ECE 3141 Information and Networks: Downsampling

Edmund Lee Wai Loon, Tan Jin Chun, Ng Zhan Lok

School of Electrical and Computer System Engineering

Monash University Malaysia

Definition Of Downsampling

Downsampling can be defined in two ways, in terms of audio and in terms of images.

Audio Downsampling: Defined as sampling an audio signal with a sampling rate below the Nyquist rate, which causes aliasing (distortion) in the signals. Downsampling and aliasing are used interchangeably when referring to audio. A few reasons downsampling is done is to decrease the bit rate to transmit over a band-limited channel or to convert to more limited audio formats.[1]

Image Downsampling: Defined as reducing the spatial resolution (number of pixels) of an image while the 2-D representation of the image is still maintained. Image downsampling will look similar to the original image but with lesser detail and might be indistinguishable if the reduced resolution is too low. A few reasons downsampling is done is to decrease the color depth of the image and to cater to the demands of the storage/transmission of the image.[1]

Audio Downsampling (Aliasing)

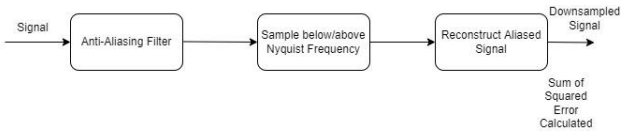


Diagram 1: Block diagram of the process of audio downsampling

When audio is sampled, it goes through a few essential steps. These steps can be seen in Diagram 1 where a sampling rate is first chosen, then the signal will be sampled and then reconstructed (Anti-Aliasing Filter block is optional/added when aliasing could occur). To avoid aliasing, a signal needs to be sampled above a minimum rate which is known as the Nyquist Rate (twice the bandwidth of a signal).[2] When aliased, replicated spectrums during sampling will overlap with the original spectrum, causing the reconstructed signal to be distorted. An audio signal is sampled in MATLAB with 2 different sampling rates, one much lower than the sampling frequency, fs, given by MATLAB (44100 Hz) and another using that exact sampling frequency (MATLAB sampling frequency, fs, has already been optimized to avoid downsampling of signals) to illustrate the effects of aliasing.

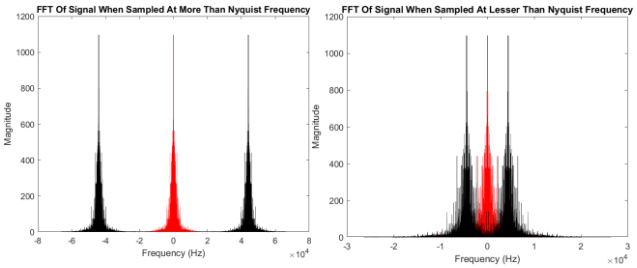


Figure 1 & 2: FFT of Signal when sampled at the original sampling rate and downsampled by an order of 10

Figure 1 and 2 are the FFT (Fast Fourier Transform) of the signal sampled at the original sampling rate, fs, and downsampled by an order of 10 (0.1 of the original sampling rate) respectively. The FFT represents the spectrum of frequencies present in a particular signal.[3] The red spectrums of both figures represent the original spectrum while the black spectrums represent the replicated spectrums. As can be seen from the figures, there is no overlapping between the spectrums in figure 1 while the spectrums overlap in figure 2. This is due to downsampling causing aliasing to occur in figure 2 and when reconstruction of the signal is done, it will contain not only the original spectrum but also the overlapping section of the replicated spectrum.

Anti-Aliasing Filter (AAF)

Downsampling is sometimes unavoidable, that is why there is a technique, among others, called an anti-aliasing filter (AAF), which is a low pass filter with cut-off frequency (at most) the Nyquist frequency (which should not be confused with the Nyquist rate), which is half the chosen sampling rate (fs/2).[4] This filter is applied before sampling is done so that aliasing will not occur (or allowed to occur a little depending on requirements). AAF is used to prevent aliasing in audio signals by controlling the frequency range to prevent the maximum frequency of the signal from exceeding the Nyquist frequency when the signal is sampled but this comes at a cost where some of the higher frequencies present in the original signal will be lost (filtered). Another common application of an AAF is to remove noise.

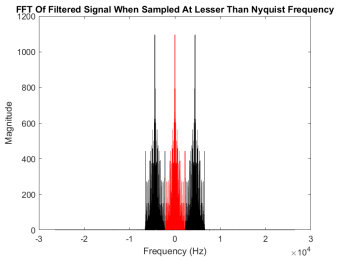


Figure 3: FFT of signal when sampled at less than Nyquist Frequency

When an AAF is applied, based on setting the maximum cut-off frequency as π/L rad/sample where L represents the order of downsampling, it can be seen that very minimal aliasing occurs as can be seen from figure 3.

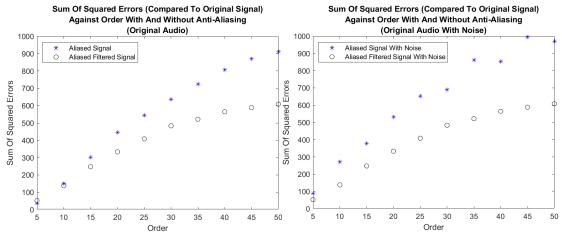


Figure 4 & 5: Sum of Squared Errors Against Order with and without Anti-Aliasing filters

When the order of downsampling was varied and the sum of squared errors was calculated between the original signal and both the aliased signal without anti-aliasing and the aliased signal with anti-aliasing. The results show a clear trend where the anti-aliased signal would provide significant improvement as the order of downsampling increases. When noise was added to the original signal before anti-aliasing and sampling, the improvement became even more clear due to the anti-aliasing filter also removing noise from the signal. However, at lower orders, the improvement is not as significant which shows that it is only worth using an anti aliasing filter with a high order downsample.

Image Downsampling (Resolution Reduction)

When images are downsampled, the resolution suffers as lesser pixels are available to present the details that are in the image. This causes the image to be less clear but still recognisable at low downsampling orders,but when the downsampling order becomes too high, the image will become indistinguishable. [7]

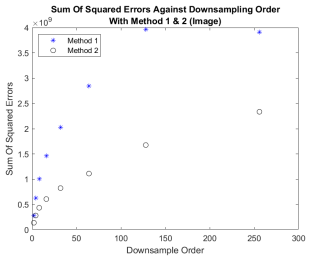


Figure 6: Sum of Squared Errors Against Downsampling Order with Method 1 & 2 (Image)

An image is downsampled in MATLAB using two different methods. The first method is where every nth pixel is taken from each row, where n is the downsampling order for the columns while every mth pixel is taken from each column, where m is the downsampling order for the rows. The second method is where the image is split into equal n x m boxes, where n is the downsampling order for the columns while m is the downsampling order for the rows. In this simulation, the downsampling order is set as n = m to simplify matters. The sum of squared errors was calculated by comparing each downsampled pixel with each of its corresponding n x m pixels in the original image. The downsample order was varied and as can be seen from figure 6, the second method performs much better in representing the original image when downsampled, but similar to the anti aliasing filter, the improvement in using method 2 becomes more significant when the downsample order increases.

Conclusion

Downsampling causes aliasing in audio signals and lower resolution in images.Audio signals should be sampled above the Nyquist rate and images should not be downsampled. Downsampling is unavoidable due to needing to decrease bitrate for audio transmission over a band-limited channel, or to reduce size of image due to limited storage space.Downsampling would inherently lower the overall quality of our audio/image. From our plots above, we concluded that AAF is a very effective way to improve on the quality of an audio signal when downsampling as it prevents the Nyquist frequency to be exceeded, while it is better to average pixel values rather than taking a pixel at fixed intervals for an image when downsampling.

