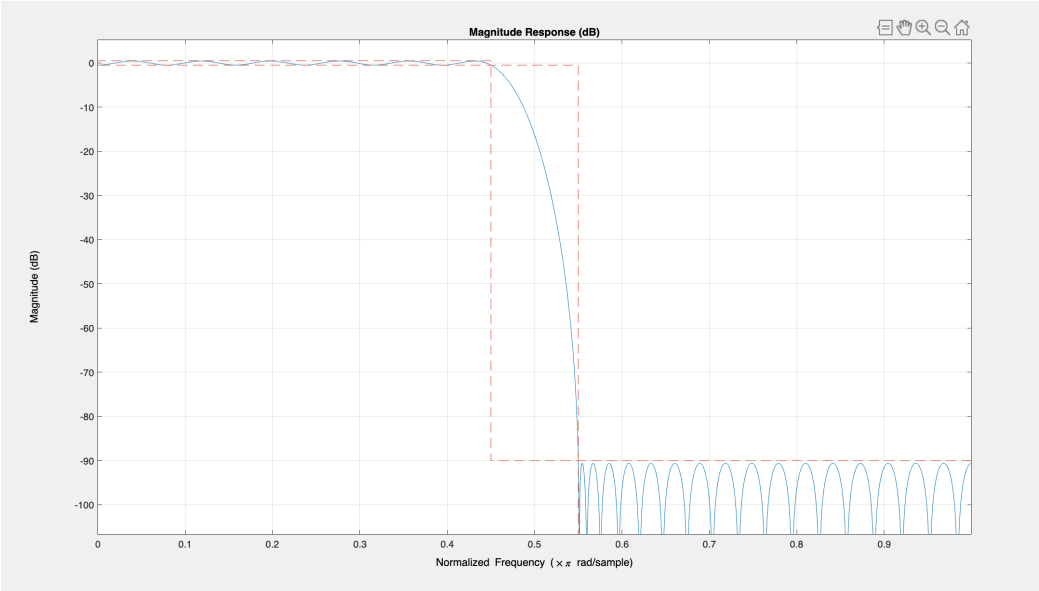


Figure 5: The magnitude response of the Filter thus designed

We use the below command to design the filter

$$Hd = fdesign.lowpass('Fp, Fst, Ap, Ast', 0.45, 0.55, 1, 90);$$

F_p is the Pass band cut off frequency. Note that we are giving the proportional constants as parameters. This leads to normalization and gives us the desired output.
F_s is the Stop band cut off frequency.

F_p=**0.45**, We take the proportional values keeping normalisation in mind.
F_s=**0.55**
δ_p=A_p=**1**
A_s=**60dB**, Matlab creates a filter with a specific δ_s corresponding to the A_s value. Greater the A_s, better the suppression in the stop band.

We can also use

$$Hd = fdesign.lowpass('Fp, Fst, Ap, Ast', 1800, 2200, 1, 90, 8000);$$

Where
$1800 = \frac{4.5\times\pi}{2\times\pi} \times 8000$
$2200 = \frac{5.5\times\pi}{2\times\pi} \times 8000$

This gives us a Low Pass Filter with a cut off frequency $\omega_c = 0.5\pi$.
Once the equiripple filter is designed, we find the coefficients of the designed filter. Since we have designed an **FIR** filter, we do not have any Denominator terms and the second term in the filter function is simply 1. Note that the `conv()` function is applicable to just **FIR** filters while the `filter()` function is applicable to both **FIR** and **IIR** filters.

The coefficients are obtained in the `struct` datatype, which needs to be converted to `double` datatype before convolving.

