Experiment-1

Sampling

A) Sampling of a Sinusoidal Signal

The Kotolnikov-Shannon-Nyquist sampling theorem states that a bandlimited signal with band limit f_m, can be completely reconstructed from its samples, provided it is sampled at a frequency $\mathbf{F_s > 2 * f_m}$

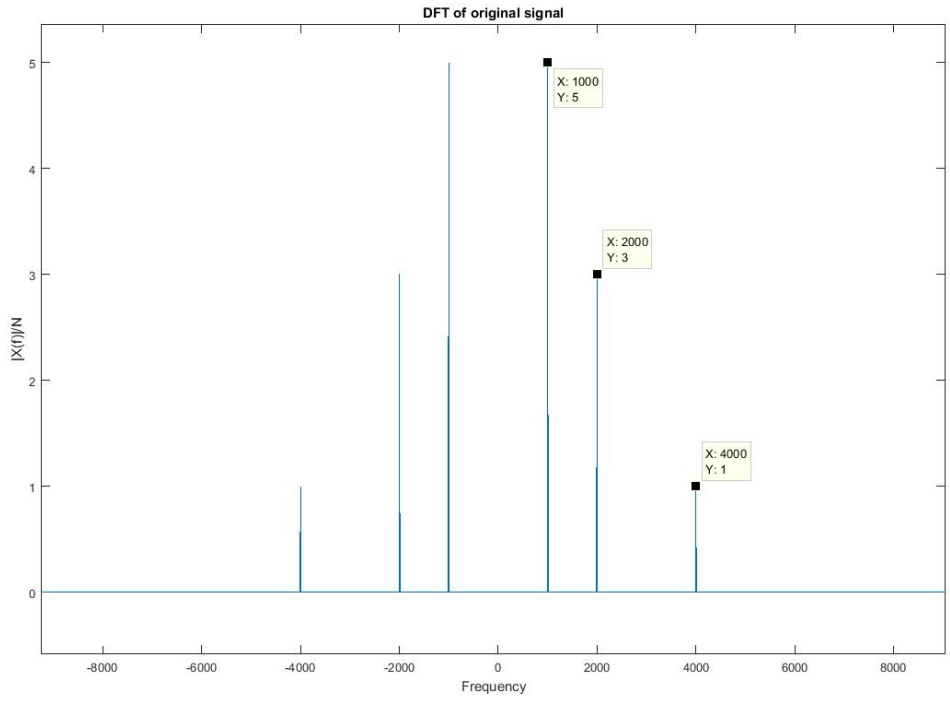
It is worth noting that in MATLAB, everything is discrete time and our original data is in fact, discrete time data sampled at a very high frequency (100kHz)

Also, the fft algorithm maps N points in the time domain to N points in the frequency domain. These are actually the 'samples' of DTFT, since continuous time values cannot be stored in hardware. This again, is at a rate of 100kHz (decreases as N is reduced)

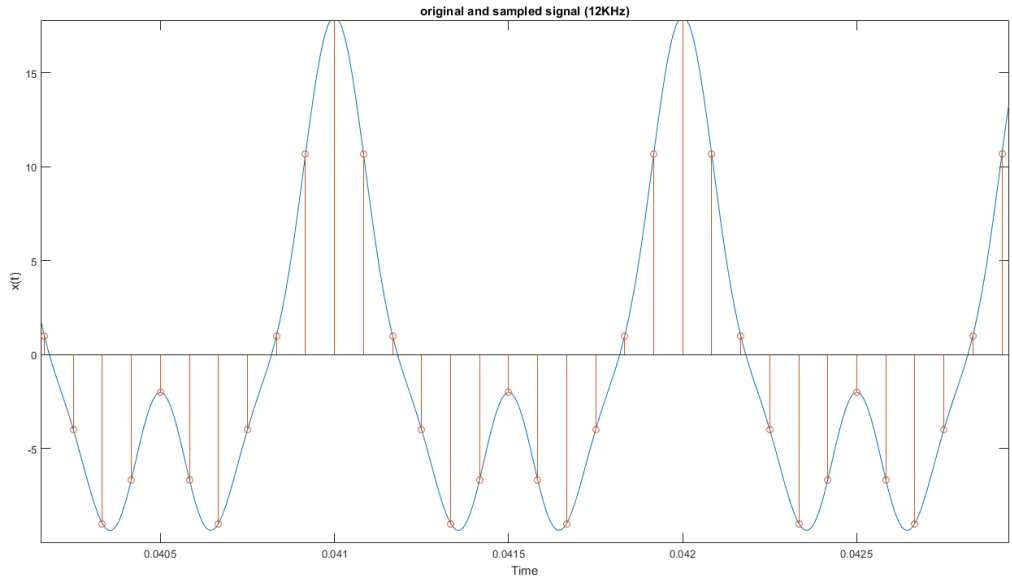
As the window of truncation N is reduced, we have more and more spreading in the spectrum. However, for computation purposes, reducing N reduces the time as well as the storage space required.

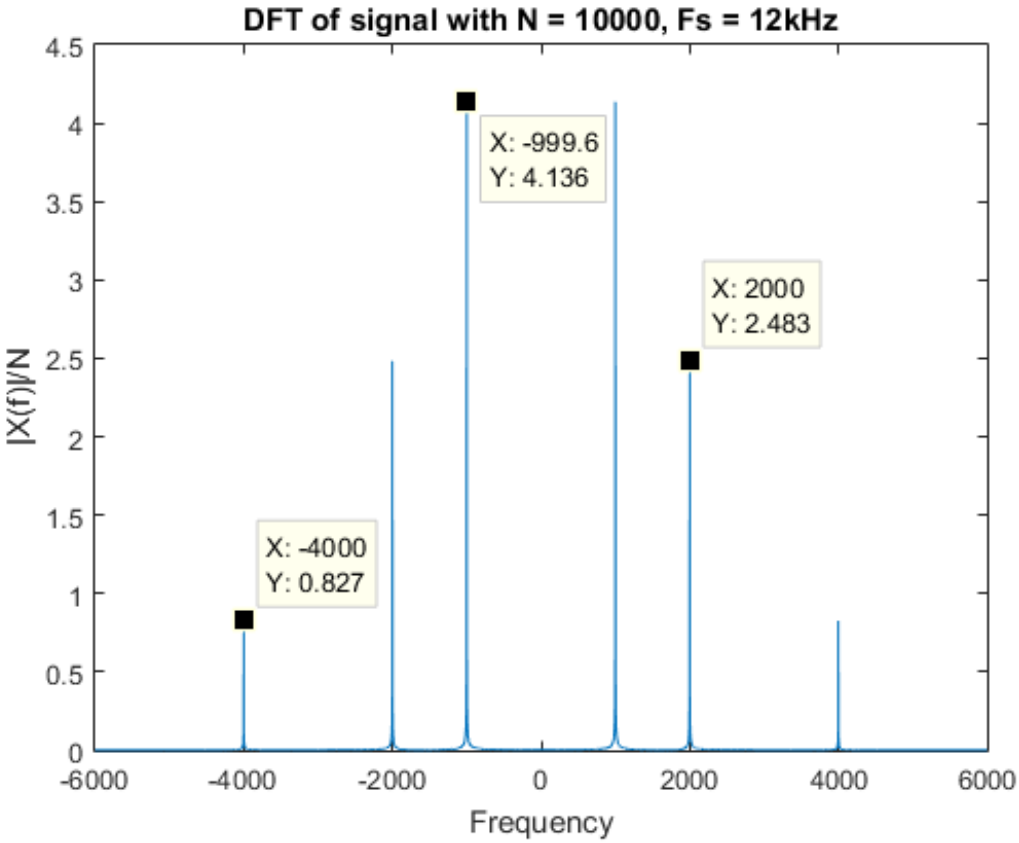
Our signal is:

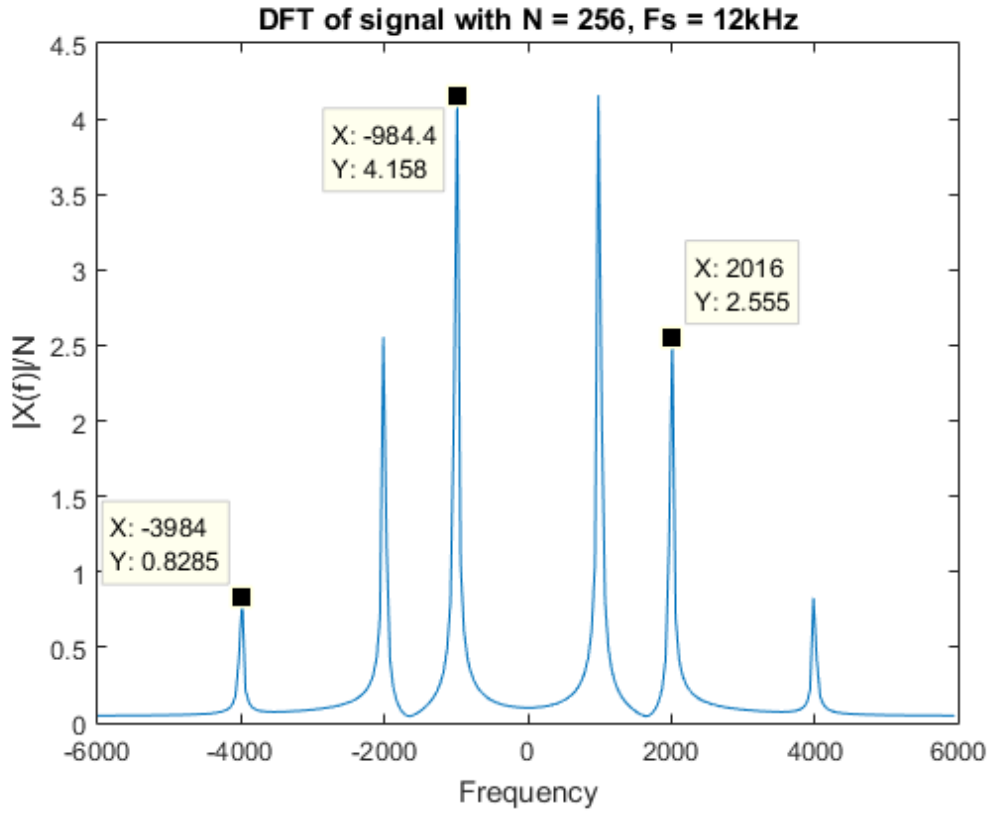
$$x(t) = 10\cos(2\pi \times 10^3 t) + 6\cos(2\pi \times 2 \times 10^3 t) + 2\cos(2\pi \times 4 \times 10^3 t).$$

