Suez Canal University

Faculty of Engineering

Electrical Engineering Department

Analog communication

Superheterodyne receiver

a) Discuss how you designed each of the blocks of the system:

1.AM Modulator

Discussion:

First the transmitter gets the five audio signals in a digital form sampled what we do is reading them by matlab then turning them from stereo to mono form because the receiver is monophonic so no need for stereo then we equalize all signals to be the same length so we can do the same operations on all of them then we modulate every audio signal at specific carrier freq and then generate FDM signal considering that maximum freq we reach < (Sampling frequency /2) to not violate the Nyquist criteria so we needed to upsampling the signals by factor 16 to achieve that for all signals and then sending them.(reason for 16 in Oscillator part)

2.The RF Stage

Discussion:

RF stage Tuning :We made BandPass filter selects the desired signal with bandwidth much bigger than the bandwidth of the signal (The nature of this filter at high frequency operation)

We chose Suitable values for filter parameters to avoid image frequency interference.

Filtering the FDM signal to get the desired station from other stations signals
At Modulator section we make sure that frequency carrier difference between every signal is high enough to give RF filter good range between signals to operate.

3. Local Oscillator

Discussion:

in this part we generate a carrier signal with frequency = fn+IF to adjust the signal we picked after RF filter to be centered at low intermediate frequency IF which IF Filter will deal with it next

We made sure that the high frequency folded back signal component would not go deeper in lower frequencies and interact with our signal centered at IF frequency by considering that Folded freq = (Fs - Fmax) is bigger than IF + Bandwidth of our signal and we adjusted Fs to be multiplied by factor 16 to achieve this for the five signals.

Mixer

Discussion:

Here we just multiply the catched signal with the generated Oscillator carrier signal

The IF Stage

Discussion:

Here we filter the desired signal with BandPass filter centered at IF and its Bandwidth is the same as our signal BandWidth.

We adjusted parameters of it to be high selective filter as it should be to avoid involving the folded back frequencies and by doing this with the adjusted Fs we successfully avoided adjacent channel interference.

4.Baseband Detection Stage

Discussion:

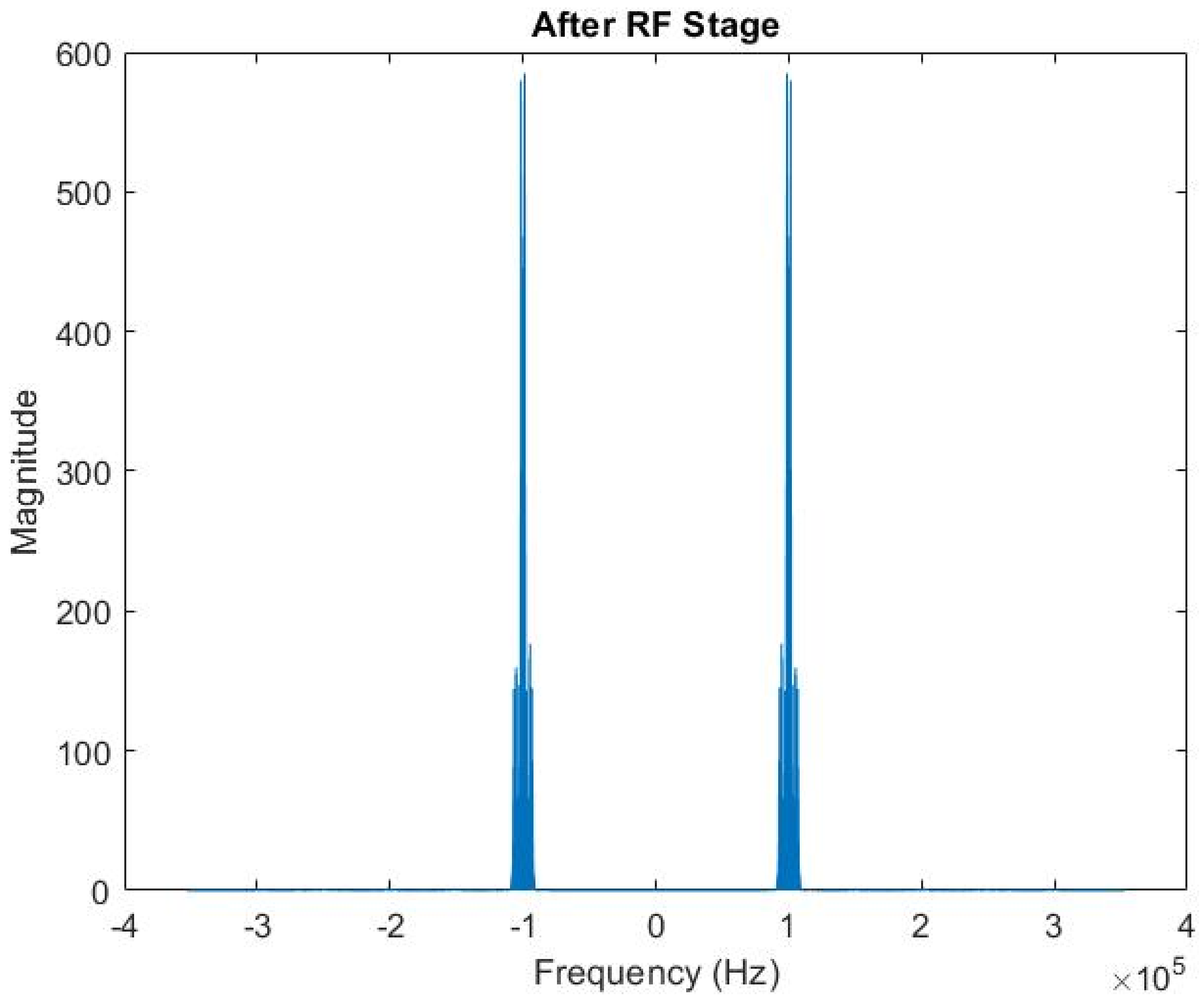
This stage thus involves mixing the signal we well attached with a carrier of IF frequency to demodulate the signal to be centered on its base frequency with 2 copies of it at ±2*IF frequencies due to the last carrier we added and then filtering the signal using a low-pass filter (LPF) with bandwidth equal to our signal bandwidth to get our main signal in its base band and cut the other two components.