2.3 Matlab code for Part 2

```
%Designinig the filter

Hd = fdesign.lowpass('Fp,Fst,Ap,Ast',0.45,0.55,1,90);
d = design(Hd,'equiripple');

%fvtool(d)

%To see the propertes of the filter

lpFIR = design(Hd,'equiripple','SystemObject',true);
FIRcost = cost(lpFIR);

%Reading the music files
[speech,Fs_speech]= audioread('/Applications/speech8khz.wav');
[music,Fs_music]= audioread('/Applications/music16khz.wav');


x1=downsample(speech,2);
NFFT1 = length(x1);
Y1 = fft(x1,NFFT1);
magnitudeY1 = abs(Y1);
figure(1);
freq1=(0:NFFT1-1) * (Fs_speech/2) / NFFT1;
plot(freq1,magnitudeY1);
title('Without AA, After downsampling for speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


b1=coeffs(d);
C= struct2cell(b1);
A= cell2mat(C);
down=filter(A,1,speech);  %or use the conv() function
x=downsample(down,2);
NFFT = length(x);
Y = fft(x,NFFT);
magnitudeY = abs(Y);
freq=(0:NFFT-1) * (Fs_speech/2) / NFFT;
figure(2);
plot(freq,magnitudeY)
title('With AA, After downsampling for speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');



[y,Fs]= audioread('/Applications/music16khz.wav');

x1_m=downsample(music,2);
```

```
NFFT1_m = length(x1_m);
Y1_m = fft(x1_m,NFFT1_m);
magnitudeY1_m = abs(Y1_m);
figure(3);
freq1_m=(0:NFFT1_m-1) * (Fs_music/2) / NFFT1_m;
plot(freq1_m,magnitudeY1_m);
title('Without AA, After downsampling for music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');



C_m= struct2cell(b1);
A_m= cell2mat(C_m);
down_m=filter(A_m,1,music);  %or use the conv() function
x_m=downsample(down_m,2);
NFFT_m = length(x_m);
Y_m = fft(x_m,NFFT_m);
magnitudeY_m = abs(Y_m);
freq_m=(0:NFFT_m-1) * (Fs_music/2) / NFFT_m;
figure(4);
plot(freq_m,magnitudeY_m)
title('With AA, After downsampling for music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%Listening to the audio samples
sound(x1,Fs_speech/2)
pause(10)
sound(x,Fs_speech/2)
pause(10)
sound(x1_m,Fs_music)
pause(10)
sound(x_m,Fs_music/2)
pause(10)


```

Listing 2: Code snippet used in the experiment for Part 2

2.4 Observations

- The performance of the filter depends on the size of the ripple bands δ_p and δ_s. We can either decrease A_p or increase A_s to improve the performance. Which in other words is to decrease either of δ_p and δ_s
- IIR filters (in particular biquad filters) are used in applications (such as audio signal processing) where phase linearity is not a concern.
- However, the use of minimum-phase and multirate designs can result in FIR filters comparable to IIR filters in terms of group delay and computational efficiency. We

have thus developed an **FIR** Filter in this experiment.

- The downsampled signal without the AA filter has portions that sound like different sounds spoken simultaneously, while the one with AA filter maintains the sounds separately. However, the distinction between sounds themselves is not as clear as in the original.

3 PART 3

3.1 Matlab code for Part 3

We design a filter with pass band at 0.22π and stop band at 0.28π. This is an LPF with cutoff frequency at 0.25π. The transition from Pass to stop band is steeper and more abrupt in this case than the filter in Part2.

```
Hd = fdesign.lowpass('Fp,Fst,Ap,Ast',0.22,0.28,0.00001,90);
d = design(Hd,'equiripple');
freqz(d)
```