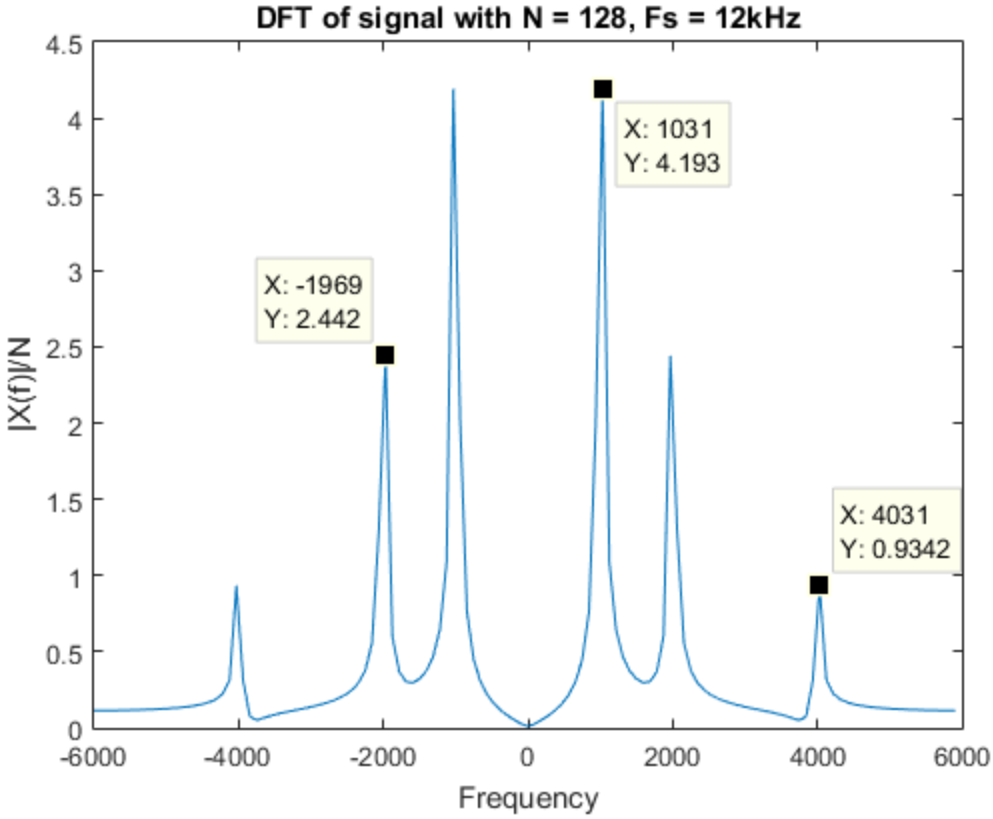


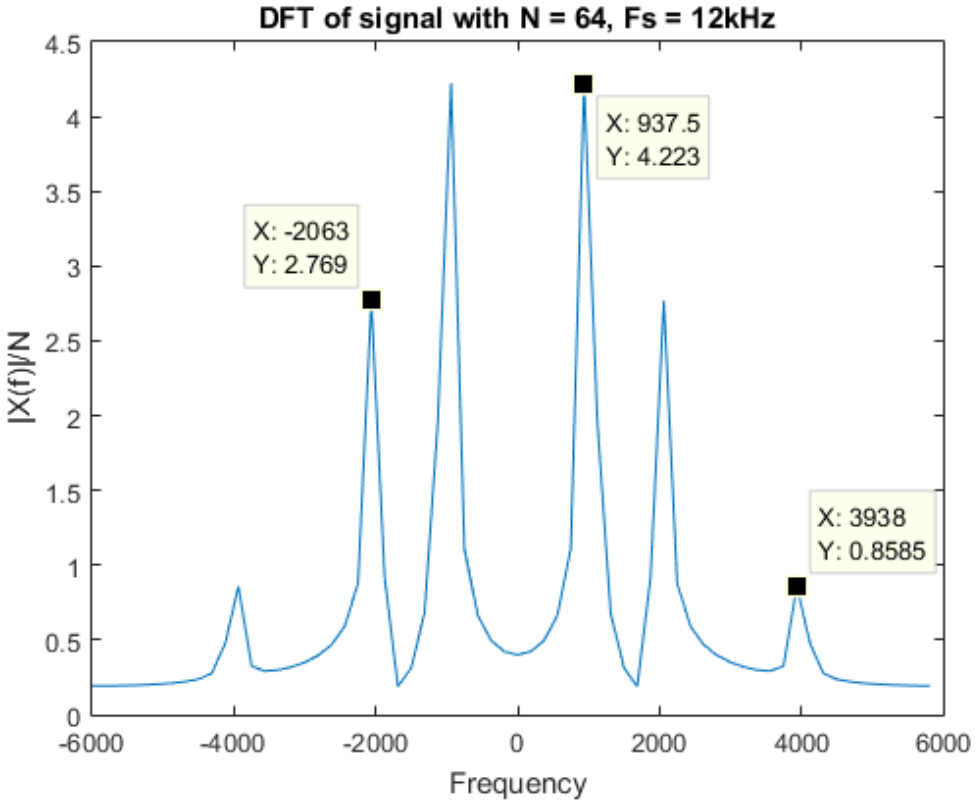
Since the maximum frequency is 4kHz, the Nyquist sampling frequency becomes 8kHz. Let us sample it at 12kHz, so as to avoid any aliasing effects.





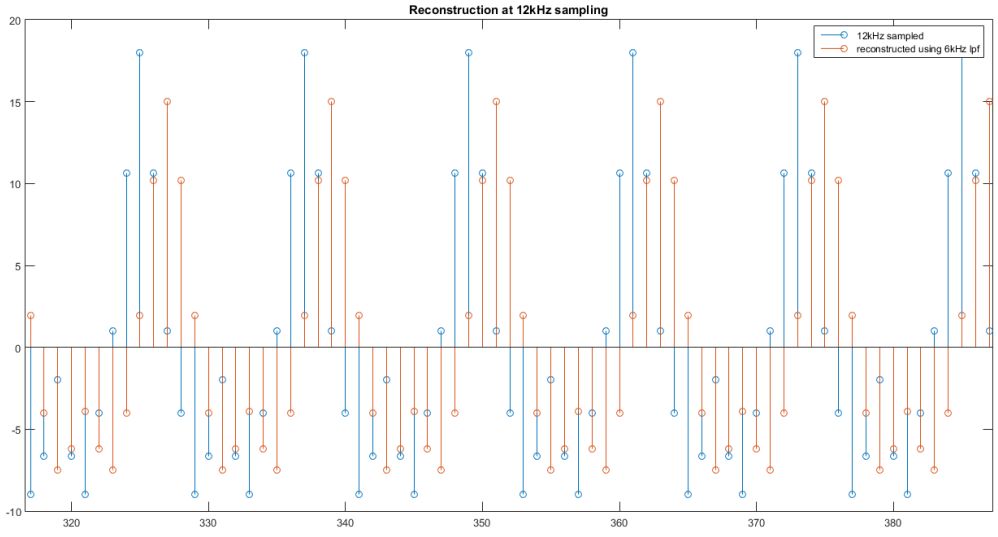

DFT of signal with N = 256, Fs = 12kHz
X: -984.4
Y: 4.158
X: 2016
Y: 2.555
X: -3984
Y: 0.8285
|X(f)|/N
Frequency
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-6000
-4000
-2000
0
2000
4000
6000


DFT of signal with N = 128, Fs = 12kHz
X: 1031
Y: 4.193
X: -1969
Y: 2.442
X: 4031
Y: 0.9342
|X(f)|/N
Frequency
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-6000
-4000
-2000
0
2000
4000
6000



As we can see, there is maximum spreading effect seen when N is reduced to 64. The magnitudes of the original frequencies are also lowered as a result of the spreading (from 5 to 4.22, 3 to 2.77 and so on)

Now, let us demonstrate the reconstruction property of this sampled signal. For this, simply pass it through a low-pass filter of cutoff $F_s/2$ = 6kHz



We can see that aside from a scaling and delay factor (which is acceptable in channel communication), we are getting the original signal back. However, this will not be the case when we go below Nyquist rate.

MATLAB Code:

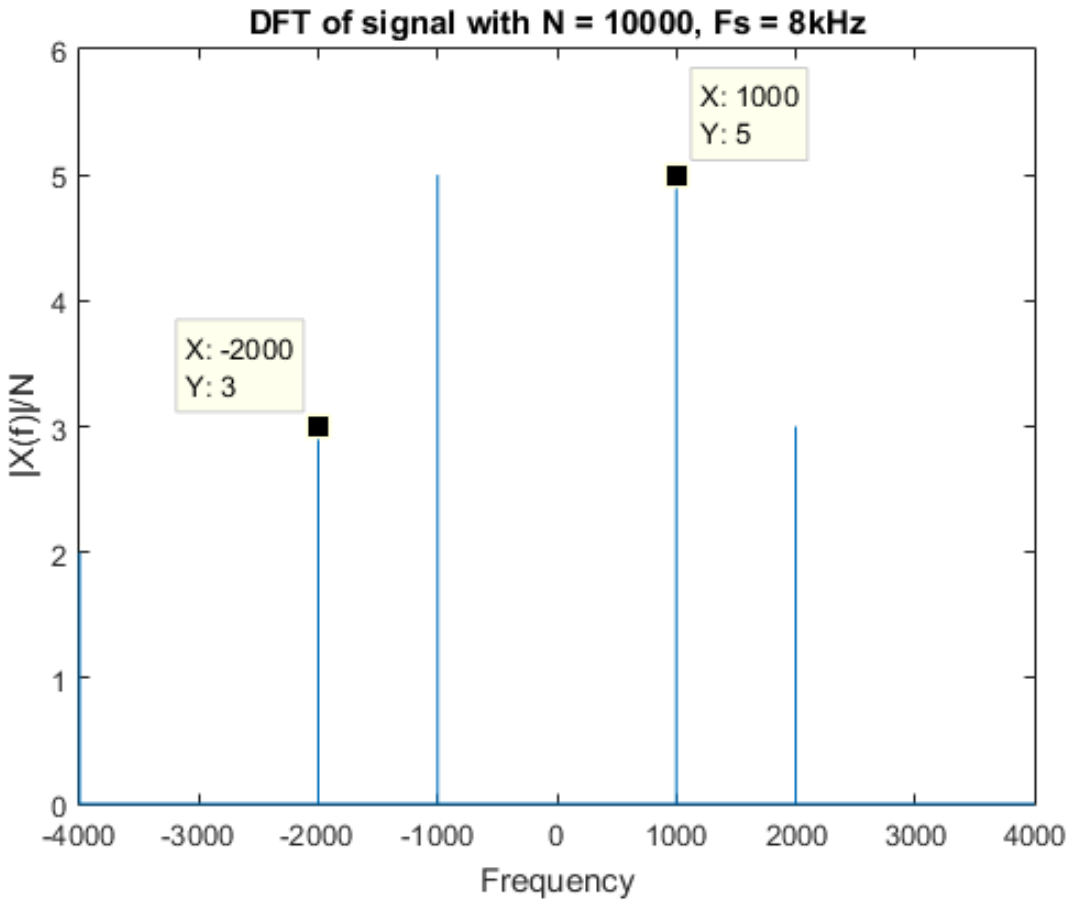
```
%% Sampling and effect of Nyquist Criteria
Fs = 12e3;
%Setting the Sampling Rate
t=0:0.01*10^-3:(100-0.01)*10^-3;
%Creating the time array for original signal
t1 = 0:1/Fs:(100-0.01)*10^-3;
%Creating the time
x=10*cos(2*pi*1e3*t)+6*cos(2*pi*2*1e3*t)+2*cos(2*pi*4*1e3*t);
% Original Function
x1=10*cos(2*pi*1e3*t1)+6*cos(2*pi*2*1e3*t1)+2*cos(2*pi*4*1e3*t1)
;          % Sampled function at 12kHz
plot(t,x);
%Plotting the original function
hold on;
stem(t1,x1);
%Plotting the sampled function
xlabel('Time');
%X-Axis Label
ylabel('x(t)');
%Y-Axis Label
title('original and sampled signal');
%Plot title
xlim([0.039, 0.0415]);
hold off
```