References

[1]. “Definition of downsample,” PCMAG. [Online]. Retrieved From: https://www.pcmag.com/encyclopedia/term/downsample.[Accessed 24 April 2022]

[2]. IRCAM, AudioSculpt 3.0 User Manual. Retrieved from: http://support.ircam.fr/doc s/AudioSculpt/ 3.0/co/Aliasing.html

[3]. “ *Fast Fourier Transformation FFT - Basics*.” NTI Audio. [Online]. Retrieved From: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft#:~:text=The%20%22Fast%20 Fourier%20Transform%22%20. [Accessed 24 April 2022]

[4]. Author Cadence PCB Solutions,“Anti-aliasing filter design and applications in sampling,” Cadence, 17-Mar-2022. [Online]. Retrieved From: https://resources.pcb.cadence.com/blog/2020-anti-aliasing-filter-design-and-applications-in-sampling. [Accessed 22 April 2022]

[5]. Neeman, Sol Ph.D., "Using MATLAB to Illustrate the 'Phenomenon of Aliasing'" (1995). Engineering Studies Faculty Publications and Creative Works. Paper 8. Retrieved from: https://scholarsarchive.jwu.edu/cgi/viewcontent.cgi?article=1006&context=engineering_fac

[6]. Abdou Youssef, “ Abdou Youssef Audio and Image Search”. [Online]. Retrieved From: https://www2.seas.gwu.edu/~ayoussef/publications.html#MultimediaSearch. [Accessed 24 April 2022]

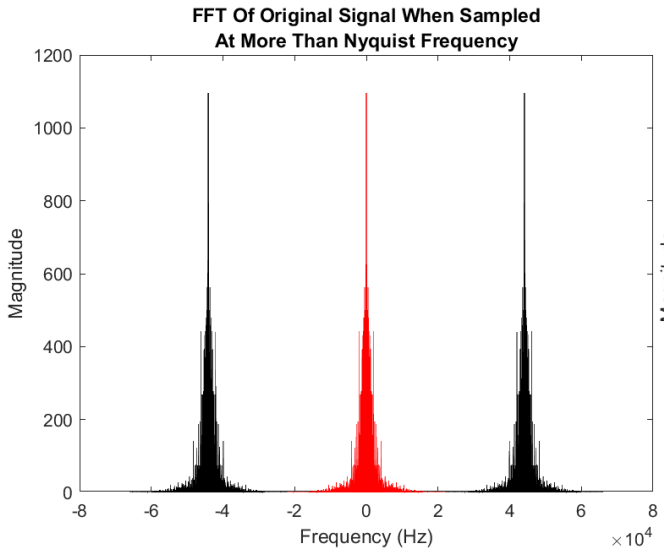
[7]. “Abdou Youssef,” Abdou Youssef Image Processing. [Online]. Retrieved From: https://www2.seas.gwu.edu/~ayoussef/publications.html#imageprocecccing. [Accessed 24 April 2022]

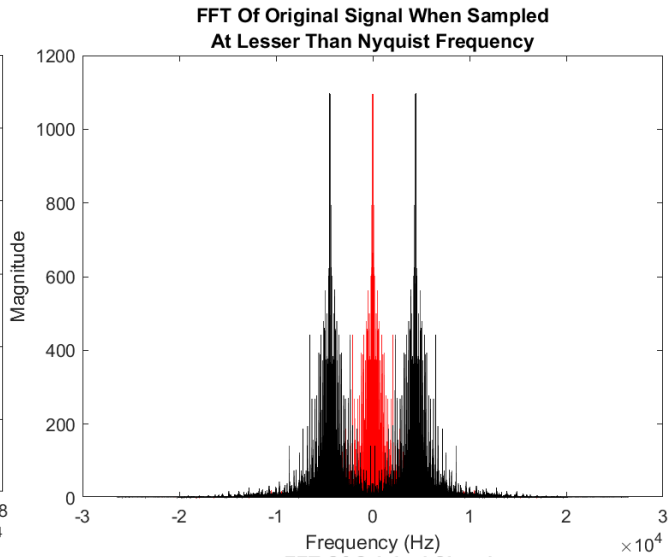
[8]. “What is anti-aliasing and which type should you use?,” *GPU Mag*, 26-Oct-2021. [Online]. Retrieved From: https://www.gpumag.com/anti-aliasing/.

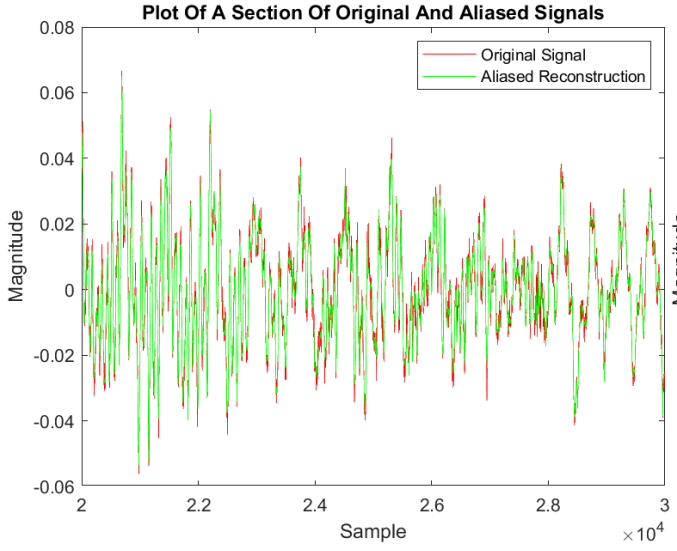
[9]. Trentacoste, Matthew, Mantiuk, Rafal, & Heidrich, Wolfgang, Blur-Aware Image Downsampling. Computer Graphics Forum, 2011, pp. 573–582.