Listing 3: Code snippet used in the experiment for Part 3

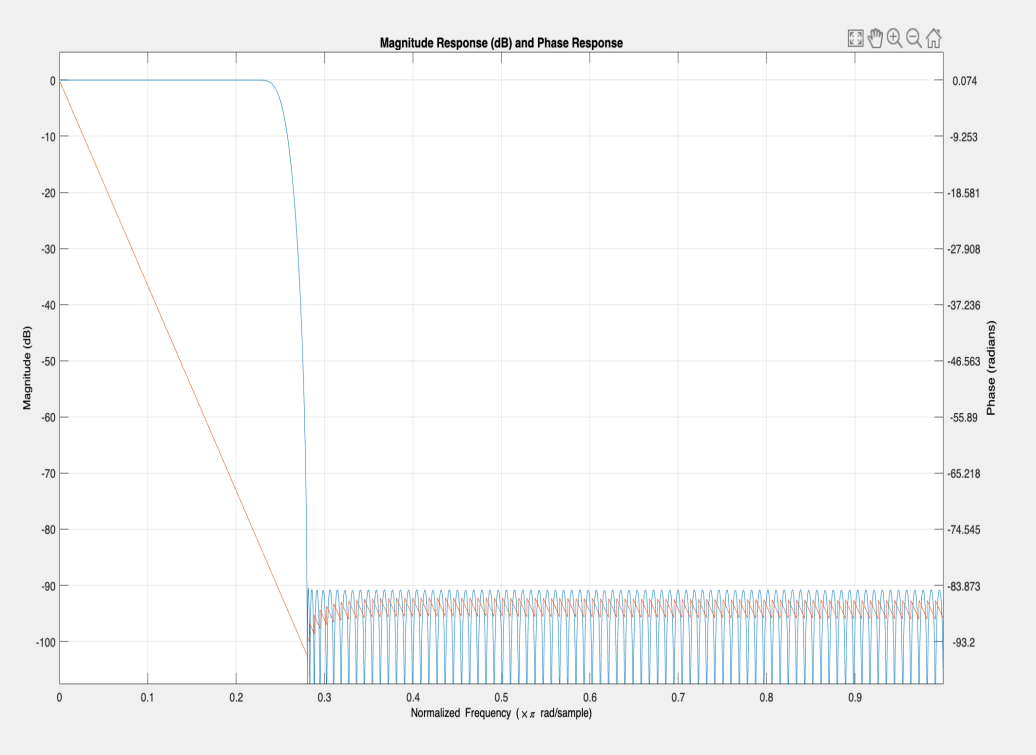
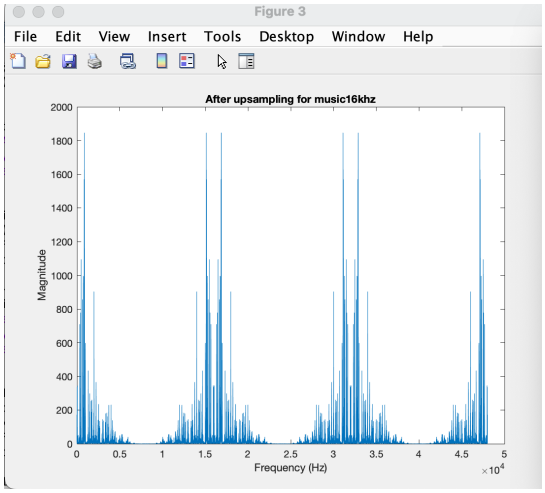
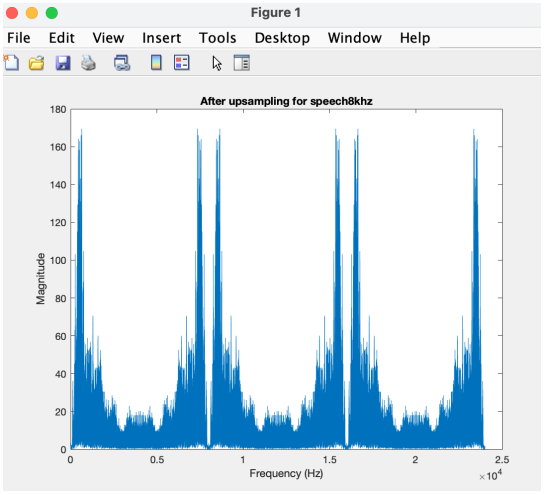