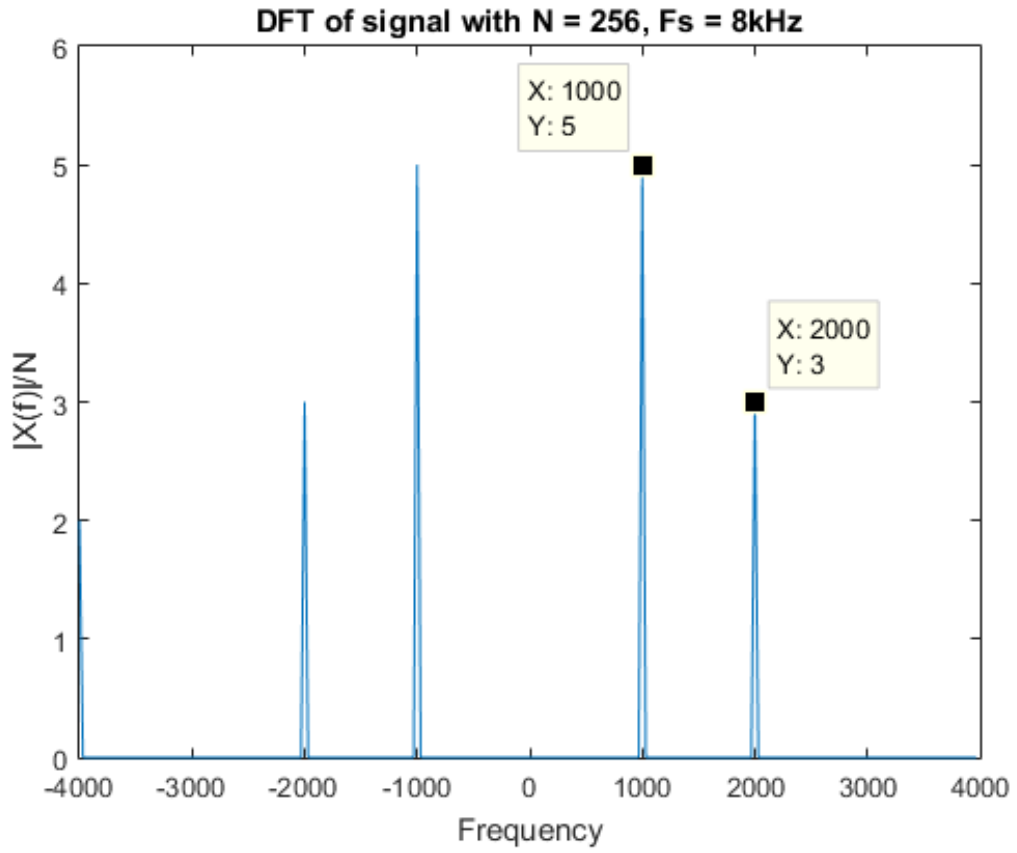
B) Sampling at and below Nyquist Rate

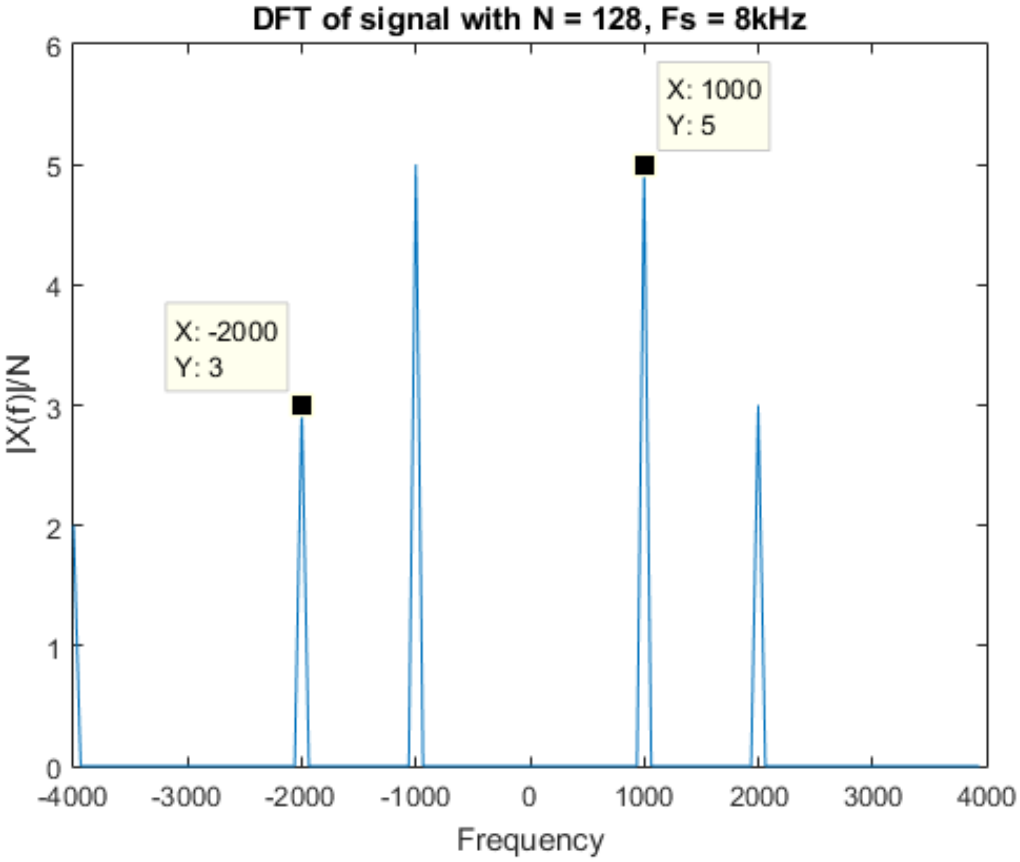
Now, let us try sampling at and below Nyquist sampling rate ($2*f_m$ = **8kHz**)

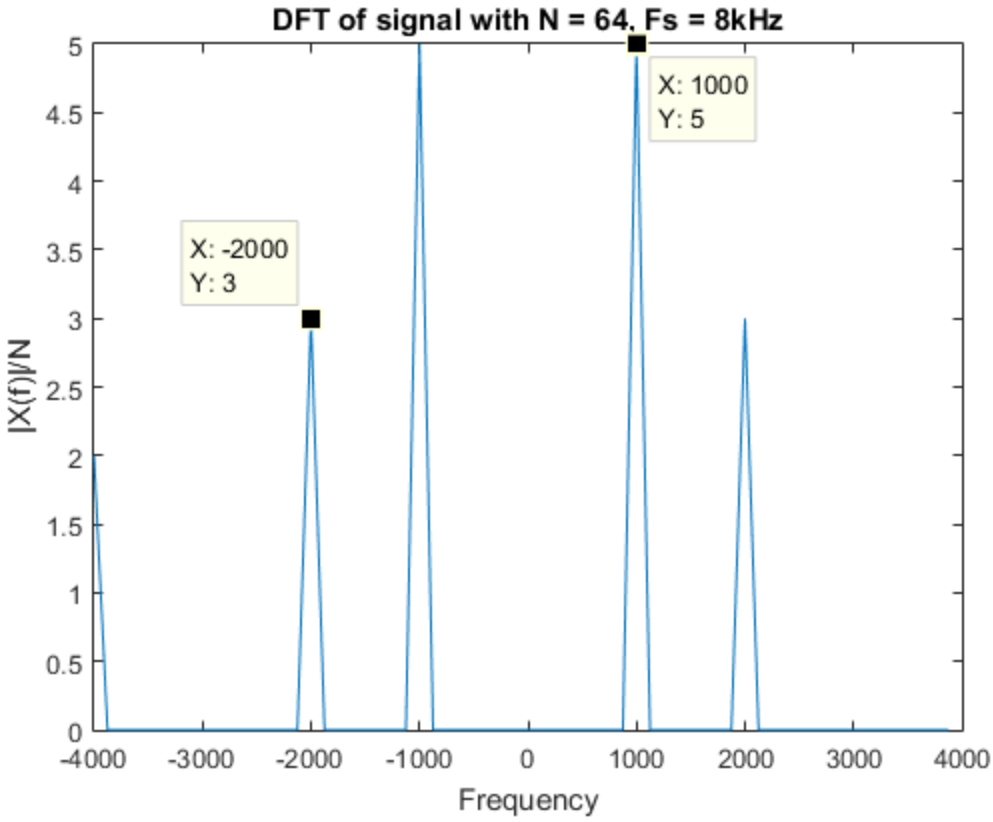
This will cause aliasing between the frequency components and consequentially, corruption of the frequency content of the original signal. Once aliased, there is no way to reconstruct the original signal back from the samples. This sort of data compression has actually led to a loss and corruption in signal frequency.

1. **8kHz**

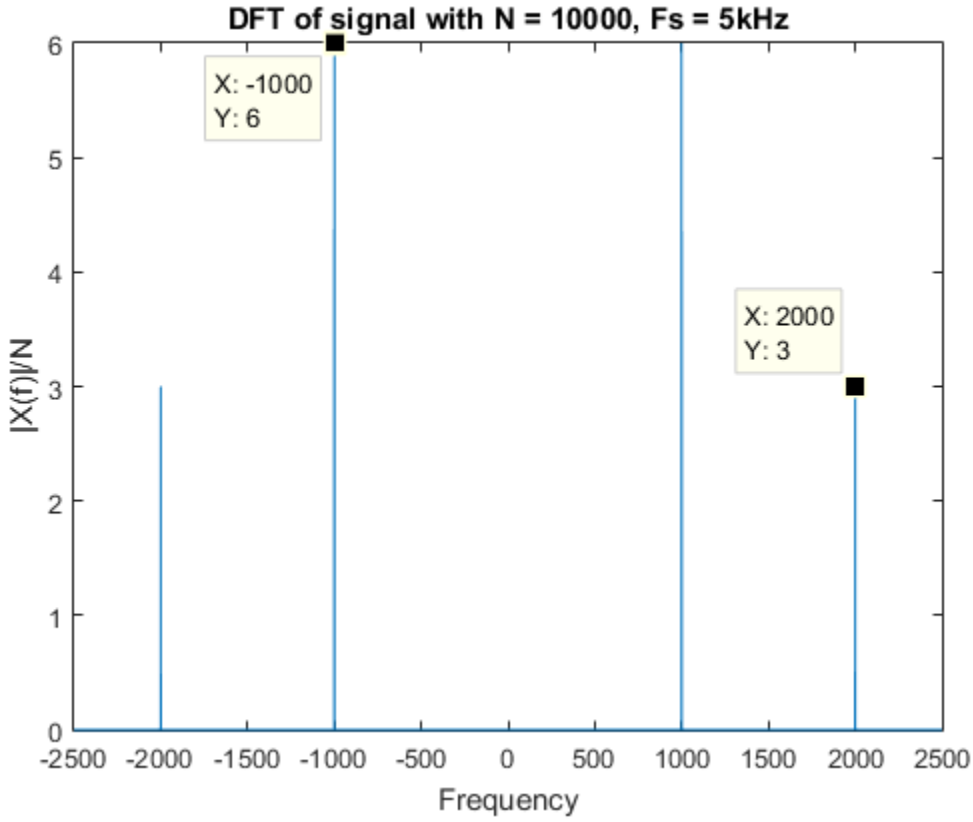



DFT of signal with N = 256, Fs = 8kHz
X: 1000
Y: 5
X: 2000
Y: 3
|X(f)|/N
Frequency
0
1
2
3
4
5
6
-4000
-3000
-2000
-1000
0
1000
2000
3000
4000

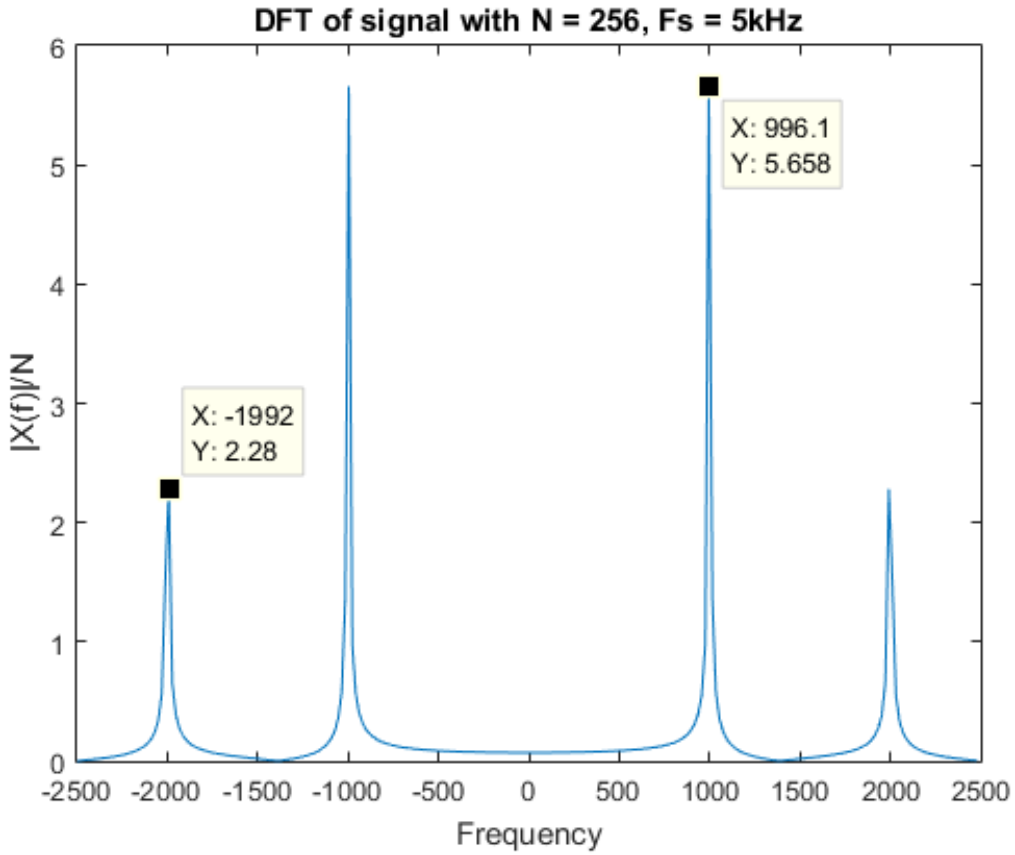

DFT of signal with N = 128, Fs = 8kHz
X: 1000
Y: 5
X: -2000
Y: 3
|X(f)|/N
Frequency
0
1
2
3
4
5
6
-4000
-3000
-2000
-1000
0
1000
2000
3000
4000