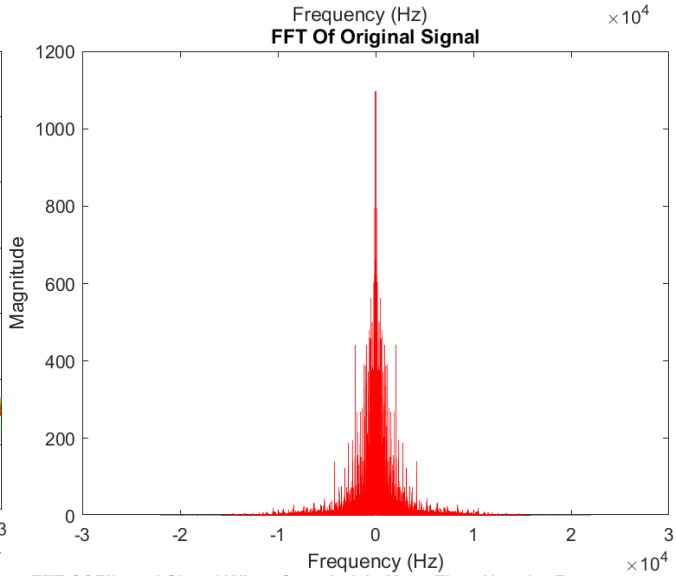
Appendix

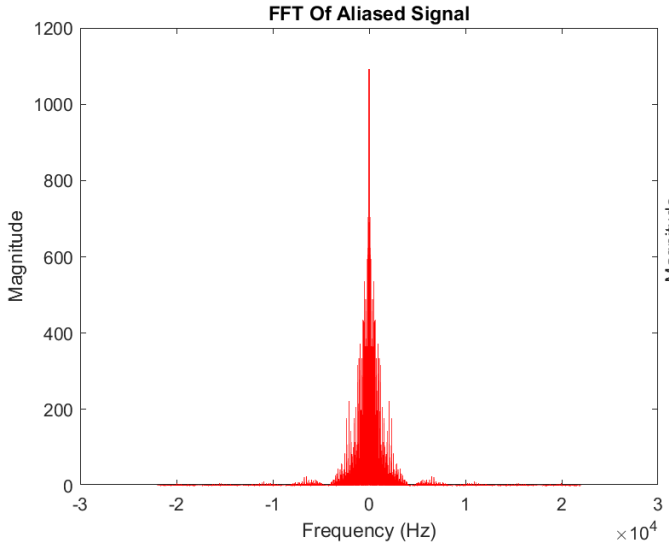
Figures

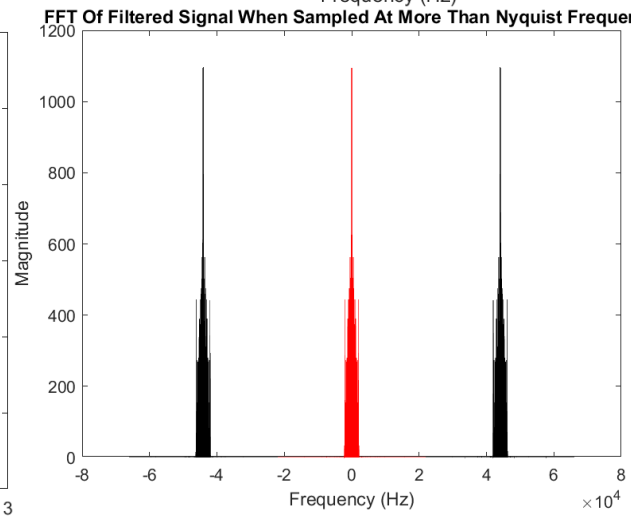


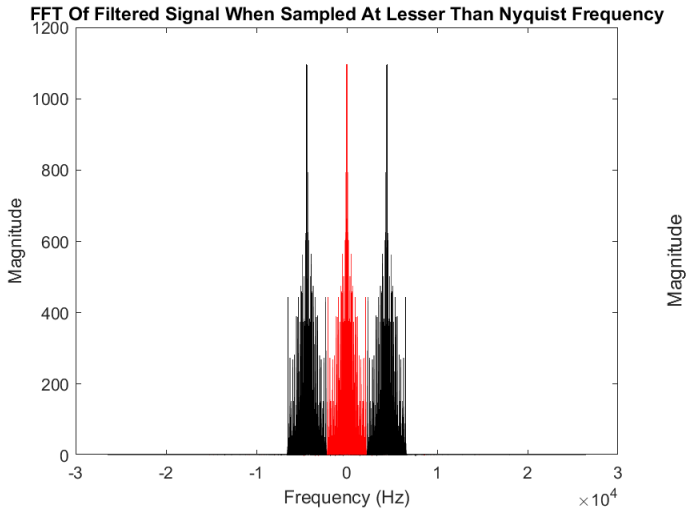


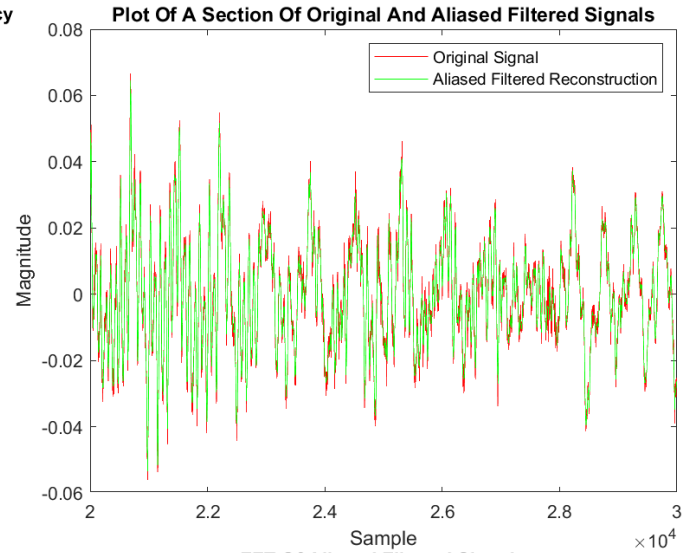


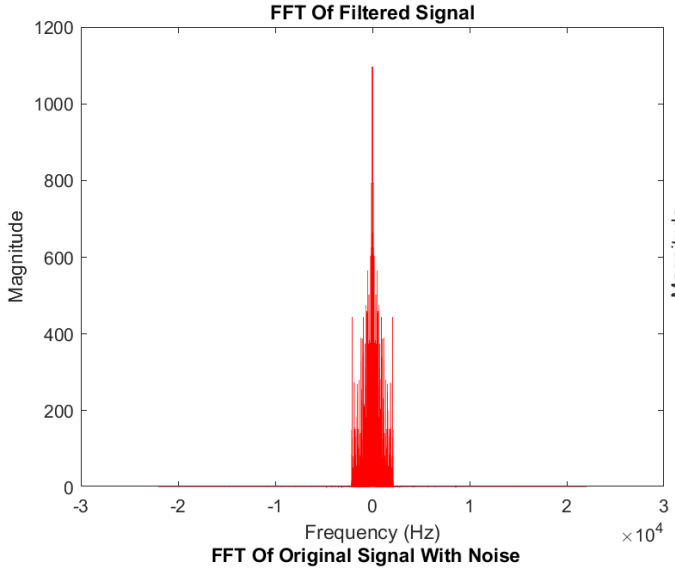


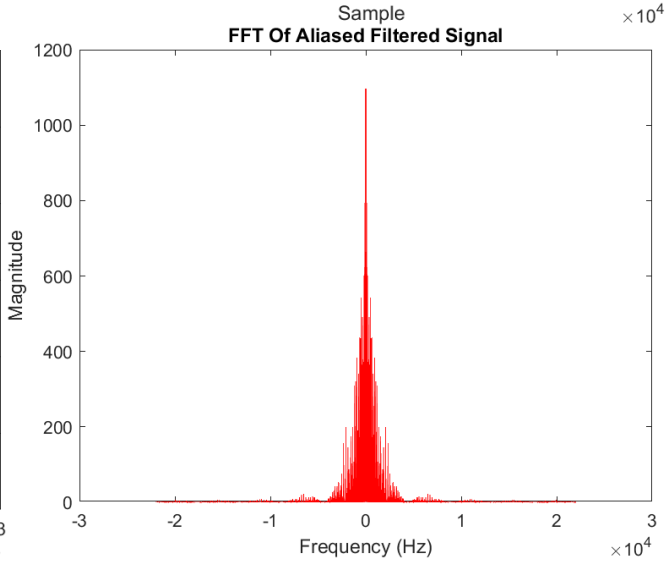


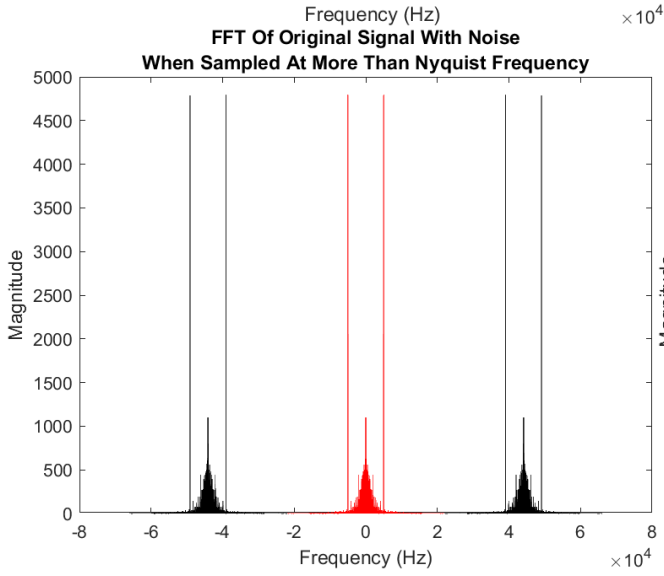


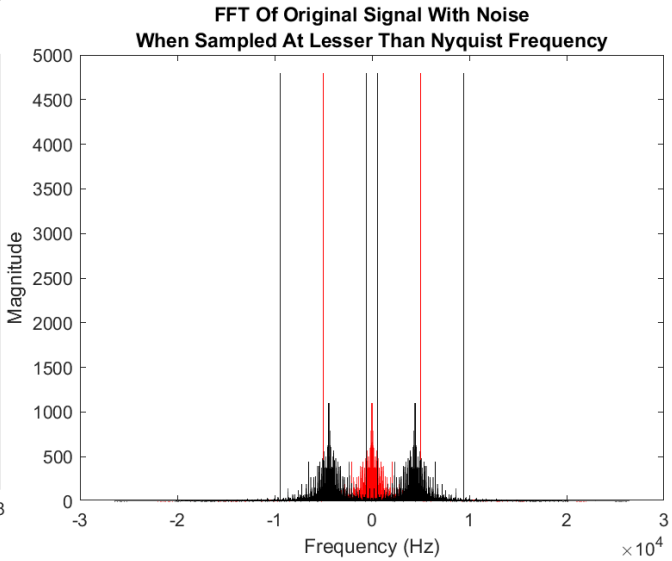

FFT Of Filtered Signal When Sampled At Lesser Than Nyquist Frequency
Magnitude
Frequency (Hz)
×10^4