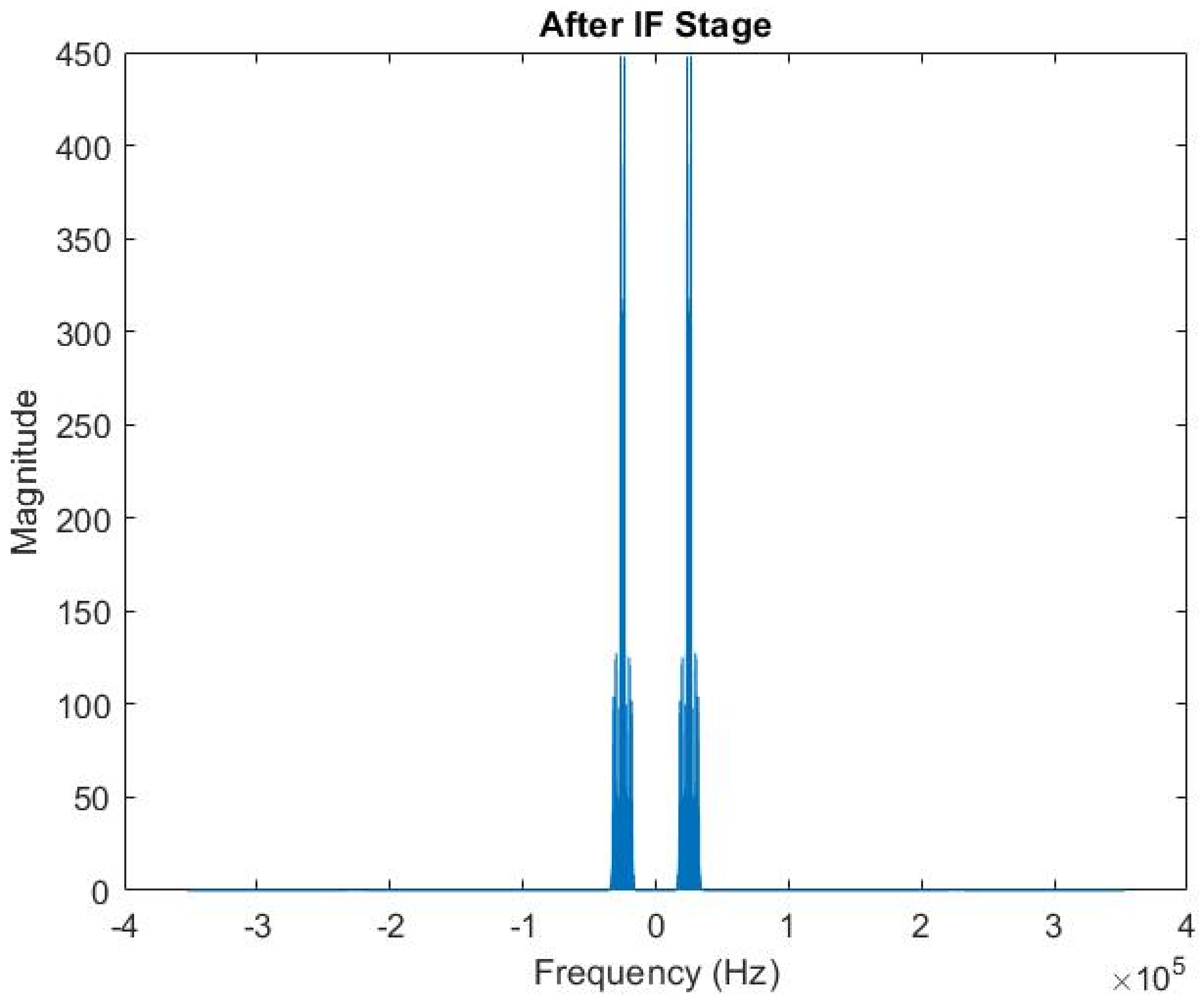
b) *1.In two or three sentences, discuss the role of the RF, the IF and the baseband detector. Indicate why we need the IF stage?*