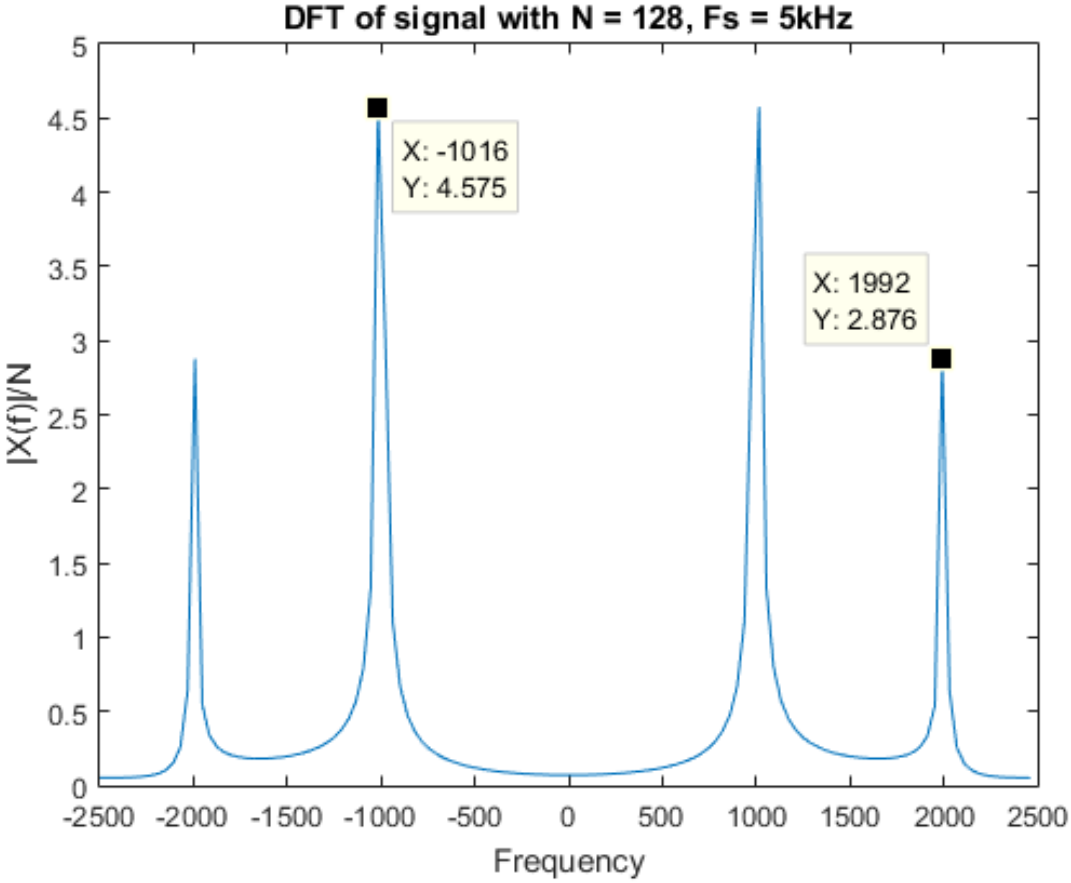


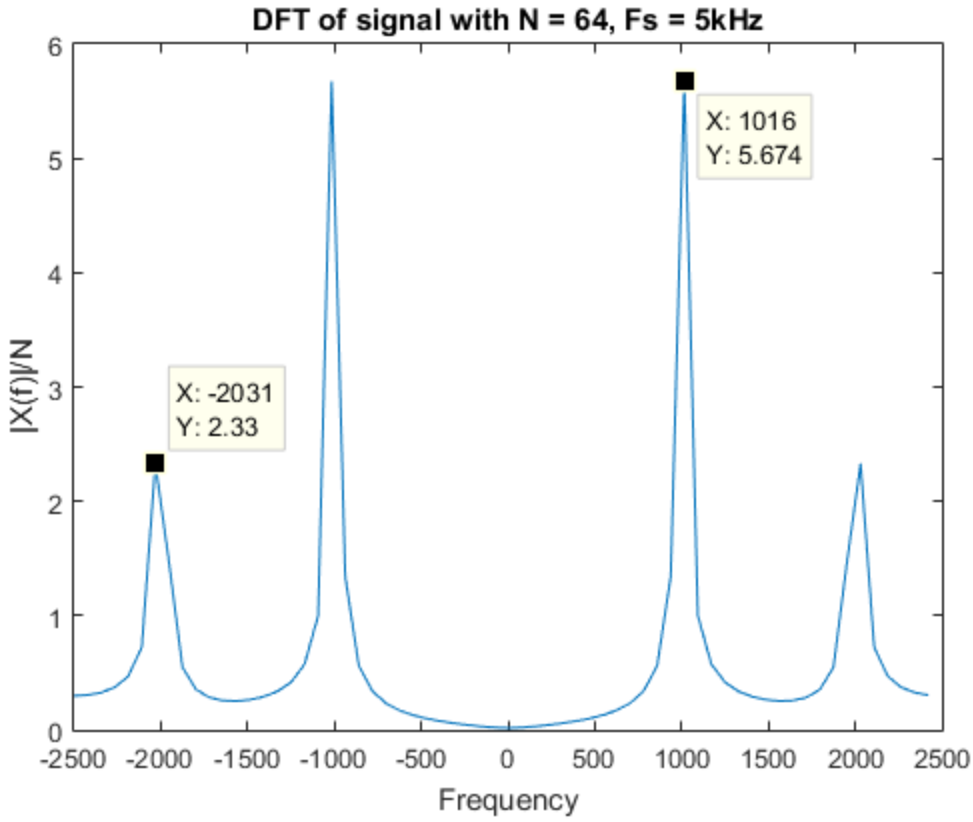
2. **<u>5kHz</u>**



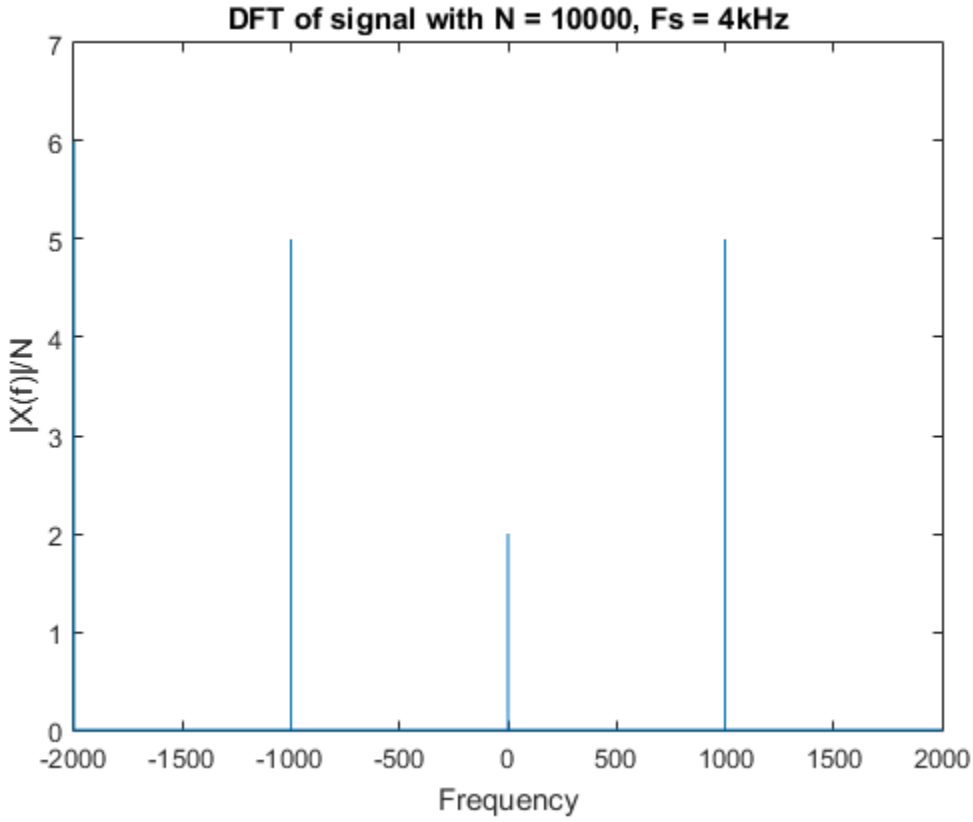
Folding of frequency takes place, and the frequency at Fs/2+1.5kHz gets added to Fs/2-1.5kHz. So, the 4kHz component gets aliased with the frequency component at 1kHz. This can be easily seen as 5+1=6 is the magnitude at 1kHz.


DFT of signal with N = 256, Fs = 5kHz
X: 996.1
Y: 5.658
X: -1992
Y: 2.28
|X(f)|/N
Frequency
0
1
2
3
4
5
6
-2500
-2000
-1500
-1000
-500
0
500
1000
1500
2000
2500


DFT of signal with N = 128, Fs = 5kHz
X: -1016
Y: 4.575
X: 1992
Y: 2.876
|X(f)|/N
Frequency
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
-2500
-2000
-1500
-1000
-500
0
500
1000
1500
2000
2500