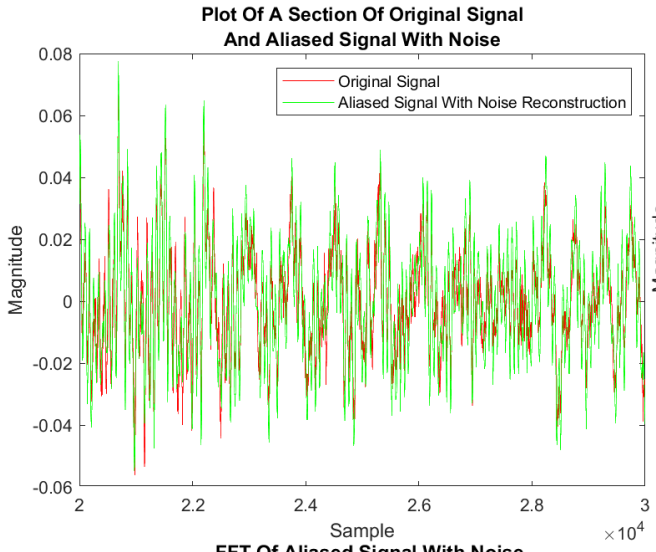

Plot Of A Section Of Original And Aliased Filtered Signals
Original Signal
Aliased Filtered Reconstruction
Magnitude
Sample
×10^4

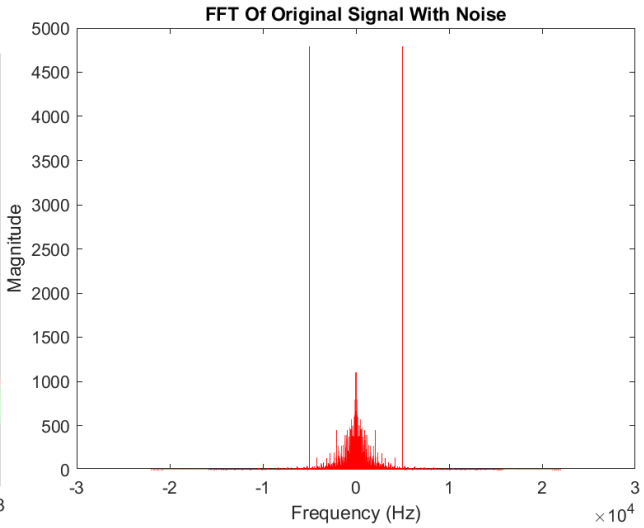

FFT Of Filtered Signal
Magnitude
Frequency (Hz)
×10^4

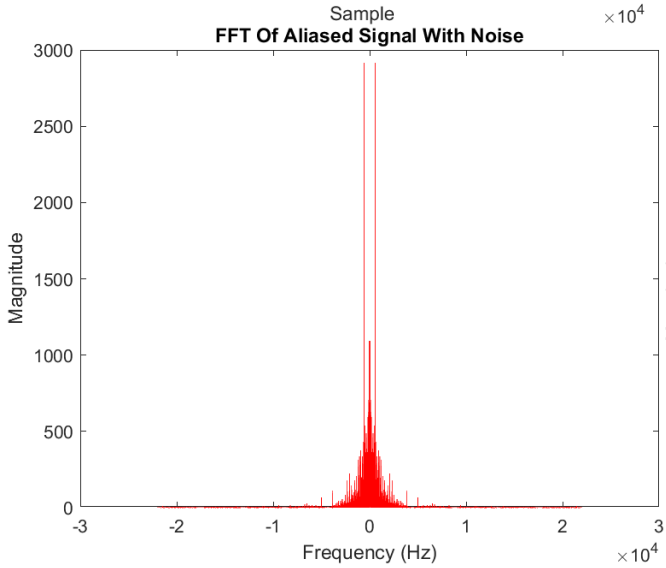

FFT Of Aliased Filtered Signal
Magnitude
Frequency (Hz)
×10^4

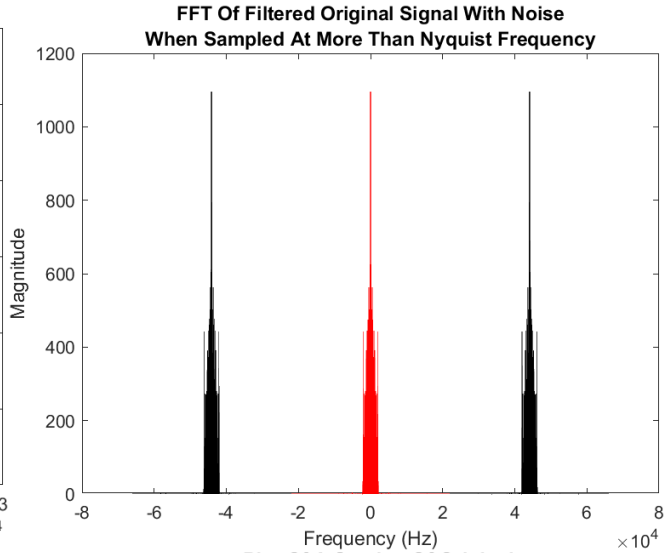

FFT Of Original Signal With Noise
When Sampled At More Than Nyquist Frequency
Magnitude
Frequency (Hz)
×10^4

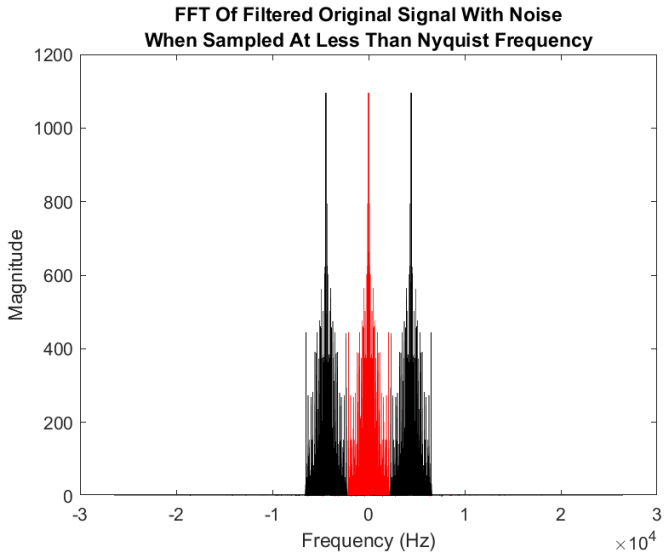

FFT Of Original Signal With Noise
When Sampled At Lesser Than Nyquist Frequency
Magnitude
Frequency (Hz)
×10^4

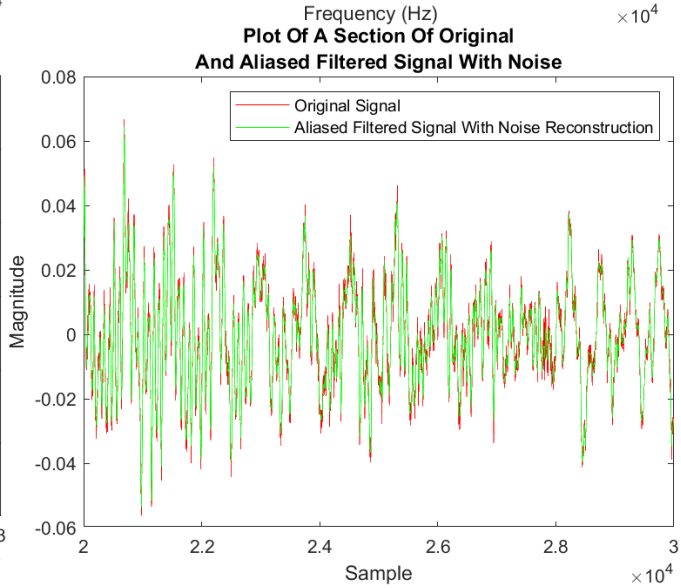

Plot Of A Section Of Original Signal
And Aliased Signal With Noise
Original Signal
Aliased Signal With Noise Reconstruction
Magnitude
Sample
×10^4