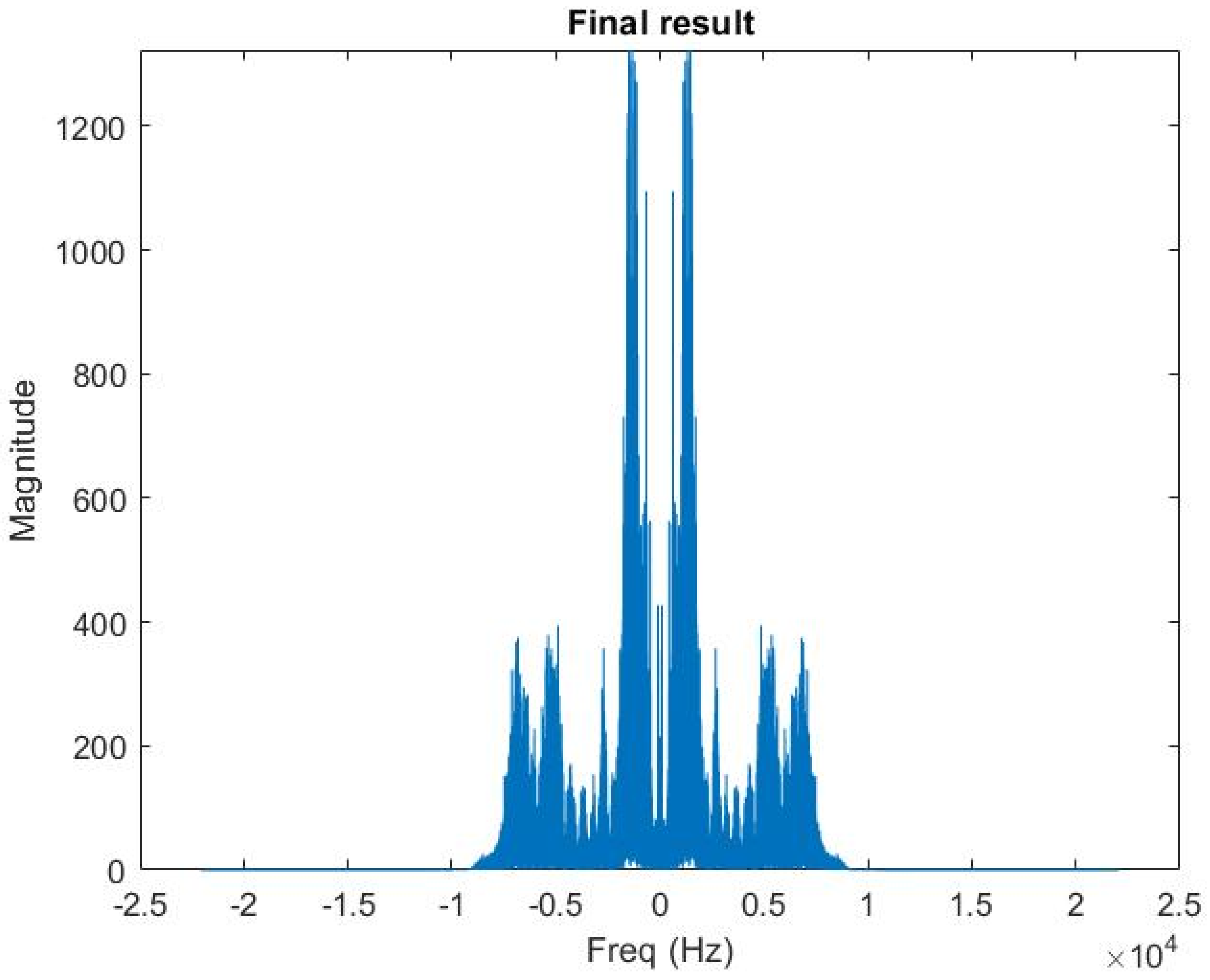
- **RF stage:** Consists basically of a tunable filter and an amplifier that picks up the desired station by tuning the filter to the right frequency band. The main role of RF section is image frequency rejection
- .
- **IF stage:** IF section can effectively suppress adjacent-channel interference because of its high selectivity. It operates at fixed low frequency makes it easier to amplify and filter the signal.
- **Baseband detector:** Demodulates the signal out from IF stage to recover the original information-carrying baseband signal.

We need IF Stage because it is filtering at a lower frequency. This is simpler and more efficient compared to working only at the high RF stage frequencies where filters are more difficult to design and implement and also its high selectivity.

2. *Suppose you want to demodulate the first station (i.e. at $\omega 0$), plot the spectrum of the outputs of the RF, the IF and the baseband stages.*


After RF Stage
Magnitude
600
500
400
300
200
100
0
-4
-3
-2
-1
0
1
2
3
4
Frequency (Hz)
$\times 10^5$


After IF Stage
Magnitude
Frequency (Hz)
$\times 10^5$
0
50
100
150
200
250
300
350
400
450
-4
-3
-2
-1
0
1
2
3
4