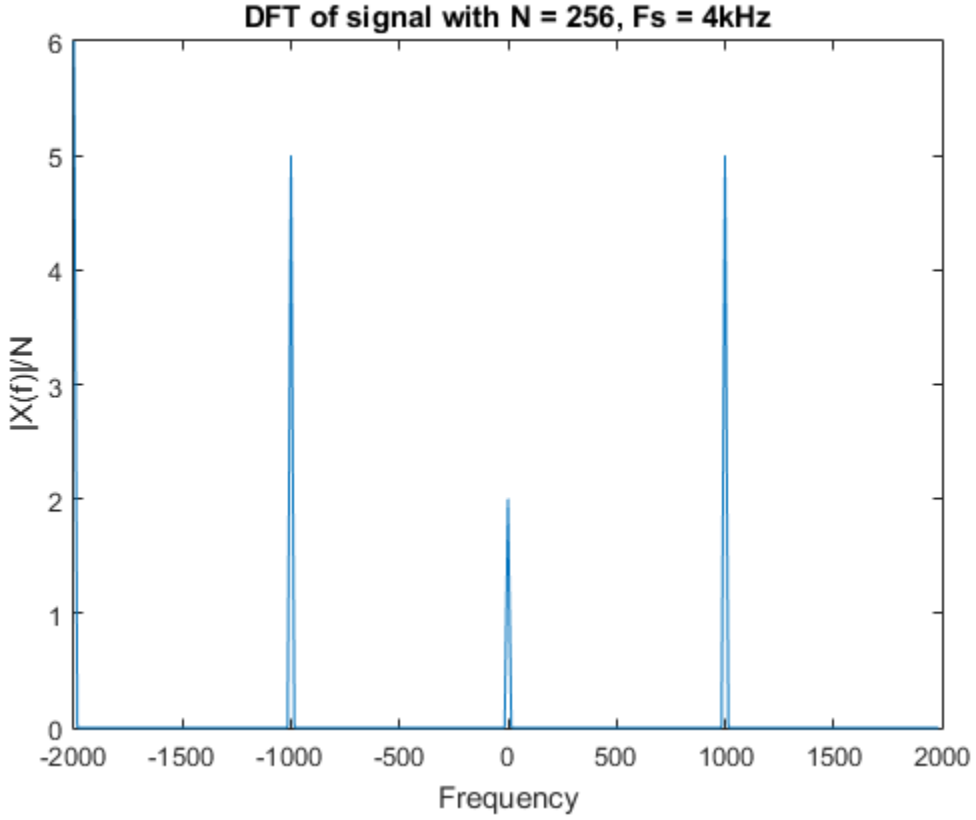


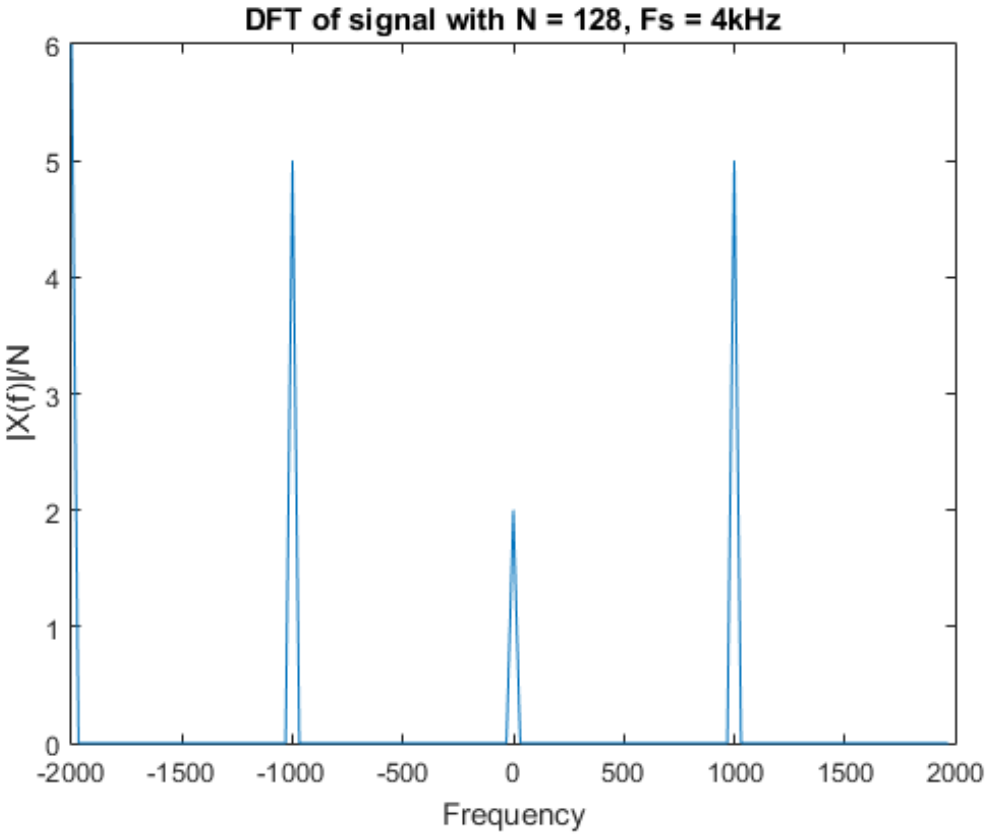
3. **<u>4kHz</u>**

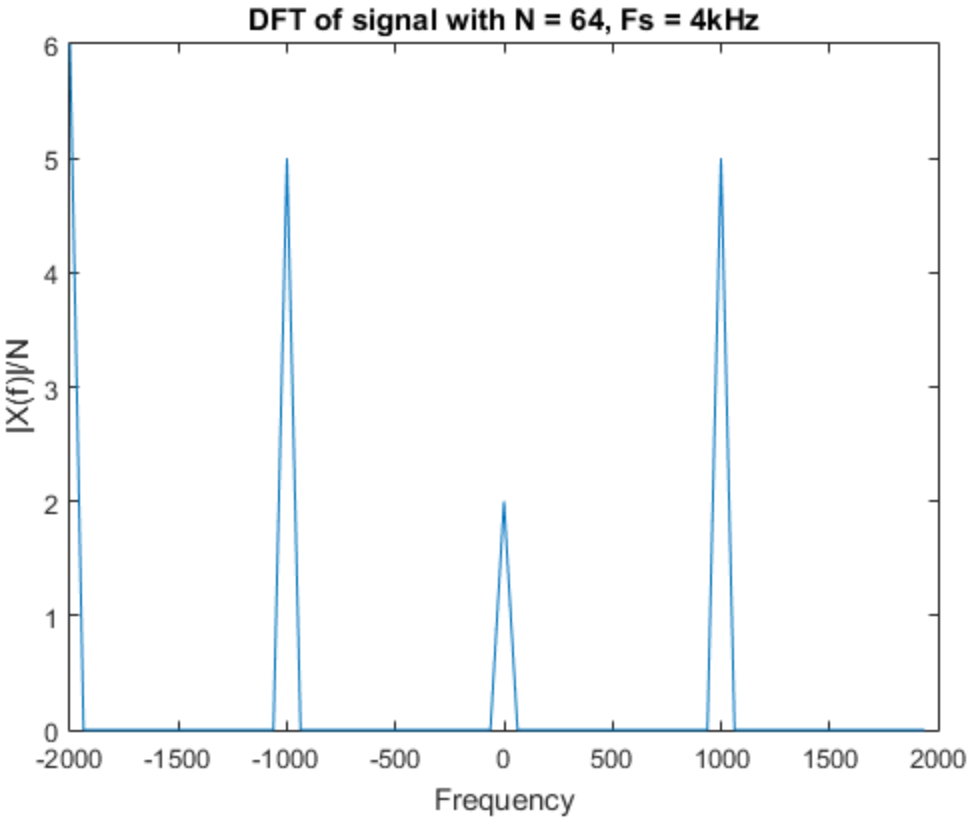


Here, the 4kHz frequency content gets aliased with 2-2=0kHz, ie. aliasing actually generates a dc component, where there was none in the original signal. This is particularly troublesome and this is the reason why when sampling below

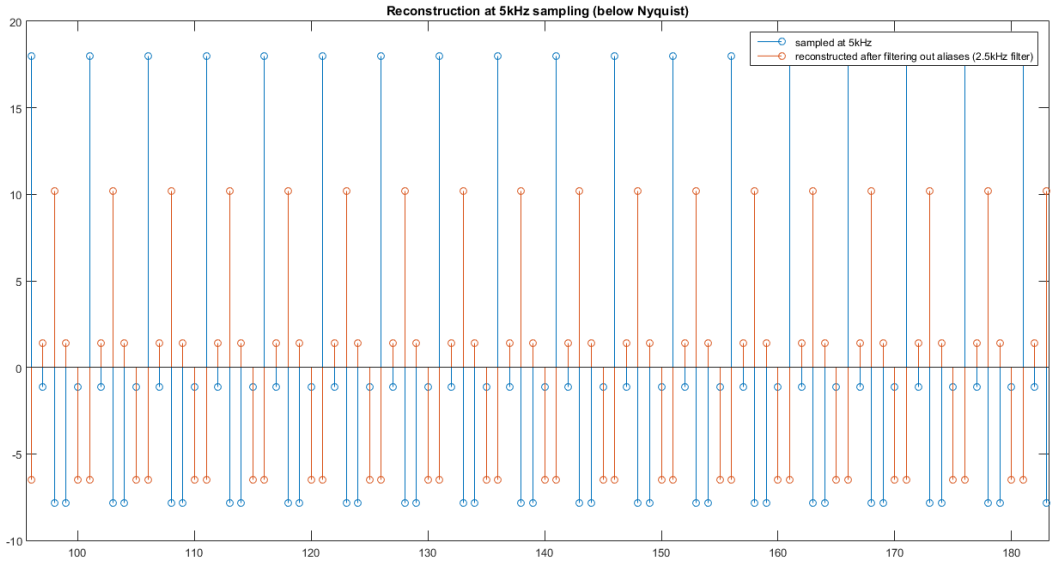
Nyquist rate, we usually low pass filter the original signal (see section on downsampling)







For 5kHz, let us try to reconstruct the signal. Here, since there is aliasing, we won't be able to reconstruct as well as in previous case. To reconstruct, we use a low-pass filter of cutoff 5/2 = 2.5kHz.



We observe that the signal reconstruction is really poor due to aliasing. The low frequency component (1kHz) has been increased and the high frequency component (4kHz) has been removed. This is evident as we don't have those sharp peaks anymore.

MATLAB Code:

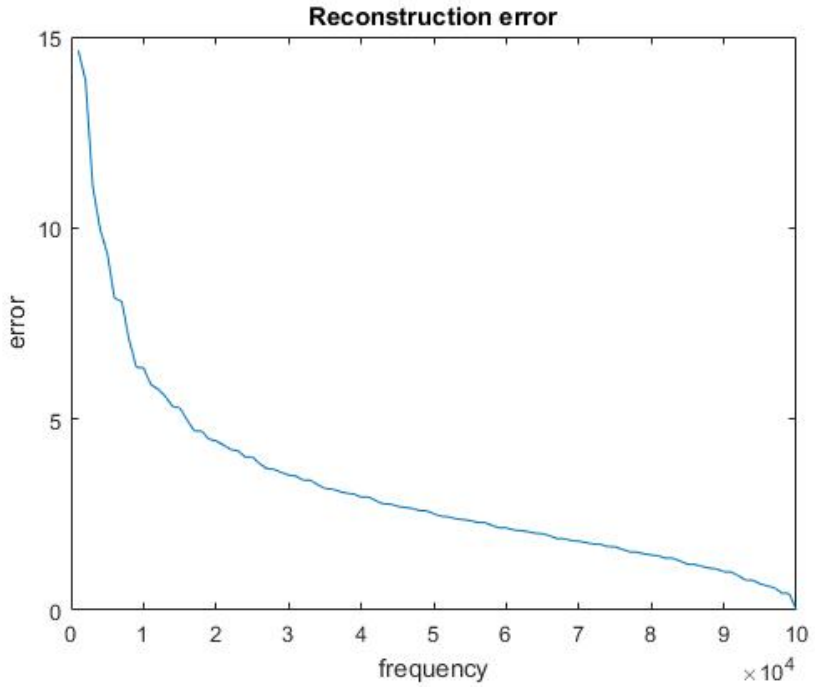
```
%sampling at different Fs
DFT(10000,12);
DFT(64,12);
DFT(128,12);
DFT(256,12);

DFT(10000,8);
DFT(64,8);
DFT(128,8);
DFT(256,8);

DFT(10000,5);
DFT(64,5);
DFT(128,5);
DFT(256,5);

DFT(10000,4);
DFT(64,4);
DFT(128,4);
DFT(256,4);
hold off

%reconstruction above Nyquist: 12KHz
ts = 0:1/12e3:(100-0.01)*10^-3;
xs=10*cos(2*pi*1e3*ts)+6*cos(2*pi*2*1e3*ts)+2*cos(2*pi*4*1e3*ts)
;
A=fir1(4,0.5,'low');
x_reconstruct1 = filter(A,1,xs);
%reconstruction below Nyquist: 5KHz
ts = 0:1/5e3:(100-0.01)*10^-3;
A=fir1(4,0.5,'low');
xs=10*cos(2*pi*1e3*ts)+6*cos(2*pi*2*1e3*ts)+2*cos(2*pi*4*1e3*ts)
;
x_reconstruct2 = filter(A,1,xs);
```

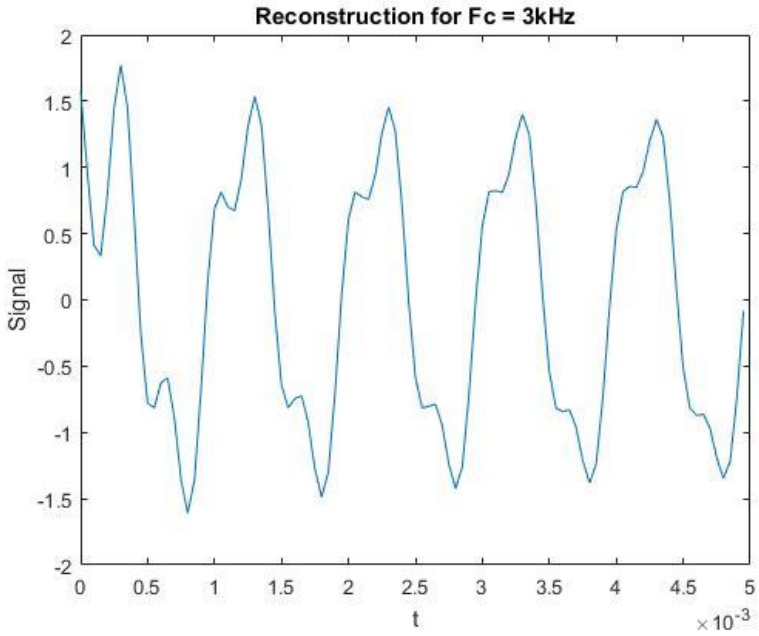
```
function DFT(L,Fs)

Fs = Fs * 1e3;              % Sampling frequency
T = 1/Fs;                % Sampling period
%L = 10000;               % Length of signal
t = (0:L-1)*T;           % Time vector
x = 10*cos(2*pi*1e3*t)+6*cos(2*pi*2*1e3*t)+2*cos(2*pi*4*1e3*t);
f = Fs *(0:L-1)/L;
y = circshift(abs(fft(x,L))/L,[0 L/2]);
figure;
plot(f- Fs/2 ,y);
xlabel('Frequency');
ylabel('|X(f)|');
ylabel('|X(f)|/N');
title(['DFT of signal with N = ' num2str(L) ', Fs = '
num2str(Fs/1000) 'kHz']);
end
```

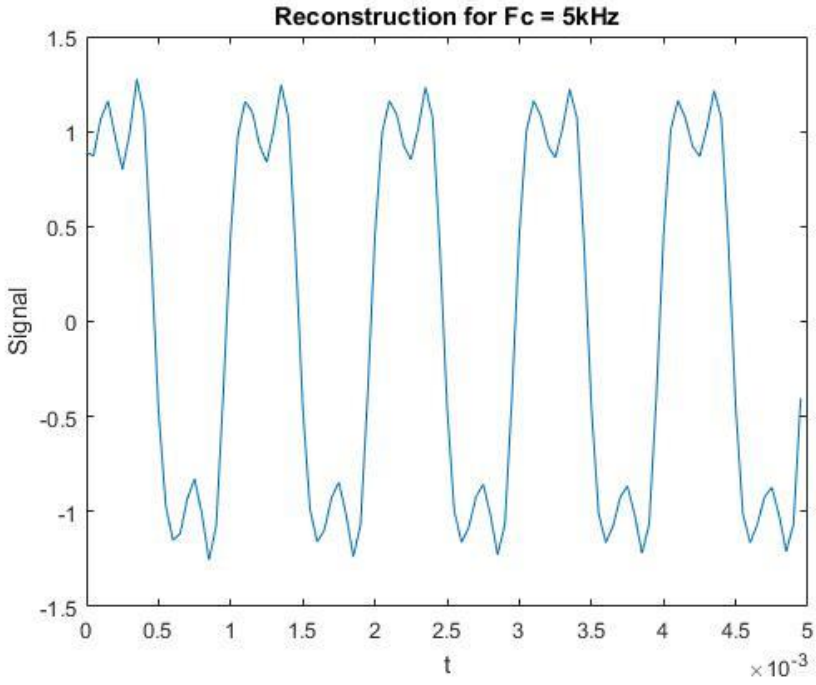
C) Sampling and reconstruction of a Square Wave

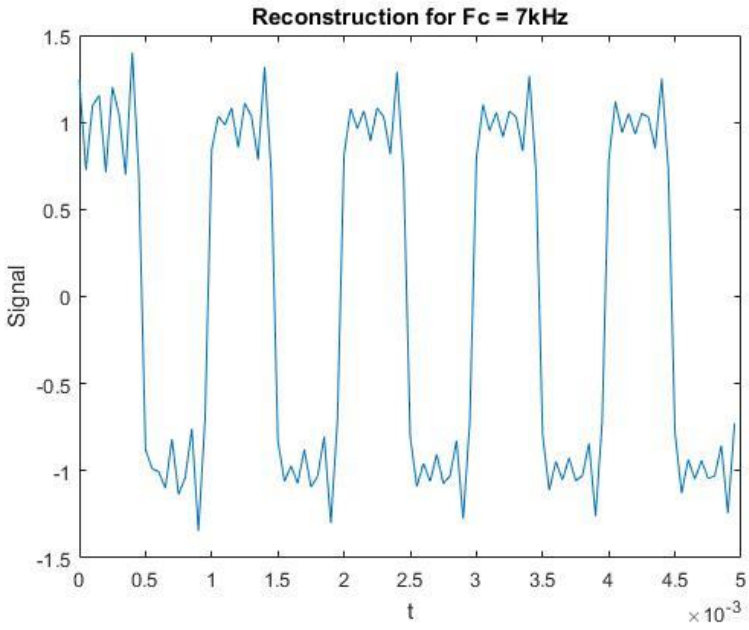
In this section, we take the original square wave sampled at 20kHz and obtain it's DFT.