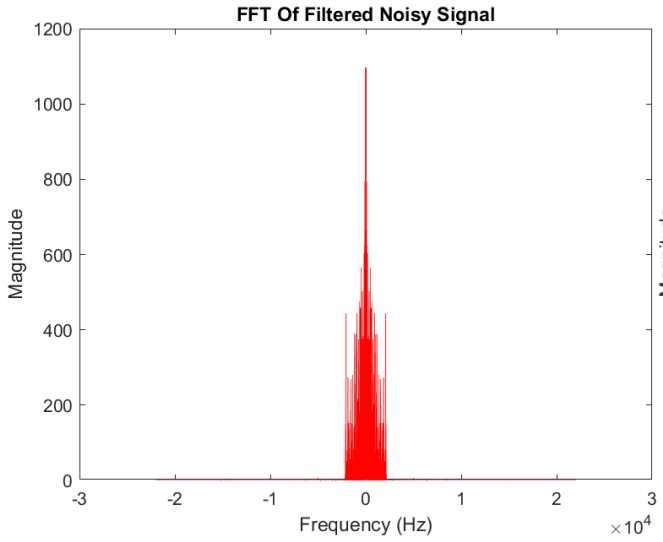

FFT Of Original Signal With Noise
Magnitude
Frequency (Hz)
×10^4

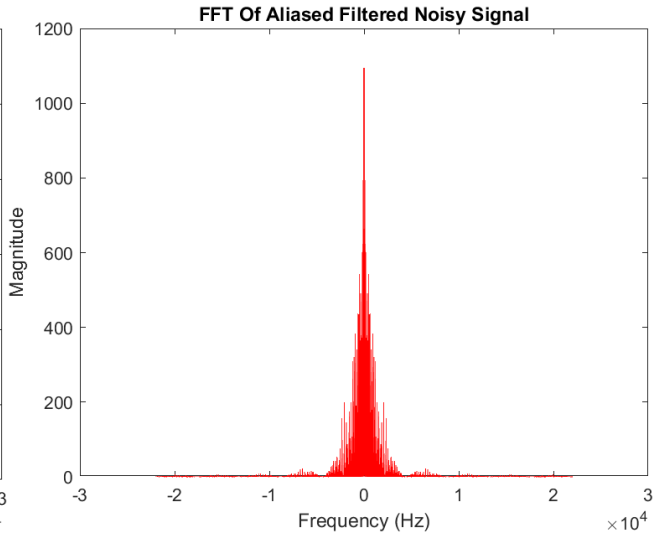

FFT Of Aliased Signal With Noise
Magnitude
Frequency (Hz)
×10^4


FFT Of Filtered Original Signal With Noise
When Sampled At More Than Nyquist Frequency
Magnitude
Frequency (Hz)
×10^4


FFT Of Filtered Original Signal With Noise
When Sampled At Less Than Nyquist Frequency
Magnitude
Frequency (Hz)
×10^4

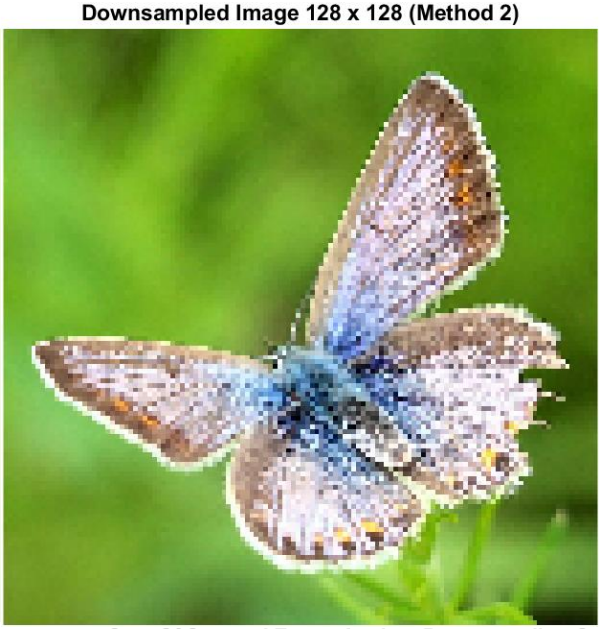

Plot Of A Section Of Original
And Aliased Filtered Signal With Noise
Original Signal
Aliased Filtered Signal With Noise Reconstruction
Magnitude
Sample
×10^4





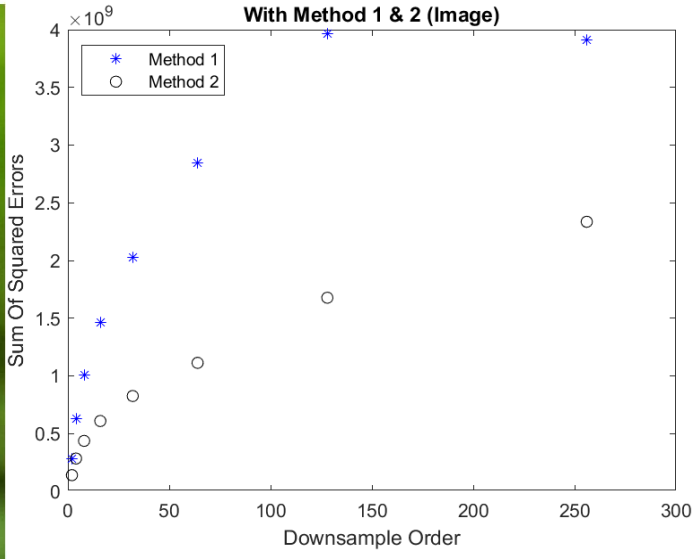
Downsampled Image 128 x 128 (Method 1)

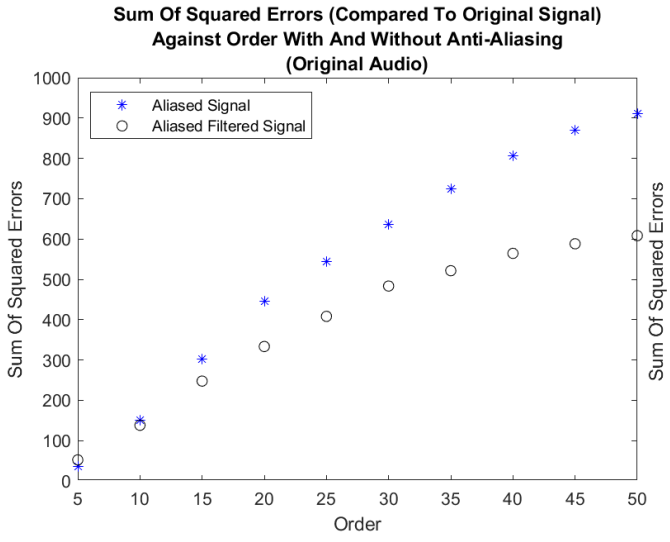
Downsampled Image 128 x 128 (Method 2)

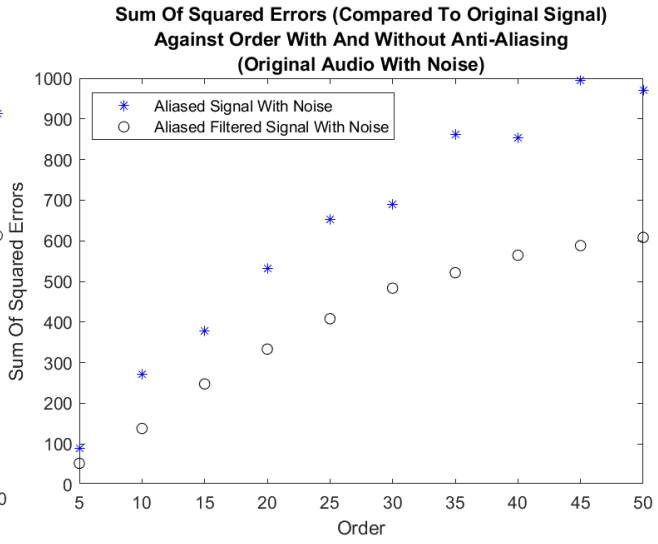


Original Image 512 x 512

Sum Of Squared Errors Against Downsampling Order With Method 1 & 2 (Image)




Sum Of Squared Errors (Compared To Original Signal)
Against Order With And Without Anti-Aliasing
(Original Audio)
Aliased Signal
Aliased Filtered Signal
Sum Of Squared Errors
Order
0
100
200
300
400
500
600
700
800
900
1000
5
10
15
20
25
30
35
40
45
50