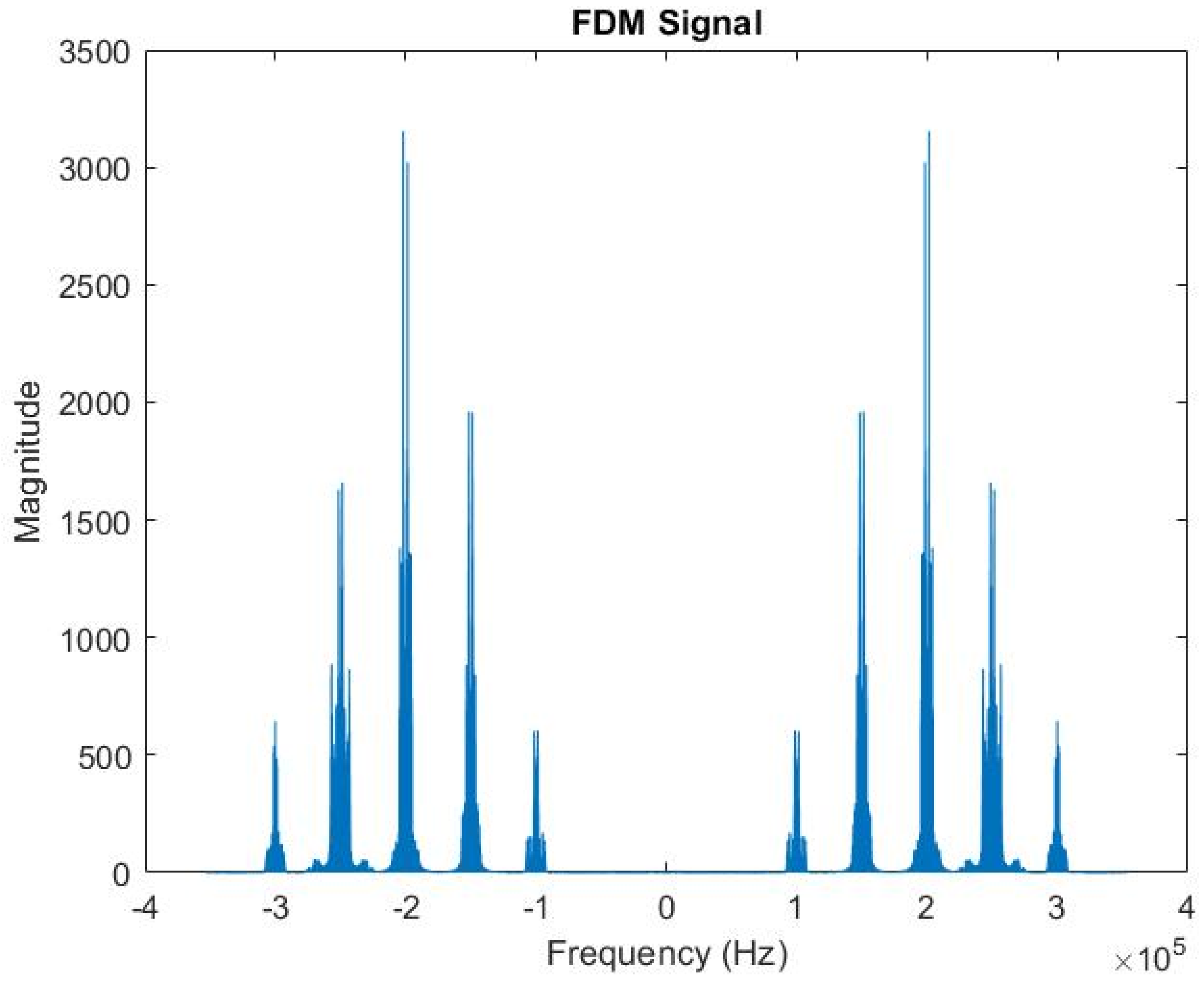

Final result
Magnitude
Freq (Hz)
×10^4
0
200
400
600
800
1000
1200
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5

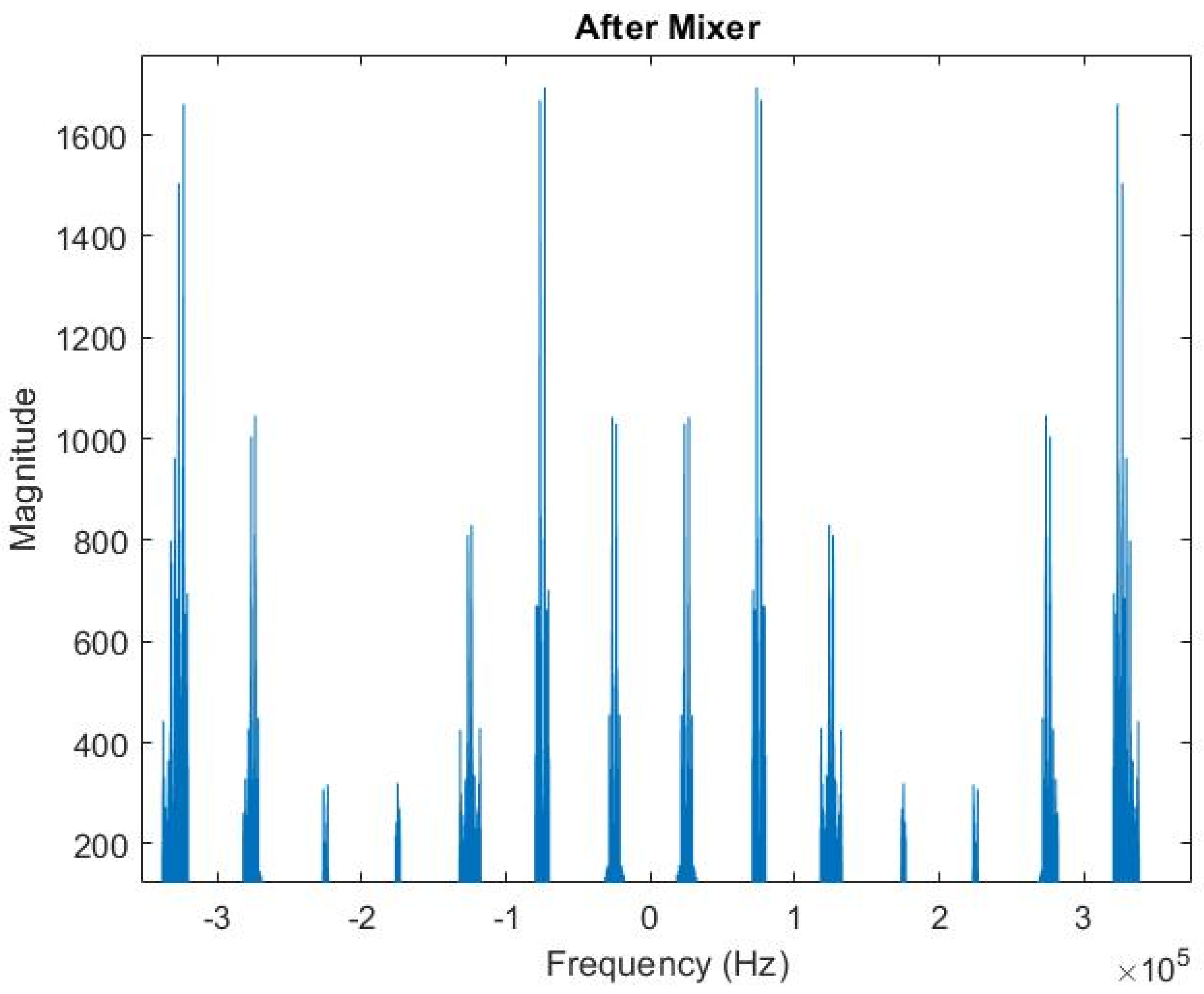
3. *Use the command ‘sound’ on the demodulated signal and check whether you can successfully listen to the radio station. Please comment about this step in your report.*

Commment : sound is fine, design could deal with it perfectly!!