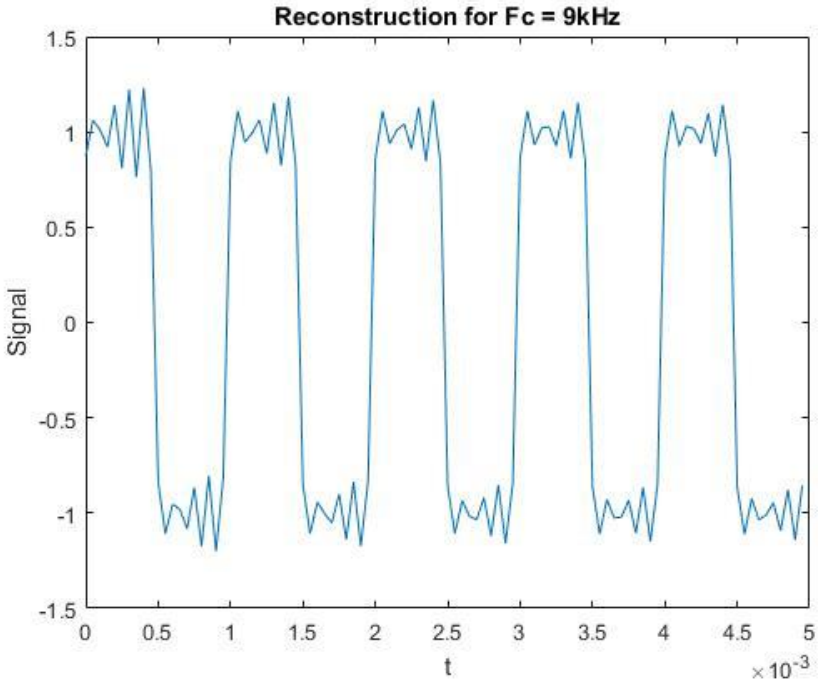
Now, we attempt to reconstruct the square wave using a filtered DFT after passing through a low-pass filter and record the reconstruction error for various cutoff frequencies.

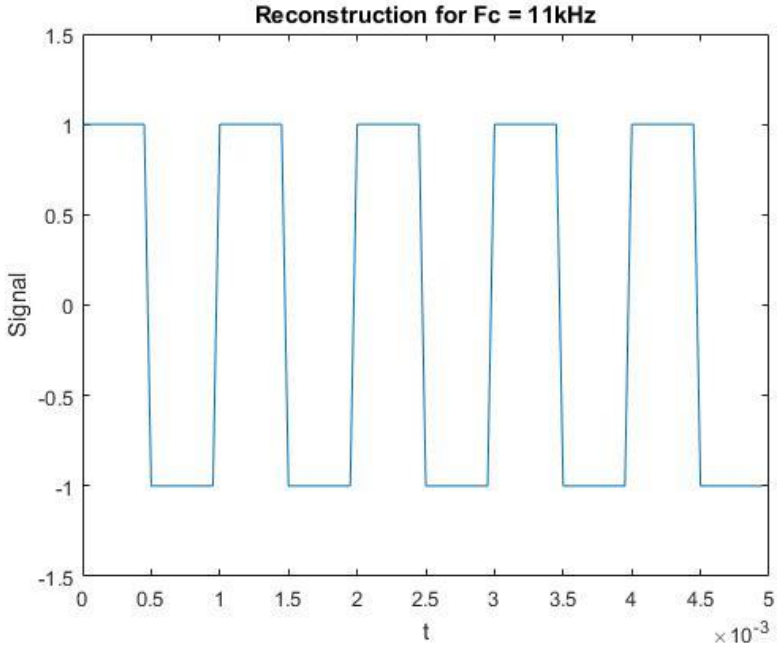





Reconstruction for Fc = 5kHz
Signal
t
×10^-3


Reconstruction for Fc = 7kHz
Signal
t
×10^-3


Reconstruction for Fc = 9kHz
Signal
t
×10^-3
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Reconstruction for Fc = 11kHz
Signal
t
×10^-3
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

MATLAB Code:

```
Fs1 = 200e3;
L = 1024;
T = 1/Fs1;
%Sampling period
t = (0:L-1)*T;
% Time vector
x1 = square(2*pi*1e3*t);

L = length(x1);
f = Fs1 *(0:L-1)/L;
f = f-Fs1/2;
err = zeros(100,1);

for Fc = 1e3:1e3:100e3

filt = zeros(L,1);
for i = 1:L
if(f(i)> -Fc && f(i)< Fc)
    filt(i) = 1;
end
end

Y_filt = fftshift(fft(x1)) .* filt';
y_filt = ifft(ifftshift(Y_filt));

error = y_filt-x1;
error = sqrt(error*error');
err(Fc/1e3) = error;
end
figure;
plot(1e3:1e3:100e3,err);
xlabel('frequency');
ylabel('error');
title('Reconstruction error');
```

D) Upsampling / Downsampling

In this section, we are going to see the effect of upsampling and downsampling a sampled signal (12kHz).

1. Upsampling

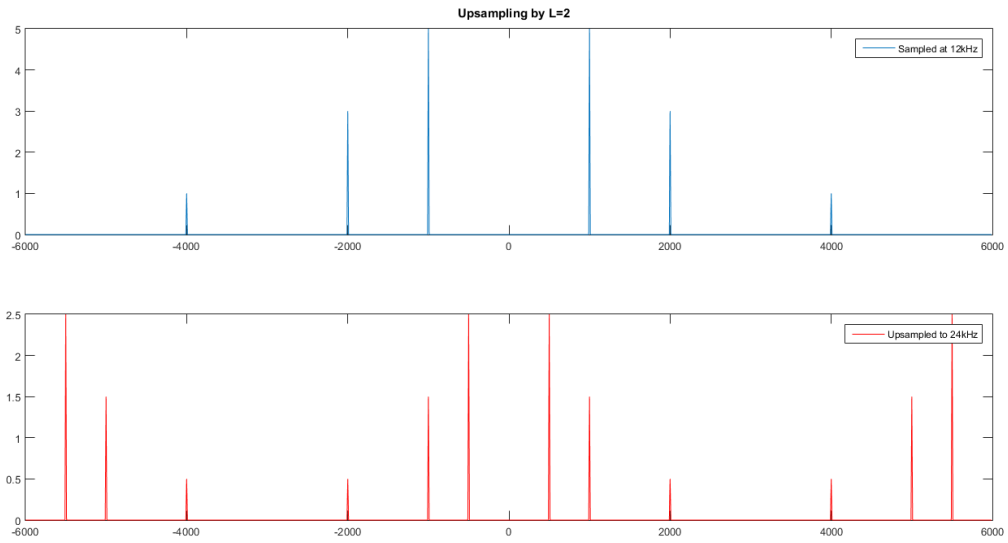
For upsampling by a factor of L, we need to interpolate (L-1) values between every two values in the currently sampled signal. This causes the sampling frequency to be scaled up L times (upsampled). This can be viewed as an increase in 'resolution' of the signal. Moreover, if the signal wasn't previously aliased, the upsampled version will be identical to the sampled version of the original signal at this upsample frequency. This is as interpolation is performed by a low-pass filter, which removes the images and the resulting spectrum basically becomes $X(e^{j2*w})$, filtered to $\mathbf{(L*F_s/2)/L = F_s/2}$, which is the same as sampling original signal at $L*F_s$ sampling rate.

Upsampling can be viewed as composed of two steps. First is expansion by insertion of (M-1) zeroes in between every two values. This expands the axis in time domain and shrinks it in frequency domain.

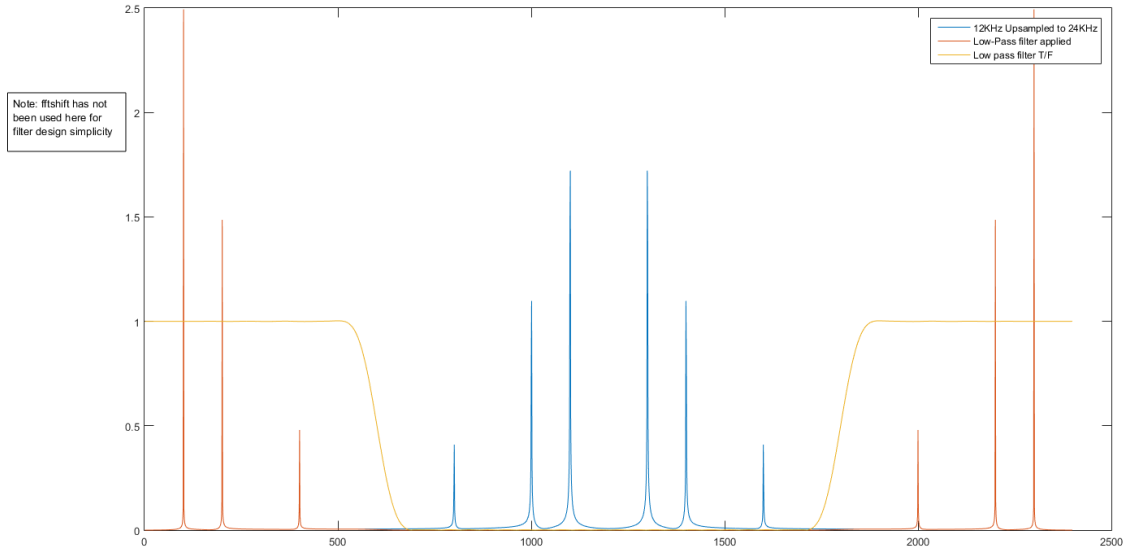
$X_1(e^{jw}) = X(e^{j2*w})$

This is then low-pass filtered to remove the 'image' frequencies. In time domain, this implies filling in the intermediate zero values with a **Sinc reconstruction kernel** (time domain response of a rectangular window lpf)

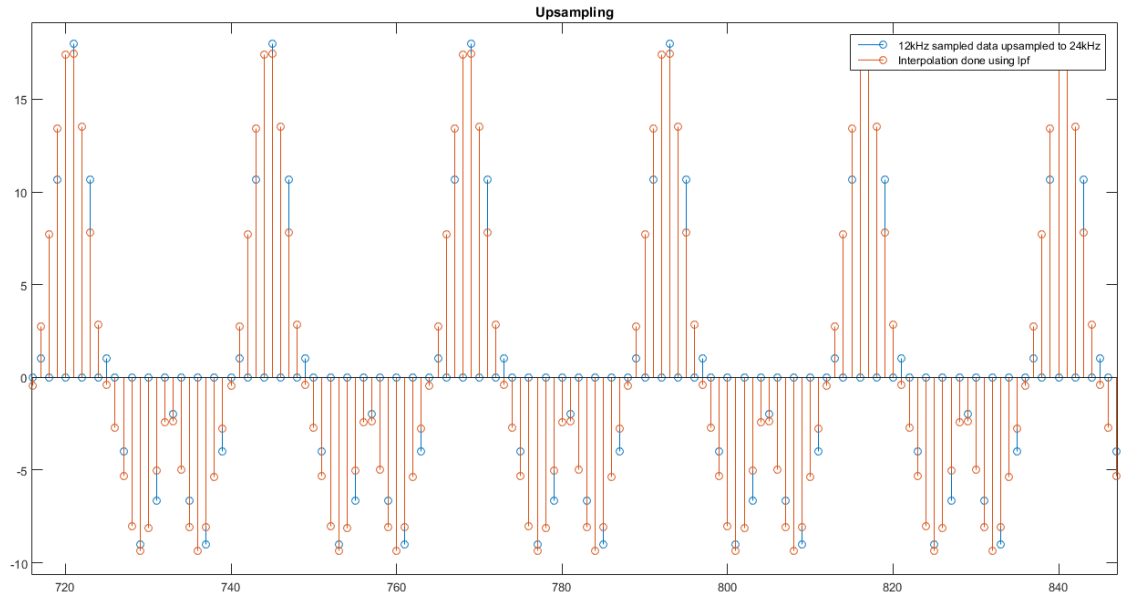
Here are the results for upsampling 12kHz to 24kHz (M=2):



This is plotted in the same frequency scale, and hence we can see that insertion of zeroes has caused compression in frequency domain. Now, let us apply a low pass filter of $(L*F_s/2)/L = F_s/2 = 2kHz$, to remove the image frequencies.