Figure 6: The magnitude response of the Filter thus designed

4 PART 4

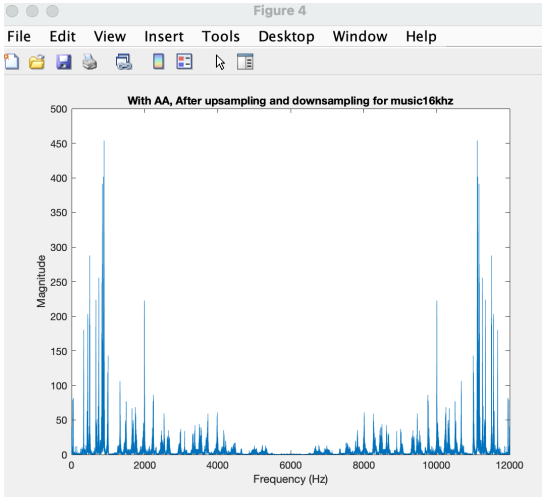


(a) Magnitude Spectrum of input `music16khz.wav` after upsampling

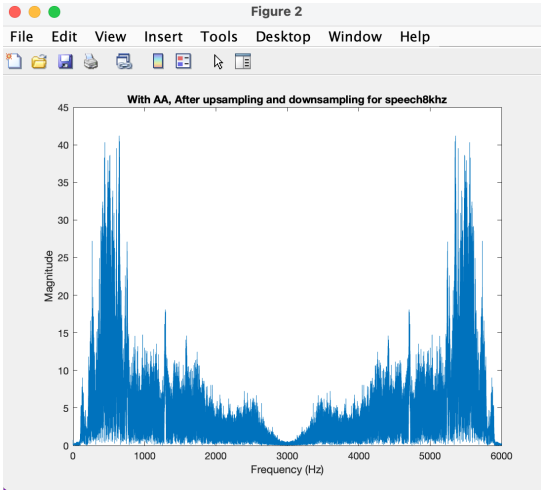


(b) Magnitude Spectrum of input `speech8khz.wav` after upsampling

Figure 7: The output after upsampling both the inputs



(a) Magnitude Spectrum of input `music16khz.wav` after downsampling by 4/3 and anti-aliasing



(b) Magnitude Spectrum of input `speech8khz.wav` after downsampling by 4/3 and anti-aliasing