Sum Of Squared Errors (Compared To Original Signal)
Against Order With And Without Anti-Aliasing
(Original Audio With Noise)
Aliased Signal With Noise
Aliased Filtered Signal With Noise
Sum Of Squared Errors
Order
0
100
200
300
400
500
600
700
800
900
1000
5
10
15
20
25
30
35
40
45
50

Codes

1. Downsampling Signals Without Noise Function

```
function [sumOfSquaredErrors, sumOfSquaredErrorsFiltered] =
ECE3141_Project_Function_1(original,fs,order)



% Setting The Sample Frequency That Will Cause Aliasing

aliasedFreq = fs/order;



% Downsample Original Signal

aliasedSignal1 = original(1:order:end, 1);

aliasedSignal2 = original(1:order:end, 2);



% Reconstruct Aliased Signal

    aliasedRecon1 = interp1(1:order:length(original), aliasedSignal1,
1:length(original), 'linear', 0);

aliasedRecon2 = interp1(1:order:length(original), aliasedSignal2,
1:length(original), 'linear', 0);

aliasedRecon = [aliasedRecon1', aliasedRecon2'];



% Play The Reconstructed Signal

% soundsc(aliasedRecon, fs);

% pause(5);



% FFT of Original Audio

NOriginal = length(original);

XOriginal = fft(original);

omegaOriginal = (-floor(NOriginal/2):(NOriginal - 1 -
floor(NOriginal/2)))*(fs/NOriginal);



% FFT of Aliased
```

```
NAliased = length(aliasedRecon);

XAliased = fft(aliasedRecon);

omegaAliased = (-floor(NAliased/2):(NAliased - 1 -
floor(NAliased/2)))*(fs/NAliased);



% Plot Of FFT When Sampled At Higher Than Nyquist Frequency

figure

plot(omegaOriginal, fftshift(abs(XOriginal)), 'r')

hold on

plot(omegaOriginal - fs, fftshift(abs(XOriginal)), 'k')

plot(omegaOriginal + fs, fftshift(abs(XOriginal)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Original Signal When Sampled\nAt More Than Nyquist
Frequency'))



% Plot Of FFT When Sampled At Lower Than Nyquist Frequency

figure

plot(omegaOriginal, fftshift(abs(XOriginal)), 'r')

hold on

plot(omegaOriginal - aliasedFreq, fftshift(abs(XOriginal)), 'k')

plot(omegaOriginal + aliasedFreq, fftshift(abs(XOriginal)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Original Signal When Sampled\nAt Lesser Than Nyquist
Frequency'))



% Plot Of Original And Aliased Signal On The Same Plot

figure
```

```
plot(20001:30000, original(20001:30000), 'r')
hold on
plot(20001:30000, aliasedRecon(20001:30000), 'g')
xlabel('Sample')
ylabel('Magnitude')
legend('Original Signal', 'Aliased Reconstruction')
title('Plot Of A Section Of Original And Aliased Signals')

% Plot Of FFT Of Original Signal
figure
plot(omegaOriginal, fftshift(abs(XOriginal)), 'r')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
title('FFT Of Original Signal')

% Plot Of FFT Of Aliased Signal
figure
plot(omegaAliased, fftshift(abs(XAliased)), 'r')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
title('FFT Of Aliased Signal')

% Calculate Sum Of Squared Errors Between Original And Aliased Signals
[rowLength, colLength] = size(original);
sumOfSquaredErrors = 0;
for i = 1:colLength
      sumOfColumnErrors = 0;
      for j = 1:rowLength
```

```
            squaredError = (original(j,i) - aliasedRecon(j,i)).^2;

            sumOfColumnErrors = sumOfColumnErrors + squaredError;

        end

        sumOfSquaredErrors = sumOfSquaredErrors + sumOfColumnErrors;

end


% Applying the anti-aliasing filter here

% Anti-Aliasing Filter

[n,fo,mo,~] = firpmord([floor((1/order)*(fs/2))-100 floor((1/order)*(fs/2))],[1
0],[0.001 0.001],fs);

h = firpm(n,fo,mo);

filtered1 = conv(original(:,1), h);

filtered2 = conv(original(:,2), h);

filtered1 = filtered1(ceil((n-1)/2):end-ceil((n-1)/2));

filtered2 = filtered2(ceil((n-1)/2):end-ceil((n-1)/2));

filteredSignal = [filtered1, filtered2];


% FFT Of Newly Filtered Signal

NFiltered = length(filteredSignal);

XFiltered = fft(filteredSignal);

omegaFiltered = (-floor(NFiltered/2):(NFiltered - 1 -
floor(NFiltered/2)))*fs/NFiltered;


% Plot FFT Of Filtered When Sampled At Higher Than Nyquist Frequency

figure

plot(omegaFiltered, fftshift(abs(XFiltered)), 'r')

hold on

plot(omegaFiltered - fs, fftshift(abs(XFiltered)), 'k')

plot(omegaFiltered + fs, fftshift(abs(XFiltered)), 'k')
```

```
xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Filtered Signal When Sampled At More Than Nyquist Frequency')


% Plot FFT Of Filtered When Sampled At Lower Than Nyquist Frequency

figure

plot(omegaFiltered, fftshift(abs(XFiltered)), 'r')

hold on

plot(omegaFiltered - aliasedFreq, fftshift(abs(XFiltered)), 'k')

plot(omegaFiltered + aliasedFreq, fftshift(abs(XFiltered)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Filtered Signal When Sampled At Lesser Than Nyquist Frequency')


% Downsample Filtered (Anti-Aliased) Signal

aliasedFilteredSignal1 = filteredSignal(1:order:end, 1);

aliasedFilteredSignal2 = filteredSignal(1:order:end, 2);


    % Reconstruct Aliased Filtered Signal

aliasedFilteredRecon1 = interp1(1:order:length(filteredSignal),
aliasedFilteredSignal1, 1:length(filteredSignal), 'linear', 0);

aliasedFilteredRecon2 = interp1(1:order:length(filteredSignal),
aliasedFilteredSignal2, 1:length(filteredSignal), 'linear', 0);

aliasedFilteredRecon = [aliasedFilteredRecon1', aliasedFilteredRecon2'];


% FFT of Aliased Filtered Signal

NFilteredAliased = length(aliasedFilteredRecon);

XFilteredAliased = fft(aliasedFilteredRecon);
```

```
omegaFilteredAliased = (-floor(NFilteredAliased/2):(NFilteredAliased - 1 -
floor(NFilteredAliased/2)))*(fs/NFilteredAliased);


% Plot Of Filtered And Aliased Filtered Signal On The Same Plot

figure

plot(20001:30000, original(20001:30000), 'r')

hold on

plot(20001:30000, aliasedFilteredRecon(20001:30000), 'g')

xlabel('Sample')

ylabel('Magnitude')

legend('Original Signal', 'Aliased Filtered Reconstruction')

title('Plot Of A Section Of Original And Aliased Filtered Signals')


% Plot Of FFT Of Filtered Signal

figure

plot(omegaFiltered, fftshift(abs(XFiltered)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Filtered Signal')


% Plot Of FFT Of Aliased Filtered Signal

figure

    plot(omegaFilteredAliased, fftshift(abs(XFilteredAliased)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Aliased Filtered Signal')


% Calculate Sum Of Squared Errors Between Original And Aliased Filtered Signal

    sumOfSquaredErrorsFiltered = 0;
```

```
for i = 1:colLength

        sumOfColumnErrorsFiltered = 0;

      for j = 1:rowLength

            squaredErrorFiltered = (original(j,i) - aliasedFilteredRecon(j,i)).^2;

            sumOfColumnErrorsFiltered = sumOfColumnErrorsFiltered +
squaredErrorFiltered;

      end

        sumOfSquaredErrorsFiltered = sumOfSquaredErrorsFiltered +
sumOfColumnErrorsFiltered;

end



end
```