4.*Repeat parts 2 and 3 but after removing the RF BPF. That is, the RF stage does not exist, what would happen if you try to demodulate the station at ωo?*

>>>>>>>>> interference between first two stations that RF stage was rejecting(no image frequency rejection) , stations 1 & 2 sounds at the same time.

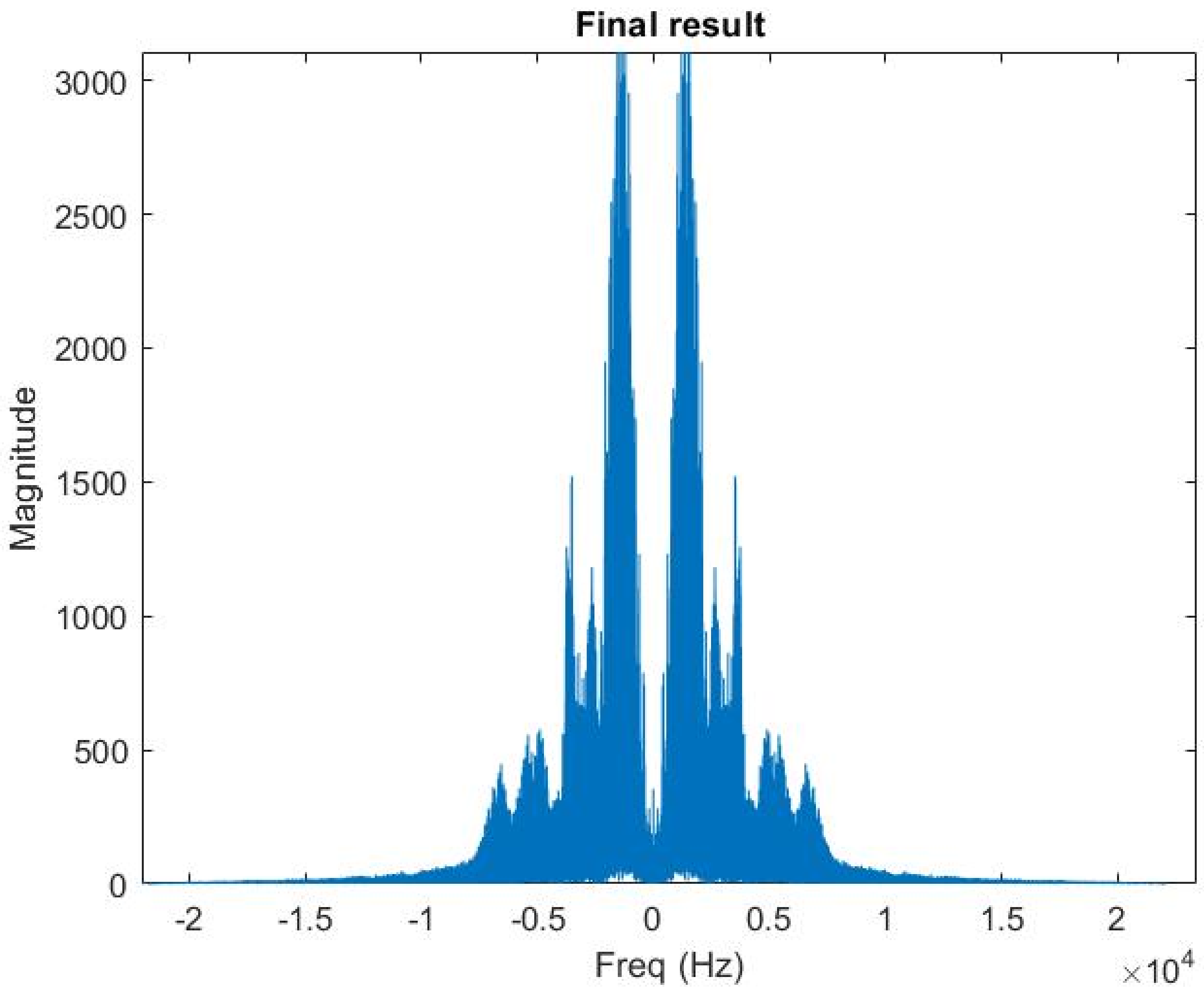

FDM Signal
Magnitude
Frequency (Hz)
×10^5
3500
3000
2500
2000
1500
1000
500
0
-4
-3
-2
-1
0
1
2
3
4


After Mixer
Magnitude
1600
1400
1200
1000
800
600
400
200
-3
-2
-1
0
1
2
3
Frequency (Hz)
$\times 10^5$