Figure 8: The output after upsampling by 3, anti aliasing and downsampling by 4 for both the inputs

```
1
2  %Reading the music files
3
4  [speech,Fs_speech]= audioread('/Applications/speech8khz.wav');
5  [music,Fs_music]= audioread('/Applications/music16khz.wav');
6
7  %upsampling
8  x=upsample(speech,3);
```

```
NFFT0 = length(x);
Y0 = fft(x,NFFT0);
magnitudeY0 = abs(Y0);
figure(1);
freq=(0:NFFT0-1) * (Fs_speech*3) / NFFT0;
plot(freq,magnitudeY0);
title('After upsampling for speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%filter design
Hd = fdesign.lowpass('Fp,Fst,Ap,Ast',0.22,0.28,1,90);
d = design(Hd,'equiripple');
b1=coeffs(d);
C= struct2cell(b1);
A= cell2mat(C);

%downsampling
down=filter(A,1,x);
x1=downsample(down,4);
NFFT = length(x1);
Y = fft(x1,NFFT);
magnitudeY = abs(Y);
figure(2);
freq1=(0:NFFT-1) * (3*Fs_speech/4) / NFFT;
plot(freq1,magnitudeY);
title('With AA, After upsampling and downsampling for speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%upsampling
x_m=upsample(music,3);
NFFT0_m = length(x_m);
Y0_m = fft(x_m,NFFT0_m);
magnitudeY0_m= abs(Y0_m);
figure(3);
freqm=(0:NFFT0_m-1) * (Fs_music*3) / NFFT0_m;
plot(freqm, magnitudeY0_m);
title('After upsampling for music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%downsampling
down_m=filter(A,1,x_m);
x1_m=downsample(down_m,4);
NFFT_m = length(x1_m);
Y_m= fft(x1_m,NFFT_m);
magnitudeY_m = abs(Y_m);
figure(4);
freq1=(0:NFFT_m-1) * (3*Fs_music/4) / NFFT_m;
```

```
plot(freq1,magnitudeY_m);
title('With AA, After upsampling and downsampling for music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

Listing 4: Code snippet used in the experiment for Part 4