2. Downsampling Signals With Noise Function

```
function [sumOfSquaredErrorsNoisy, sumOfSquaredErrorsFilteredNoisy] =
ECE3141_Project_Function_2(original,fs,order)


% Setting The Sample Frequency That Will Cause Aliasing

aliasedFreq = fs/order;


% Getting Size Of Original Signal

[rowLength, colLength] = size(original);


% Generating Noise

P1 = (original(:,1)'*original(:,1))/length(original(:,1));

P2 = (original(:,2)'*original(:,2))/length(original(:,2));

A1 = sqrt(2*P1/10);

A2 = sqrt(2*P2/10);

n1 = 1:length(original(:,1));

n2 = 1:length(original(:,2));

noise1 = A1*cos(2*pi*5000*n1./fs);

noise2 = A2*cos(2*pi*5000*n2./fs);

newNoise1 = noise1';

newNoise2 = noise2';


% Inserting Noise Into Original Signal
```

```
for i = 1:length(original)

      originalNoisy1(i) = newNoise1(i) + original(i,1);

      originalNoisy2(i) = newNoise2(i) + original(i,2);

end


% Creating Original Signal With Noise

originalNoisy = [originalNoisy1', originalNoisy2'];


% Downsample Original Signal With Noise

aliasedSignalNoisy1 = originalNoisy(1:order:end, 1);

aliasedSignalNoisy2 = originalNoisy(1:order:end, 2);


% Reconstruct Aliased Signal With Noise

aliasedReconNoisy1 = interp1(1:order:length(originalNoisy), aliasedSignalNoisy1,
1:length(originalNoisy), 'linear', 0);

aliasedReconNoisy2 = interp1(1:order:length(originalNoisy), aliasedSignalNoisy2,
1:length(originalNoisy), 'linear', 0);

aliasedReconNoisy = [aliasedReconNoisy1', aliasedReconNoisy2'];

% soundsc(aliasedReconNoisy, fs);

% pause(5);


% FFT of Original Audio With Noise

NOriginalNoisy = length(originalNoisy);

XOriginalNoisy = fft(originalNoisy);

omegaOriginalNoisy = (-floor(NOriginalNoisy/2):(NOriginalNoisy - 1 -
floor(NOriginalNoisy/2)))*(fs/NOriginalNoisy);


% FFT of Aliased Audio With Noise

NAliasedNoisy = length(aliasedReconNoisy);
```

```
XAliasedNoisy = fft(aliasedReconNoisy);

omegaAliasedNoisy = (-floor(NAliasedNoisy/2):(NAliasedNoisy - 1 -
floor(NAliasedNoisy/2)))*(fs/NAliasedNoisy);



% Plot Of FFT Of Original Signal With Noise When Sampled At Higher Than Nyquist
Frequency

figure

plot(omegaOriginalNoisy, fftshift(abs(XOriginalNoisy)), 'r')

hold on

plot(omegaOriginalNoisy - fs, fftshift(abs(XOriginalNoisy)), 'k')

plot(omegaOriginalNoisy + fs, fftshift(abs(XOriginalNoisy)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Original Signal With Noise\nWhen Sampled At More Than Nyquist
Frequency'))



% Plot Of FFT Of Original Signal With Noise When Sampled At Lower Than Nyquist
Frequency

figure

plot(omegaOriginalNoisy, fftshift(abs(XOriginalNoisy)), 'r')

hold on

plot(omegaOriginalNoisy - aliasedFreq, fftshift(abs(XOriginalNoisy)), 'k')

plot(omegaOriginalNoisy + aliasedFreq, fftshift(abs(XOriginalNoisy)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Original Signal With Noise\nWhen Sampled At Lesser Than
Nyquist Frequency'))



% Plot Of Original Signal And Aliased Signal With Noise On The Same Plot

figure
```

```
plot(20001:30000, original(20001:30000), 'r')

hold on

plot(20001:30000, aliasedReconNoisy(20001:30000), 'g')

xlabel('Sample')

ylabel('Magnitude')

legend('Original Signal', 'Aliased Signal With Noise Reconstruction')

title(sprintf('Plot Of A Section Of Original Signal\nAnd Aliased Signal With
Noise'))


% Plot Of FFT Of Original Signal With Noise

figure

plot(omegaOriginalNoisy, fftshift(abs(XOriginalNoisy)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Original Signal With Noise')


% Plot Of FFT Of Aliased Signal With Noise

figure

plot(omegaAliasedNoisy, fftshift(abs(XAliasedNoisy)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Aliased Signal With Noise')


% Calculate Sum Of Squared Errors Between Original Signal And Aliased

% Signal With Noise

sumOfSquaredErrorsNoisy = 0;

for i = 1:colLength

        sumOfColumnErrorsNoisy = 0;

      for j = 1:rowLength
```

```
            squaredErrorNoisy = (original(j,i) - aliasedReconNoisy(j,i)).^2;

            sumOfColumnErrorsNoisy = sumOfColumnErrorsNoisy + squaredErrorNoisy;

       end

         sumOfSquaredErrorsNoisy = sumOfSquaredErrorsNoisy + sumOfColumnErrorsNoisy;

    end


% Apply Anti-Aliasing Filter To Original Signal With Noise

[n,fo,mo,~] = firpmord([floor((1/order)*(fs/2))-100 floor((1/order)*(fs/2))],[1
0],[0.001 0.001],fs);

h = firpm(n,fo,mo);

filteredNoisy1 = conv(originalNoisy(:,1), h);

    filteredNoisy2 = conv(originalNoisy(:,2), h);

filteredNoisy1 = filteredNoisy1(ceil((n-1)/2):end-ceil((n-1)/2));

filteredNoisy2 = filteredNoisy2(ceil((n-1)/2):end-ceil((n-1)/2));

filteredSignalNoisy = [filteredNoisy1, filteredNoisy2];


% FFT Of Newly Filtered Signal (With Noise)

NFilteredNoisy = length(filteredSignalNoisy);

XFilteredNoisy = fft(filteredSignalNoisy);

omegaFilteredNoisy = (-floor(NFilteredNoisy/2):(NFilteredNoisy - 1 -
floor(NFilteredNoisy/2)))*fs/NFilteredNoisy;


% Plot Of FFT Of Filtered Signal With Noise When Sampled At Higher Than Nyquist
Frequency

figure

plot(omegaFilteredNoisy, fftshift(abs(XFilteredNoisy)), 'r')

hold on

plot(omegaFilteredNoisy - fs, fftshift(abs(XFilteredNoisy)), 'k')

plot(omegaFilteredNoisy + fs, fftshift(abs(XFilteredNoisy)), 'k')
```

```
xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Filtered Original Signal With Noise\nWhen Sampled At More Than Nyquist Frequency'))


% Plot Of FFT Of Filtered Signal With Noise When Sampled At Lower Than Nyquist Frequency

figure

plot(omegaFilteredNoisy, fftshift(abs(XFilteredNoisy)), 'r')

hold on

plot(omegaFilteredNoisy - aliasedFreq, fftshift(abs(XFilteredNoisy)), 'k')

plot(omegaFilteredNoisy + aliasedFreq, fftshift(abs(XFilteredNoisy)), 'k')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title(sprintf('FFT Of Filtered Original Signal With Noise\nWhen Sampled At Less Than Nyquist Frequency'))


% Downsample Filtered Signal With Noise

    aliasedFilteredSignalNoisy1 = filteredSignalNoisy(1:order:end, 1);

    aliasedFilteredSignalNoisy2 = filteredSignalNoisy(1:order:end, 2);


% Reconstruct Aliased Filtered Signal With Noise

    aliasedFilteredReconNoisy1 = interp1(1:order:length(filteredSignalNoisy), aliasedFilteredSignalNoisy1, 1:length(filteredSignalNoisy), 'linear', 0);

    aliasedFilteredReconNoisy2 = interp1(1:order:length(filteredSignalNoisy), aliasedFilteredSignalNoisy2, 1:length(filteredSignalNoisy), 'linear', 0);

    aliasedFilteredReconNoisy = [aliasedFilteredReconNoisy1', aliasedFilteredReconNoisy2'];


% FFT of Aliased Filtered Signal With Noise

NFilteredAliasedNoisy = length(aliasedFilteredReconNoisy);
```

```
XFilteredAliasedNoisy = fft(aliasedFilteredReconNoisy);

    omegaFilteredAliasedNoisy = (-
floor(NFilteredAliasedNoisy/2):(NFilteredAliasedNoisy - 1 -
floor(NFilteredAliasedNoisy/2)))*(fs/NFilteredAliasedNoisy);



% Plot Of Original And Aliased Filtered Signal With Noise On The Same Plot

figure

plot(20001:30000, original(20001:30000), 'r')

hold on

plot(20001:30000, aliasedFilteredReconNoisy(20001:30000), 'g')

xlabel('Sample')

ylabel('Magnitude')

legend('Original Signal', 'Aliased Filtered Signal With Noise Reconstruction')

title(sprintf('Plot Of A Section Of Original\nAnd Aliased Filtered Signal With
Noise'))



% Plot Of FFT Of Filtered Signal With Noise And Aliased Filtered Signal With Noise

figure

plot(omegaFilteredNoisy, fftshift(abs(XFilteredNoisy)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Filtered Noisy Signal')



% Plot Of FFT Of Aliased Filtered Signal With Noise

figure

    plot(omegaFilteredAliasedNoisy, fftshift(abs(XFilteredAliasedNoisy)), 'r')

xlabel('Frequency (Hz)')

ylabel('Magnitude')

title('FFT Of Aliased Filtered Noisy Signal')
```

```
% Calculate Sum Of Squared Errors Between Original And Aliased Filtered

% Signal With Noise

    sumOfSquaredErrorsFilteredNoisy = 0;

for i = 1:colLength

        sumOfColumnErrorsFilteredNoisy = 0;

      for j = 1:rowLength

            squaredErrorFilteredNoisy = (original(j,i) -
aliasedFilteredReconNoisy(j,i)).^2;

            sumOfColumnErrorsFilteredNoisy = sumOfColumnErrorsFilteredNoisy +
squaredErrorFilteredNoisy;

      end

        sumOfSquaredErrorsFilteredNoisy = sumOfSquaredErrorsFilteredNoisy +
sumOfColumnErrorsFilteredNoisy;

end



end
```