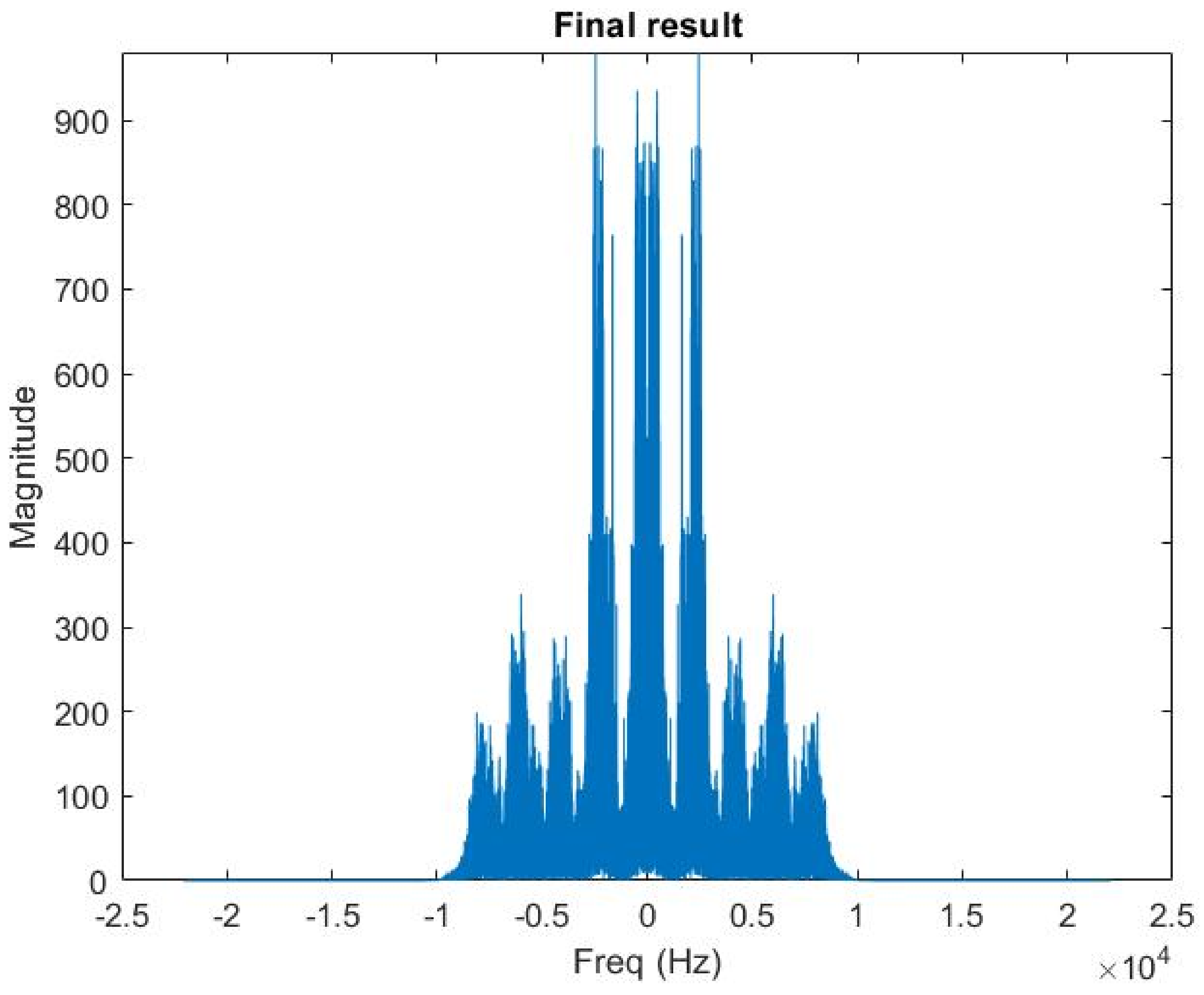
.

5.What happens (in terms of the spectrum and the sound quality) if the receiver oscillator has frequency offset by 0.1 KHz and 1 KHz?

offset by 0.1 KHz : much noise added to final spectrum and sound badly affected with much noise, This is due to frequency shift between oscillator carrier and Baseband carrier.



offset by 1 KHz : almost whole final spectrum was changed and sound has been unrecognized with this aggressive offset. This is due higher frequency shift between oscillator carrier and Baseband carrier



--CODE--

```
clear ; clc ; close all
```

Specifing the filenames of the audio files

```
Audios = ["Short_BBCArabic2.wav", "Short_FM9090.wav", "Short_QuranPalestine.wav",
"Short_RussianVoice.wav", "Short_SkyNewsArabia.wav"]; % names of audio files
```

Obtaining maximum length

```
% Initialize variables to store maximum length and corresponding filename


max_audio_length = 0;


% Loop through each audio

for i = 1:length(Audios) % i = 1:5

% Read the audio file and obtain the audio data

audio_signal = audioread(Audios{i});

% Calculate the length of the audio file (total number of samples)

audio_length = size(audio_signal, 1); %% '1' refers to one channel( one coloumn )

% Check if this audio file has the maximum length so far

if audio_length > max_audio_length

max_audio_length = audio_length;

end

end
```

Padding and Monoizing Audios

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

% Read the audio file and obtain the audio data

[audio_signal, Fs] = audioread(Audios{i}); % getting audio data and sampling frequency

audio_signal = sum(audio_signal, 2) / size(audio_signal, 2); % [2]Convert from stereo
to mono ( The sum function adds the two channels & The size function determines the
number of channels)


% Padding short signals

audio_signal(end + max_audio_length - length(audio_signal)) = 0; % [3]short channels
will be padded with zeros at the last remaining samples of its length after subtracting
from maximum length
```

```
audiowrite(Audios{i}, audio_signal, Fs); % save the padded monoized audio signals
end
```

FFT Of Desired Audio At The Beginning

```
fprintf("Choose one of these channels:\n1. Short_BBCArabic2\n2. Short_FM9090\n3.
Short_QuranPalestine\n4. Short_RussianVoice\n5. Short_SkyNewsArabia\n");


choose_channel = input("Choose: ");


[audio_signal, Fs] = audioread(Audios(choose_channel)); % Read audio and get sampling
frequency
Fs % [1] display sampling freq to use in hand calculations
AUDIO_SIGNAL = fftshift(fft(audio_signal, length(audio_signal))); %% to be symmetric
around 0 (has mathmatical meaning)
Frequency_vector = (-length(AUDIO_SIGNAL)/2 : length(AUDIO_SIGNAL)/2 - 1)'; % adjust
frequency axis (we converted it from row to column by (') because we next will divide
it by the AUDIO_SIGNAL array which is a column array and dividing amd multiplying must
be in same type)
F= Frequency_vector*Fs/length(AUDIO_SIGNAL); % [4]Freq axis ( freq limits [-Fs/2 --->
Fs/2] exceeding this range will cause frequencies to fold back )
plot(F, abs(AUDIO_SIGNAL)) % [5] plotting FFT
title(Audios(choose_channel) + " FFT")
xlabel("Freq (Hz)")
ylabel("Magnitude")
ylim([0 max(abs(AUDIO_SIGNAL))])
```

Obtaining BandWidth