4.1 Observations

- The periodicity of the samples after upsampling becomes $\frac{2*\pi}{L}$ i.e. the magnitude spectrum has 2 extra images of the original spectrum.
- The upsampled signal sounds "sharper" than the original - that is, it is filled with "high pitch" noises.
- It is also less louder than the original. However, the distinctive sounds are separate and their distinction is maintained clearly, just like the original.
- The upsampled and Anti Aliasing filtered and downsampled output is much like the original audio but the distinctive sounds are harder to distinguish.

5 PART 5

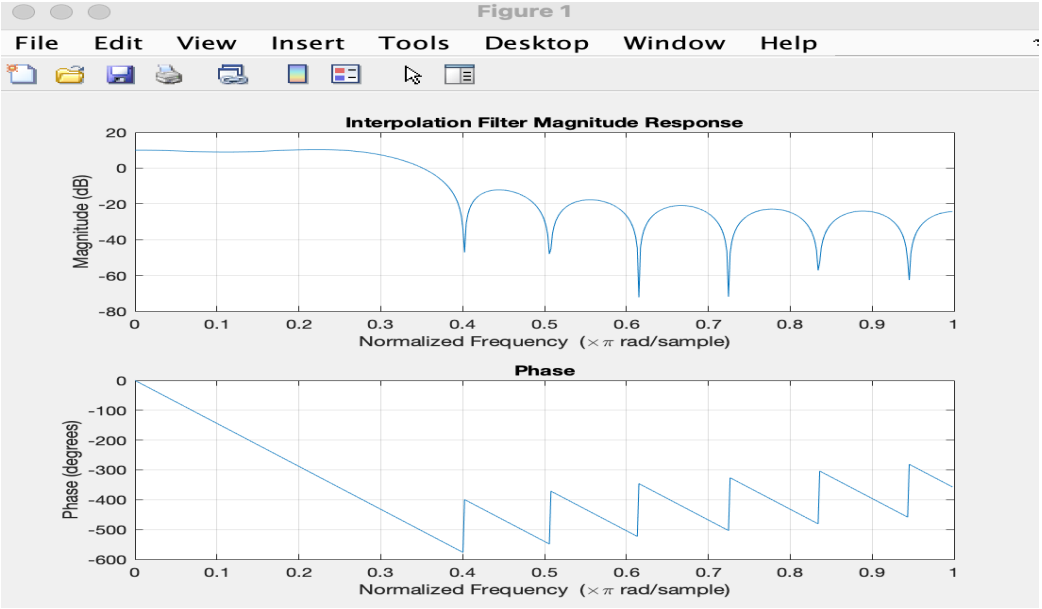


Figure 9: The magnitude response of the Interpolation Filter

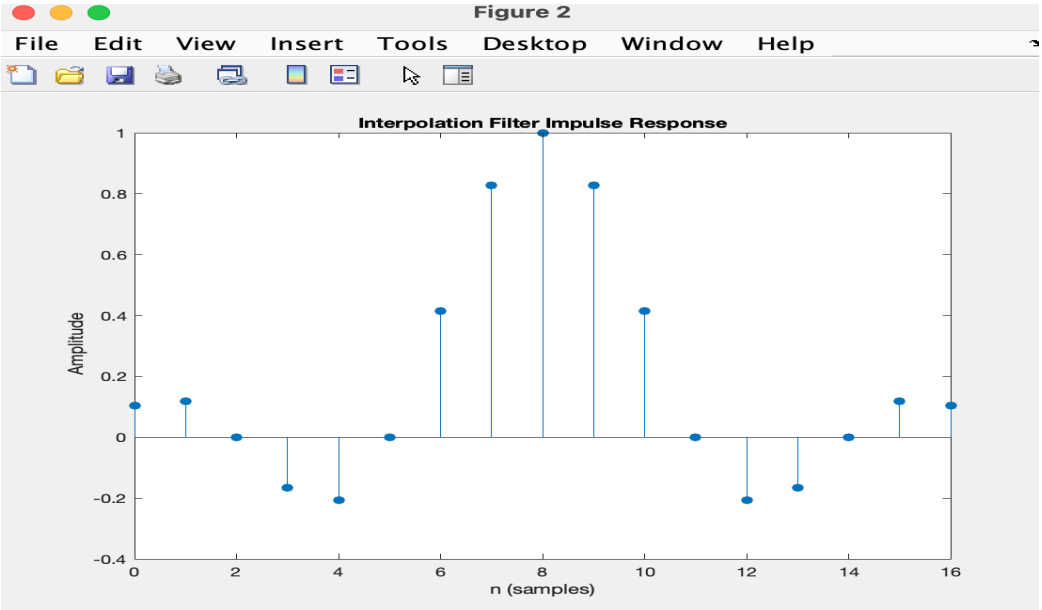
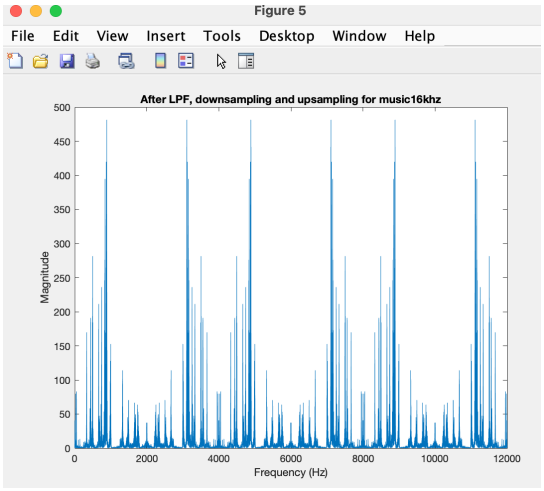
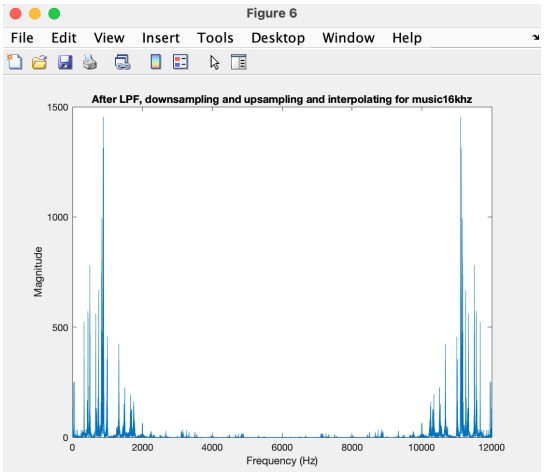


Figure 10: The impulse response of the Interpolation Filter

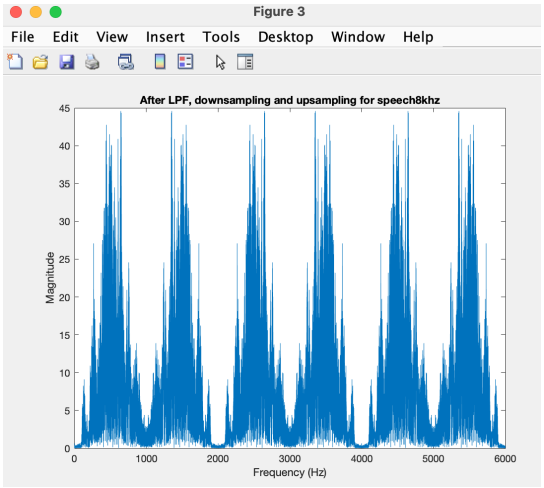


(a) Magnitude Spectrum of input `music16khz.wav` after downsampling by 4/3 and anti-aliasing without Interpolation filter

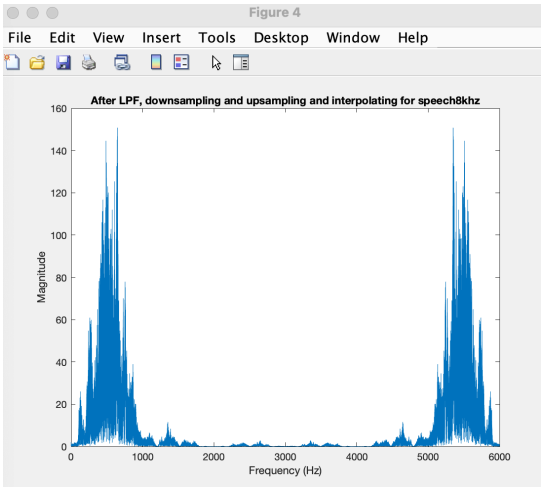


(b) Magnitude Spectrum of input `music16khz.wav` after downsampling by 4/3 and anti-aliasing followed by Interpolation filter

Figure 11: The output with and without Interpolation filtering for `music16khz.wav`



(a) Magnitude Spectrum of input `speech8khz.wav` after downsampling by 4/3 and anti-aliasing without Interpolation filter



(b) Magnitude Spectrum of input `speech8khz.wav` after downsampling by 4/3 and anti-aliasing followed by Interpolation filter

Figure 12: The output with and without Interpolation filtering for `speech8khz.wav`