3. Image Downsampling (Method 1)

```
function [sumOfSquaredTotal] = ECE3141_Project_Function_3(image, downsample)


% Converting the image to a double

image_double = double(image);


% Obtaining the size of the image

[r1,c1,z1] = size(image_double);


% Method 1: Taking every nth and mth pixel

% (Where n and m are the downsampling orders for the row and columns respectively)

Downsampled_Image_1 = image_double(1:downsample(1):r1, 1:downsample(2):c1, :);


% Showing The Downsampled Image

figure;

    imshow(uint8(Downsampled_Image_1));

[r11, c11, z11] = size(Downsampled_Image_1);
```

```
title(sprintf('Downsampled Image %.0d x %.0d (Method 2)', r11, c11));


% Calculating Sum Of Squared Errors Between The Original And

% Downsampled Image

n1 = r1/r11;

n2 = c1/c11;

sumOfSquaredTotal = 0;

for i = 1:r11

      for j = 1:c11

            for k = 1:n1

            for m = 1:n2

                  for z = 1:z11

                        sumOfSquared = (image_double(n1*(i-1) + k, n2*(j-1) + m, z)
- Downsampled_Image_1(i, j, z)).^2;

                        sumOfSquaredTotal = sumOfSquaredTotal + sumOfSquared;

                  end

            end

            end

      end

end

end
```

4. Image Downsampling (Method 2)

```
function [sumOfSquaredTotal] = ECE3141_Project_Function_4(image, downsample)


% Converting the image to a double

image_double = double(image);


% Obtaining the size of the image

[r1,c1,z1] = size(image_double);


% Number of pixels to be averaged into 1

divisor1 = downsample(1)*downsample(2);


% Method 2: Taking average of n x m pixels into a single pixel

% (Where n and m is the order of downsampling for rows and columns

% respectively)

for i = 1:(r1/downsample(1))

        for j = 1:(c1/downsample(2))
```

```
            for z = 1:z1
            sum = 0;
            for k = 1:downsample(1)
                  for m = 1:downsample(2)
                        sum = sum + image_double(downsample(1)*(i-1) + k,
downsample(2)*(j-1) + m, z);
                  end
            end
            average = sum/divisor1;
                Downsampled_Image_1(i, j, z) = average;
            end
      end
end


% Showing The Downsampled Image
figure;
    imshow(uint8(Downsampled_Image_1));
[r11, c11, z11] = size(Downsampled_Image_1);
title(sprintf('Downsampled Image %.0d x %.0d (Method 1)', r11, c11));


% Calculating Sum Of Squared Errors Between The Original And
% Downsampled Image
n1 = r1/r11;
n2 = c1/c11;
sumOfSquaredTotal = 0;
for i = 1:r11
      for j = 1:c11
            for k = 1:n1
            for m = 1:n2
```

```
                for z = 1:z11
                        sumOfSquared = (image_double(n1*(i-1) + k, n2*(j-1) + m, z)
- Downsampled_Image_1(i, j, z)).^2;
                        sumOfSquaredTotal = sumOfSquaredTotal + sumOfSquared;
                end
            end
            end
      end
end
end
```

5. Main Code

```
% ECE3141 Project: Aliasing
% Group Number: 3
% Member 1: Edmund Lee Wai Loon
% Member 2: Tan Jin Chun
% Member 3: Ng Zhan Lok

clc; clear all; close all;

% Extracting the data from the audio file
[original, fs] = audioread('The-Incredibles.wav');
% soundsc(original, fs);
% pause(5);

% Extracting the data from the image
```

```
image = imread("image.jpeg");

image_double = double(image);

[r1,c1,z1] = size(image_double);


% Showing Original Image

figure

imshow(image);

hold on

title(sprintf('Original Image %.0d x %.0d', r1, c1));


% Initialising the order of the filter

order = 5:5:50;

downsample = [2, 4, 8, 16, 32, 64, 128, 256; 2, 4, 8, 16, 32, 64, 128, 256];


% Pre-allocation

sumOfSquaredErrors = zeros(1, length(order));

sumOfSquaredErrorsFiltered = zeros(1, length(order));

sumOfSquaredErrorsNoisy = zeros(1, length(order));

sumOfSquaredErrorsFilteredNoisy = zeros(1, length(order));

sumOfSquaredTotal1 = zeros(1, length(downsample));

sumOfSquaredTotal2 = zeros(1, length(downsample));


% Looping To Get Sum Of Squared Error For Different Orders

for i = 1:length(order)

[sumOfSquaredErrors(i), sumOfSquaredErrorsFiltered(i)] =
ECE3141_Project_Function_1(original,fs,order(i));

    [sumOfSquaredErrorsNoisy(i), sumOfSquaredErrorsFilteredNoisy(i)] =
ECE3141_Project_Function_2(original,fs,order(i));

end
```

```
for i = 1:length(downsample)

[sumOfSquaredTotal1(i)] = ECE3141_Project_Function_3(image, [downsample(1,i),
downsample(2,i)]);

[sumOfSquaredTotal2(i)] = ECE3141_Project_Function_4(image, [downsample(1,i),
downsample(2,i)]);

end



% Plot trend of sum of squared errors

figure

plot(order, sumOfSquaredErrors, 'b*')

hold on

plot (order, sumOfSquaredErrorsFiltered, 'ko')

title(sprintf('Sum Of Squared Errors (Compared To Original Signal)\nAgainst Order
With And Without Anti-Aliasing\n(Original Audio)'))

legend('Aliased Signal', 'Aliased Filtered Signal', 'Location', 'northwest')

xlabel('Order')

ylabel('Sum Of Squared Errors')



figure

plot(order, sumOfSquaredErrorsNoisy, 'b*')

hold on

plot (order, sumOfSquaredErrorsFilteredNoisy, 'ko')

title(sprintf('Sum Of Squared Errors (Compared To Original Signal)\nAgainst Order
With And Without Anti-Aliasing\n(Original Audio With Noise)'))

legend('Aliased Signal With Noise', 'Aliased Filtered Signal With Noise',
'Location', 'northwest')

xlabel('Order')

ylabel('Sum Of Squared Errors')
```

```
figure

plot(downsample(1,:), sumOfSquaredTotal1, 'b*')

hold on

plot (downsample(1,:), sumOfSquaredTotal2, 'ko')

title(sprintf('Sum Of Squared Errors Against Downsampling Order\nWith Method 1 & 2
(Image)'))

legend('Method 1', 'Method 2', 'Location', 'northwest')

xlabel('Downsample Order')

ylabel('Sum Of Squared Errors')
```