The corresponding signal in time domain is the upsampled signal:



2. Downsampling

The reverse process of upsampling, downsampling by M implies decimation in time domain (taking 1 out of every M values). This causes an expansion by M in the frequency domain and overlap of M such expanded frequency spectrums (can be derived from analytic expression of DTFT). This in turn causes aliasing, which corrupts the frequency content.

Original sampled spectrum: centered around 0, F_s, $2F_s$ …..

On downsampling:

Term 1: centered around 0, $M*F_s$, $2M*F_s$ ….

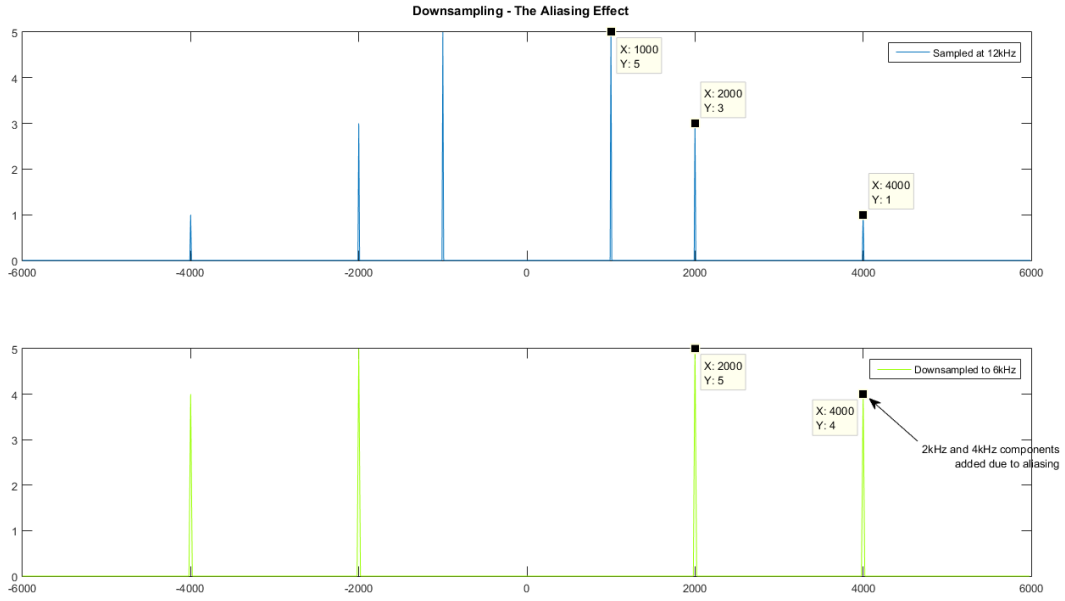
Term 2: centered around F_s, $(M+1)*F_s$, $(2M+1)*F_s$ ….
Term M: centered around $(M-1)*F_s$, $(2M-1)*F_s$, $(3M-1)*F_s$ ….

In this case, to avoid aliasing, we need to have no frequency overlaps in all of the above terms. For this, the original signal must be bandlimited to

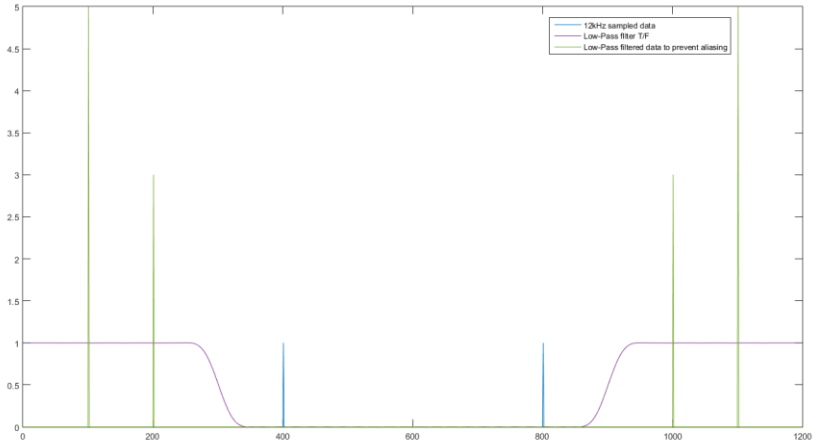
$f_{m,new} = (F_s/2)/M$ and not just $f_{m,\ old} = F_s/2$

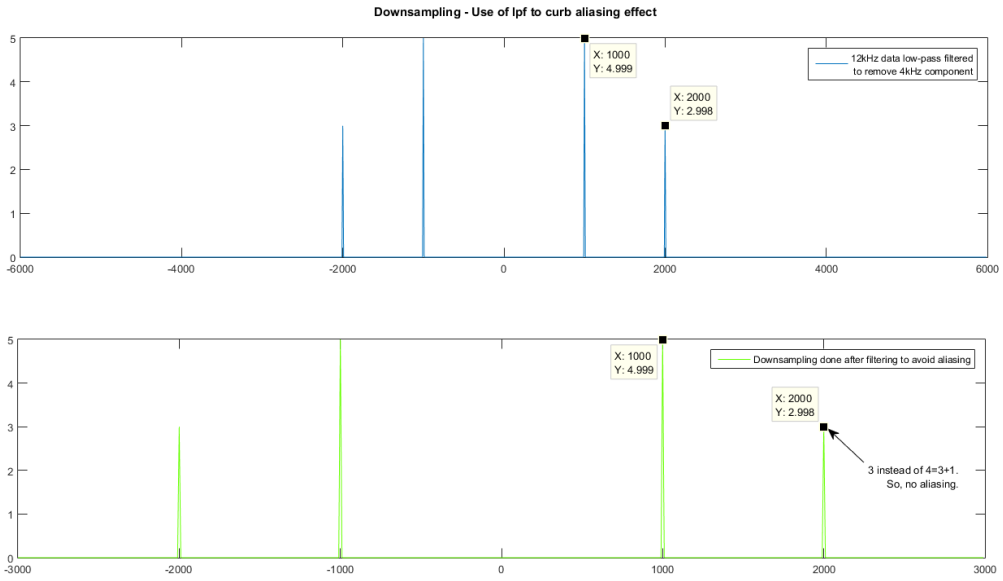
This can be interpreted as $f_{m,new} = (F_s/M)/2$, as F_s/M is the new sampling frequency and $(F_s/M)/2$ is the new Nyquist sampling rate and to avoid aliasing, max frequency should be less than Nyquist rate of sampling frequency.

This may not always be true and to avoid aliasing, we first low-filter the signal (cutoff = $(F_s/2)/M = (12/2)/2 = 3$kHz). Then we do downsampling by M to get our new signal.

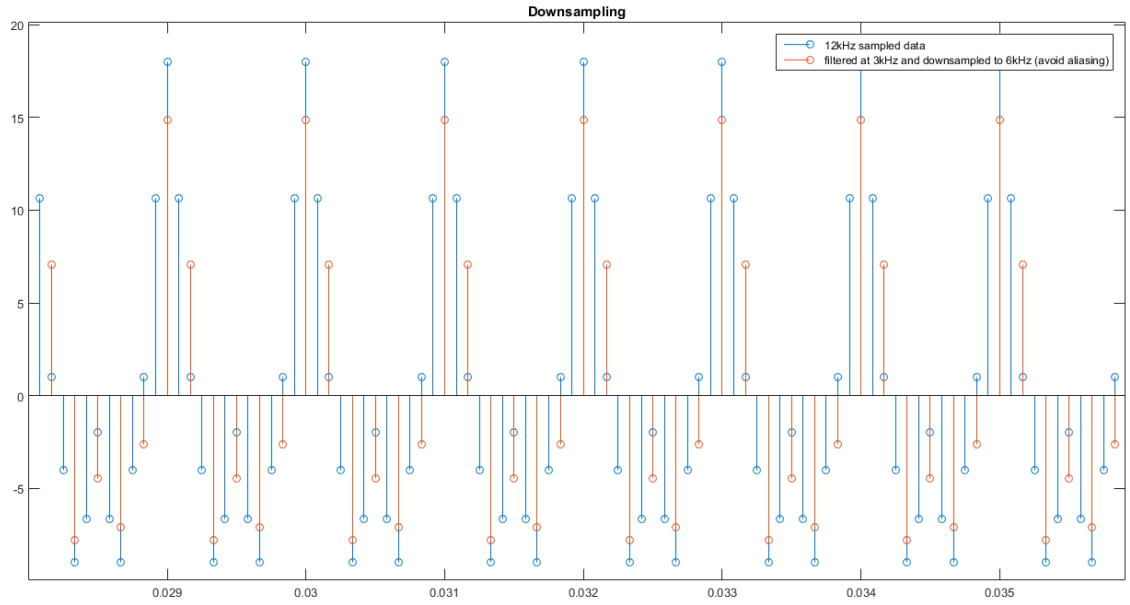


So, apply filter first and then downsample.





In time domain:



The downsampled version has been plotted in an expanded time scale for comparison purposes. As we can see, alternate values have been decimated. However, apart from a certain delay and scaling, the downsampled signal seems to be pretty similar to the 12kHz sampled signal. This is thanks to the low pass filter, which enabled us to remove the troublesome frequencies that were getting aliased.

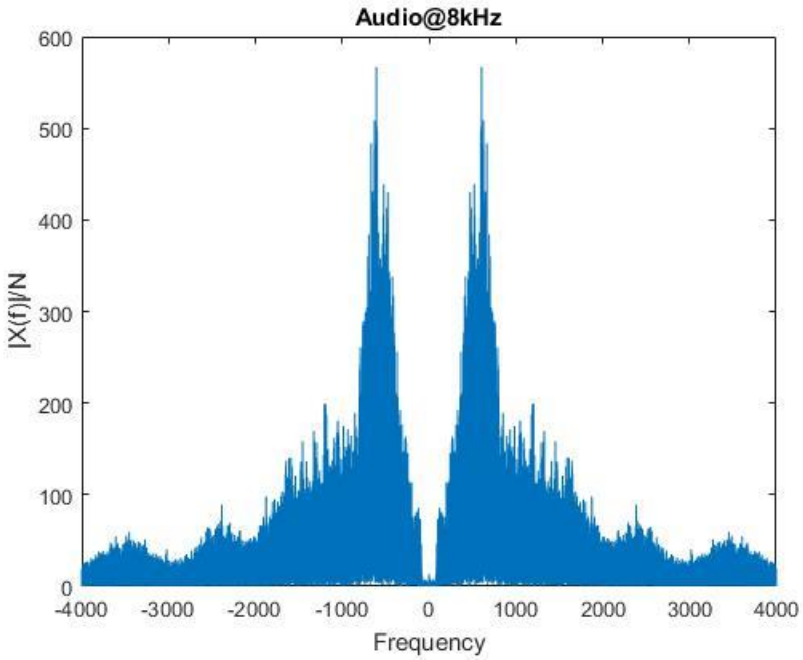
MATLAB Code:

```
%% upsampling
xu=zeros(1,2399);
tu=0:1/(2*Fs):(100-0.05)*10^-3;
xu(1:2:2399)=x1(1:1200);
yu = (fft(xu,2399)/2399);
A = fir1(49,0.5,'low');
%{
figure
plot(abs(yu))
hold on
plot(abs(fft(upsample_reconstruct,2399)/2399));
hold off
%}
upsample_reconstruct=2*filter(A,1,xu);  %2 factor put to
compensate for amplitude division in upsampling
figure
stem(t1,x1);
hold on
stem(tu,upsample_reconstruct);
hold off


%% downsampling
y1=(fft(x1,1200)/1200);
A = fir1(49,0.5,'low');
xfilt=filter(A,1,x1);

xd=xfilt(1:2:1199);
td=0:2/Fs:(100-0.05)*10^-3;
yd = fftshift(fft(xd,600)/600);
```