```
N = length(AUDIO_SIGNAL);
[pks, freqs] = findpeaks(abs(AUDIO_SIGNAL(1:N/2)), F(1:N/2), 'MinPeakHeight',
0.001*max(abs(AUDIO_SIGNAL(1:N/2)))); % save frequencies and corresponding peaks which
achieve the threshold(0.001*max) in two different arrays


% Compute bandwidth
bandwidth = max(freqs) - min(freqs);


disp("bandwidth of " + Audios(choose_channel) + " = " + bandwidth + " Hz"); %[6] BW
```

Modulating

```
fo = 100000;

n = (choose_channel-1);

delta_f = 50000;

fn = fo + n*delta_f;

%Fs = 44.1k hz and Fmax(highest freq we reach in whole code) = 2fn(Signal(5))+IF +
Bandwidth of signal 5 = 625k +9.11k = 634.11k hz


%Folding frequencies happens when we reach frequency exceeds Fs/2 and frequencies above
that will fold back to lower frequencies


%aliasing happens when folded back frequencies start to interfernce with our signal
which centered at 25k hz (after mixer) so we need to increase sampling frequency by
multipling Fs by a factor (x) to avoid this


%F(aliasing) = sampling Freq - Fmax = (x*Fs) - Fmax "should be bigger than 25k +
bandwidth of signal 5"


% x*44.1k - 634.11k > 34.11k


% x > 15.15 SO -------- x = 16


audio_signal = (1/16)*interp(audio_signal, 16); %[8] Fs(new)= 16*Fs & length(new) =
16*length & magnitude(new)=16*magnitude so we divided by 16 to not change magnitude

audio_length = (1:1:length(audio_signal))'; % adjusted to be the same length as our
signal

carrier_signal = cos(2*pi*fn*audio_length*(1/(16*Fs)));% [7] carriar signal cos(     )

modulated_signal = carrier_signal.*audio_signal; % we use (.*) when multiplying arrays


MODULATED_SIGNAL = fftshift(fft(modulated_signal));

Frequency_vector = (-length(MODULATED_SIGNAL)/2:1:length(MODULATED_SIGNAL)/2-1)';

plot( Frequency_vector*(16*Fs)/length(MODULATED_SIGNAL),abs(MODULATED_SIGNAL) )

title( Audios(choose_channel) + " Modulated")

xlabel("Freq (Hz)")

ylabel("Magnitude")

xlim([-1.5*fn 1.5*fn])

ylim([0 max(abs(MODULATED_SIGNAL))])

bandwidth2 = 2*bandwidth;
```

```
disp("bandwidth of " + Audios(choose_channel) + " Modulated = " + bandwidth2 + " Hz");
```

FDM Signal Generation

```
FDM_Signal = 0; % Frequency Division Multiplixing


for i = 1 : length(Audios)

% Read the audio file and obtain the audio data

[audio_signal, Fs] = audioread(Audios(i)); % Read audio and get sampling frequency


audio_signal = (1/16)*interp(audio_signal, 16);


fo = 100000;

n = (i-1);

delta_f = 50000;

fn = fo + n*delta_f;

audio_length = (1:1:length(audio_signal))';

carrier_signal = cos(2*pi*fn*audio_length*(1/(16*Fs)));% [7] carriar signal

modulated_signal = carrier_signal.*audio_signal;

FDM_Signal = FDM_Signal + modulated_signal; %% summing point

end

FDM_SIGNAL = fftshift(fft(FDM_Signal));

Frequency_vector = (-length(FDM_SIGNAL)/2:length(FDM_SIGNAL)/2-1)';

plot(Frequency_vector*(16*Fs)/length(FDM_SIGNAL),(abs(FDM_SIGNAL)))

title("FDM Signal")

xlabel("Frequency (Hz)")

ylabel("Magnitude")
```

The RF stage

```
fprintf("Choose:\n0. Normal operation\n1. Remove RF Filter\n2. 0.1kHz Offset\n3. 1kHz
Offset\n");


test = input("Choose: ");

if test == 0 || 2 || 3

width = 1.2*bandwidth2; %width of the RF filter should be bigger than our signal
Bandwidth

A_stop1 = 60; % Attenuation in the first stopband = 60 dB
```

```
F_stop1 = (choose_channel-1)*50000+100000-27900; % Edge of the first stopband = (fn-k) [We choosed maxiumum k that filter can use before interacting with another signal]

F_pass1 = (choose_channel-1)*50000+100000-0.5*width; % Edge of the first passband = fn-0.5*width (decreasing F_pass1 to be closer to F_stop1 will increase filter order and make operation slower but give better result & this is not practical)

F_pass2 = (choose_channel-1)*50000+100000+0.5*width; % Edge of the second passband = fn+0.5*width

F_stop2 = (choose_channel-1)*50000+100000+27900; % Edge of the second stopband = (fn+k) making it symmetric is better

A_stop2 = 60; % Attenuation in the second stopband = 60 dB

A_pass = 1; % Amount of ripple allowed in the passband = 1 dB


RF_Filter = fdesign.bandpass(F_stop1, F_pass1, F_pass2, F_stop2, A_stop1, A_pass, A_stop2, (16*Fs));

RF_Filter = design(RF_Filter, 'equiripple'); % equiripple is good in dealing with ripples espicially in bandpass filter

RF_Signal = filter(RF_Filter, FDM_Signal);

RF_SIGNAL = fftshift(fft(RF_Signal));

Frequency_vector = (-length(RF_SIGNAL)/2:1:length(RF_SIGNAL)/2-1)';


plot(Frequency_vector*(16*Fs)/length(RF_SIGNAL), abs(RF_SIGNAL))

title("After RF Stage")

xlabel("Frequency (Hz)")

ylabel("Magnitude")

end
```

Mixer(Oscillator (fc + IF))