```
interpolation = intfilt(3,3,1);
figure(1);
freqz(interpolation, 1);
title('Interpolation Filter Magnitude Response');
figure(2);
impz(interpolation, 1);
title('Interpolation Filter Impulse Response');



[speech,Fs_speech]= audioread('/Applications/speech8khz.wav');
```

```
[music,Fs_music]= audioread('/Applications/music16khz.wav');

Hd = fdesign.lowpass('Fp,Fst,Ap,Ast',0.22,0.28,1,90);
d = design(Hd,'equiripple');
b1=coeffs(d);
C= struct2cell(b1);
A= cell2mat(C);
lpf=filter(A,1,speech);
down=downsample(lpf,4);
up=upsample(down,3);
NFFT0 = length(up);
Y0 = fft(up,NFFT0);
magnitudeY0 = abs(Y0);
figure(3)
freq1=(0:NFFT0-1) * (3*Fs_speech/4) / NFFT0;
plot(freq1,magnitudeY0)
title('After LPF, downsampling and upsampling for speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


final=conv(interpolation,up);
NFFT1 = length(final);
Y1 = fft(final,NFFT1);
magnitudeY1 = abs(Y1);
figure(4)
freq=(0:NFFT1-1) * (3*Fs_speech/4) / NFFT1;
plot(freq,magnitudeY1)
title('After LPF, downsampling and upsampling and interpolating for
↪  speech8khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');



lpf_m =filter(A,1,music);
down_m =downsample(lpf_m ,4);
up_m =upsample(down_m ,3);
NFFT0_m  = length(up_m );
Y0_m  = fft(up_m ,NFFT0_m );
magnitudeY0_m  = abs(Y0_m );
figure(5)
freq1_m=(0:NFFT0_m-1) * (3*Fs_music/4) / NFFT0_m;
plot(freq1_m,magnitudeY0_m)
title('After LPF, downsampling and upsampling for music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


final_m =conv(interpolation,up_m );
```

```
NFFT1_m  = length(final_m );
Y1_m  = fft(final_m ,NFFT1_m );
magnitudeY1_m  = abs(Y1_m );
freq_m=(0:NFFT1_m-1) * (3*Fs_music/4) / NFFT1_m;
figure(6)
plot(freq_m,magnitudeY1_m )
title('After LPF, downsampling and upsampling and interpolating for
↪  music16khz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


```

Listing 5: Code snippet used in the experiment for Part 5

5.1 Observations

- The support has been considered from **0 to 16** in this case
- We observe that the sound in **part 4** is *better* than that in part 5. The latter sounding rather suppressed or muted.
- This happens because there are very few spectral components around π. This can be attributed to the fact that downsampling is performed before upsampling.
- We lose the data from $\frac{\pi}{4}$ to π right after we pass the audio through the AA filter.
- Thus we conclude that Part 4 method is better i.e. we upsample and then downsample using an Anti Aliasing filter.