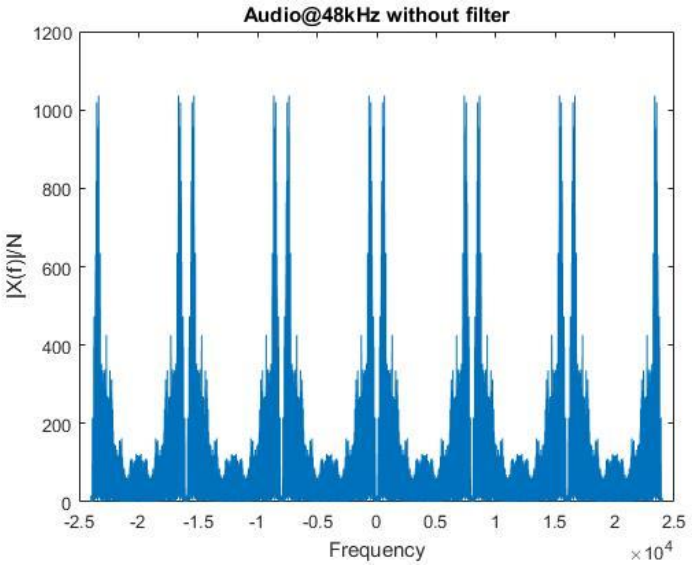
E) Audio Upsampling/Downsampling/Mixing

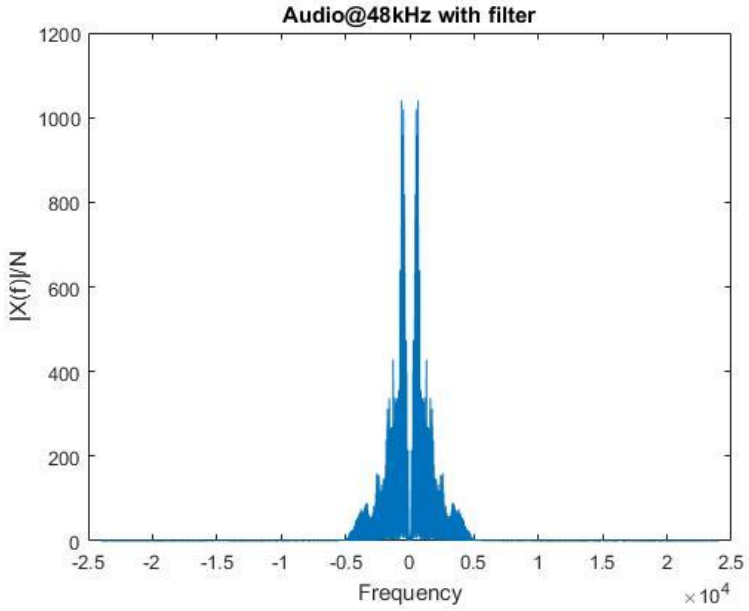
In this section we are going to perform various operations of audio/speech signal.

The music signal is taken from the MATLAB DSP demo folder and is sampled at 48kHz. The speech/audio signal is taken form the internet at is sampled at 8kHz.

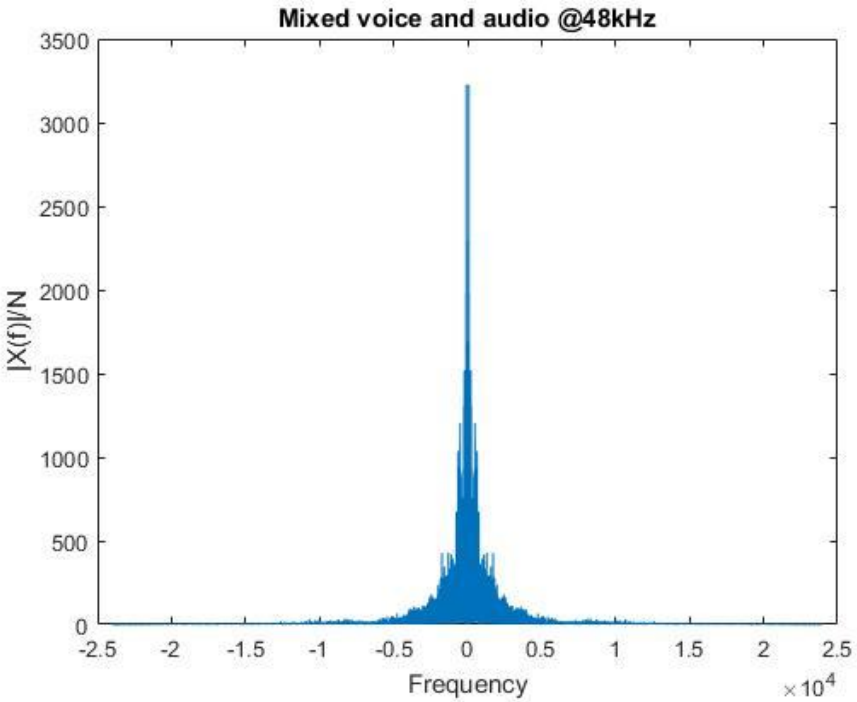


- In the first part, we upsample the speech signal to 48kHz. This operation involves upscaling the signal by a factor of 6 by adding 5 zeros in between each sample and then performing a Low-Pass filter operation to remove all the image waveforms produced in the frequency domain.



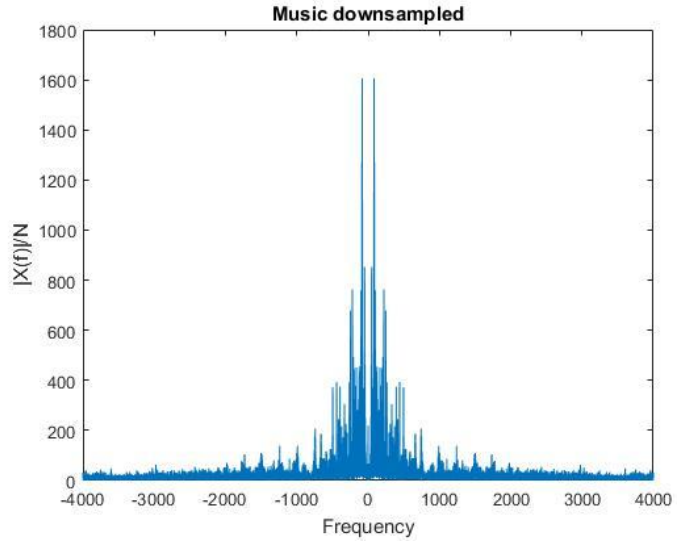


- Now that we have the speech signal at 48kHz, we can add this to the music which is also sampled at the same frequency.

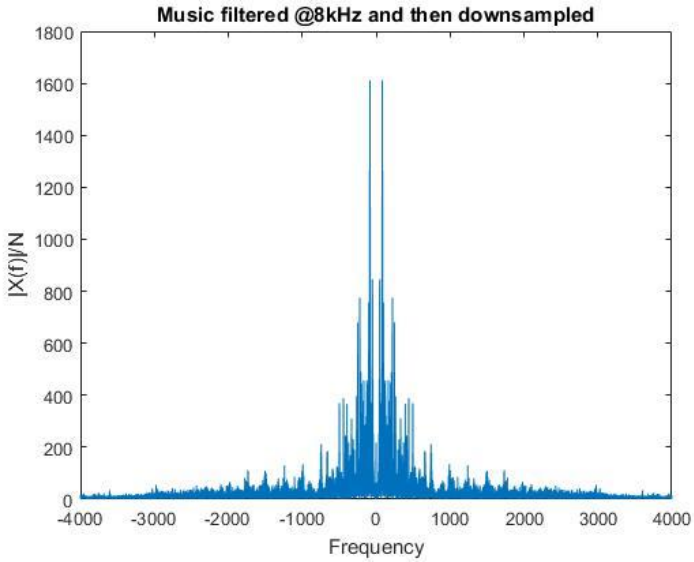


Upon listening to this mixed audio signal, we can notice that there is no aliasing or corruption of sound.

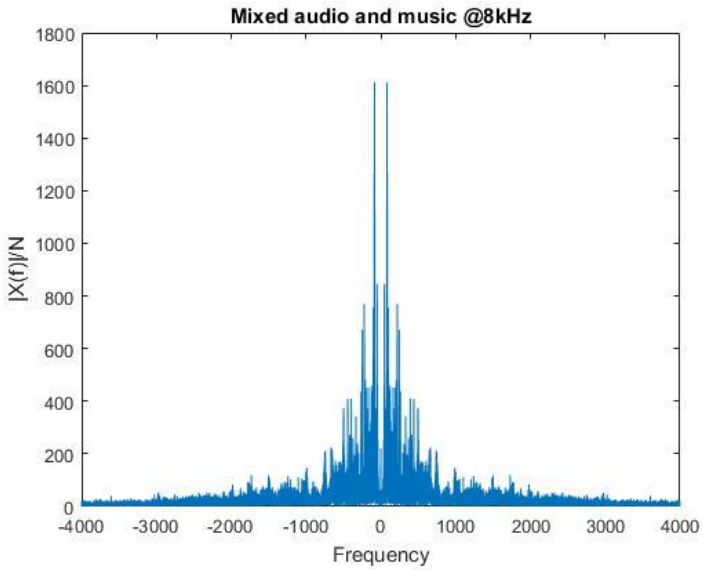
- Now the music signal is downsampled to 8kHz and heard, we can notice significant corruption in the quality of the signal.



- The music signal is first filtered with a low-pass filter of 8kHz and then downsampled.



- As the last part the downsampled music is added to the speech signal and played. We can notice significant difference in the quality of sound compared to the upsampled case. The higher frequency sounds are corrupted.



MATLAB Code:

```
[x1,Fs1]=audioread('audio48kHz.wav');
% Reading sounds recorded at 48kHz
[x2,Fs2]=audioread('male_8kHz.wav');
% Reading voice recorded @ 8kHz

x2u = 6*upsample(x2,6);
%upsampling by adding Zeros
x2u=x2u(1:size(x1,1));
A=fir1(48,1/6,'low');
%LPF Coefficients
x2u_filt=filter(A,1,x2u);
%filtering upsampled voice
y1=x1/3+x2u_filt;
%Mixing sounds
x1d=downsample(x1,6);
%Downsampling Music
x1d_filt=downsample(filter(A,1,x1),6);
%filtering Music @8kHz and then downsampling
y2=x1d_filt+x2(1:size(x1d_filt,1));
%Mixing sounds @8kHz

plotdft(x2,Fs2,'Audio@8kHz');
%sound(x2,Fs2)
% Voice recording at 8kHz

plotdft(x2u,Fs1,'Audio@48kHz without filter');
%sound(x2u,Fs1)
%voice recording upsampled but not filtered

plotdft(x2u_filt,Fs1,'Audio@48kHz with filter');
%sound(x2,Fs2),sound(x2u_filt,Fs1)
%voice recording upsampled
```

```
plotdft(y1,Fs1,'Mixed voice and audio @48kHz');
%sound(y1,Fs1)
%adding both sounds @ 48kHz

plotdft(x1d,Fs2,'Music downsampled');
%sound(x1d,Fs2)
%music downsampled

plotdft(x1d_filt, Fs2, 'Music filtered @8kHz and then downsampled')
%sound(x1d_filt,Fs2)
%music filtered at 8kHz and then downsampled

plotdft(y2, Fs2, 'Mixed audio and music @8kHz')
%sound(y2,Fs2)
%music downsampled and added
```

Discussions:

- For upsampling, we need a low pass filter after upsampling to $L*F_s$, with cutoff $F_s/2*L$. For downsampling, on the other hand, we first need to band limit it to $F_s/2*M$ to avoid aliasing and then decimate the values. Upsampling and downsampling are very common occurrences in signal processing, especially when performing tasks like audio and speech mixing.
- We see that reconstruction error of square wave decreases with increase in sampling frequency as the number of harmonics unaliased increases (higher harmonics have very small value and do not matter, neither from frequency content perspective nor from aliasing perspective)
- In upsampling, as we are introducing images, magnitude gets halved and we need to multiply by 2 again.
- Sampling below Nyquist rate can be done intelligently with use of band pass filters to avoid aliasing. This is as the frequency content is present as single tones with sufficient gap in between. The only problem is that the band pass filters need to be highly selective here and traditional FIR filters are unable to meet these requirements.
- While mixing the audio with the speech signal, when we first downsample the audio and then mix it with speech, we hear a clear distortion in the sound.