```
fn = (choose_channel-1)*50000+100000; % [100, 150, 200, 250, 300] KHz

IF = 25000; % If frequency 25 KHz

fLo = fn + IF; %Local oscillator carrier frequency

if test == 1

RF_Signal = FDM_Signal;


elseif test == 2

fLo = fLo + 100; % 0.1k offset


elseif test == 3

fLo = fLo + 1000; % 1k offset
```

```
else
fLo = fn + IF;
end

audio_length = (1:1:length(RF_Signal))';

carrier_signal = cos(2*pi*fLo*audio_length*(1/(16*Fs)));

Mixer_Output_Signal = RF_Signal.*carrier_signal;

MIXER_OUTPUT_SIGNAL = fftshift(fft(Mixer_Output_Signal));

Frequency_vector = (-length(MIXER_OUTPUT_SIGNAL)/2:1:length(MIXER_OUTPUT_SIGNAL)/2-1)';

plot(Frequency_vector*(16*Fs)/length(MIXER_OUTPUT_SIGNAL),abs(MIXER_OUTPUT_SIGNAL))
title(" After Mixer ")
xlabel("Frequency (Hz)")
ylabel("Magnitude")
```

IF stage

```
width = bandwidth2; % the same as signal bandwidth because of its high selectivity
A_stop1 = 60; % Attenuation in the first stopband = 60 dB
F_stop1 = IF-0.5*width-2000; % Edge of the first stopband (2000 is used due to
available frequency range limitaion)
F_pass1 = IF-0.5*width; % Edge of the first passband
F_pass2 = IF+0.5*width; % Edge of the second passband
F_stop2 = IF+0.5*width+2000; % Edge of the second stopband
A_stop2 = 60; % Attenuation in the second stopband = 60 dB
A_pass = 1; % Amount of ripple allowed in the passband = 1 dB

IF_Filter = fdesign.bandpass(F_stop1, F_pass1, F_pass2, F_stop2, A_stop1, A_pass,
A_stop2, (16*Fs));
IF_Filter = design(IF_Filter, 'equiripple');
IF_Signal = filter(IF_Filter, Mixer_Output_Signal);
IF_Signal = 1.5*IF_Signal; % amplification


IF_SIGNAL = fftshift(fft(IF_Signal));
Frequency_vector = (-length(IF_SIGNAL)/2:1:length(IF_SIGNAL)/2-1)';


plot(Frequency_vector*(16*Fs)/length(IF_SIGNAL), abs(IF_SIGNAL))
title("After IF Stage")
xlabel("Frequency (Hz)")
ylabel("Magnitude")
```

Base Band Stage (Demodulation)

```
fc = IF; % BaseBand carrier frequency


audio_length = (1:1:length(IF_Signal))';
carrier_signal = cos(2*pi*fc*audio_length*(1/(16*Fs)));


Base_Band_Signal = IF_Signal.*carrier_signal;
BASE_BAND_SIGNAL = fftshift(fft(Base_Band_Signal));
Frequency_vector = (-length(BASE_BAND_SIGNAL)/2:1:length(BASE_BAND_SIGNAL)/2-1)';


plot(Frequency_vector*(16*Fs)/length(BASE_BAND_SIGNAL),abs(BASE_BAND_SIGNAL))
title("Demodulator of Base Band Stage")
xlabel("Frequency (Hz)")
ylabel("Magnitude")
```

The Base Band detection (LPF)

```
width = bandwidth2;
F_stop = width; % Edge of the stopband ( we have good available range of frequencies to increase F_stop )
F_pass = width/2; % Edge of the passband (original bandwidth of the signal)
A_stop = 60; % Attenuation in the second stopband = 60 dB
A_pass = 1; % Amount of ripple allowed in the passband = 1 dB


Base_Band_Filter= fdesign.lowpass(F_pass, F_stop, A_pass, A_stop, (16*Fs));
Base_Band_Filter = design(Base_Band_Filter, 'butter'); % provide a maximally flat frequency response in the passband, which means they attenuate frequencies uniformly across the passband without any ripples
```

```
Base_Band_Signal = filter(Base_Band_Filter, Base_Band_Signal);

BASE_BAND_SIGNAL = fftshift(fft(Base_Band_Signal));

Frequency_vector = (-length(BASE_BAND_SIGNAL)/2:1:length(BASE_BAND_SIGNAL)/2-1)';


plot(Frequency_vector*(16*Fs)/length(BASE_BAND_SIGNAL), abs(BASE_BAND_SIGNAL))

title("After Base Band Stage")

xlabel("Frequency (Hz)")

ylabel("Magnitude")



Base_Band_Signal = 4*16*resample(Base_Band_Signal, 1, 16); %% restore main Fs & length
by doing the opposite the interp do & multiplied by 16 because it divides magnitude by
16 and we multiplied by another 4 due to two modulation processses

BASE_BAND_SIGNAL = fftshift(fft(Base_Band_Signal));

Frequency_vector = (-length(BASE_BAND_SIGNAL)/2:1:length(BASE_BAND_SIGNAL)/2-1)';

figure

plot((Frequency_vector*Fs/length(BASE_BAND_SIGNAL)), abs(BASE_BAND_SIGNAL))

title("Final result")

xlabel("Freq (Hz)")

ylabel("Magnitude")

ylim([0 max(abs(BASE_BAND_SIGNAL))])

sound(Base_Band_Signal, Fs) % to listen to our station


% --------------------testing different actions to sound----------------------------


errors = ["Removed_RF.wav", "0.1 KHz Offset.wav", "1 KHz Offset.wav"];% three tests

if test == 1||2||3

audiowrite(errors{test}, Base_Band_Signal, Fs); % save test audios, Test{1} Test{2} Test{3}

end
```