EECE26

Communication Project Super-Heterodyne Receiver

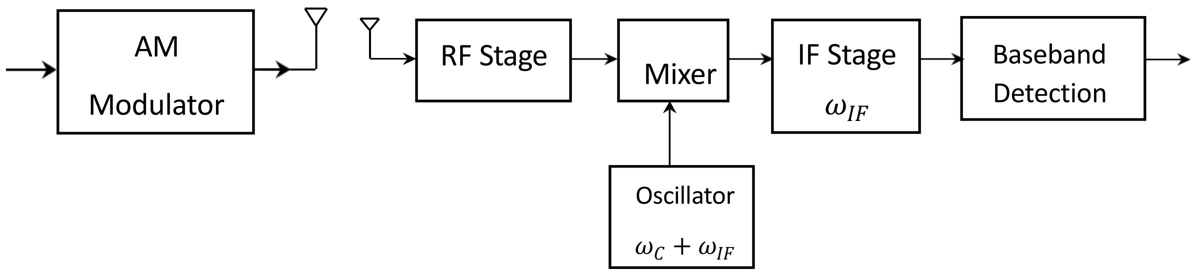
Cairo University Faculty of Engineering

Youssef Samir Zidan
9220988
Sec 4

1 Contents

1. Discussion

a. Discuss how you designed each of the blocks of the system in Figure 1.



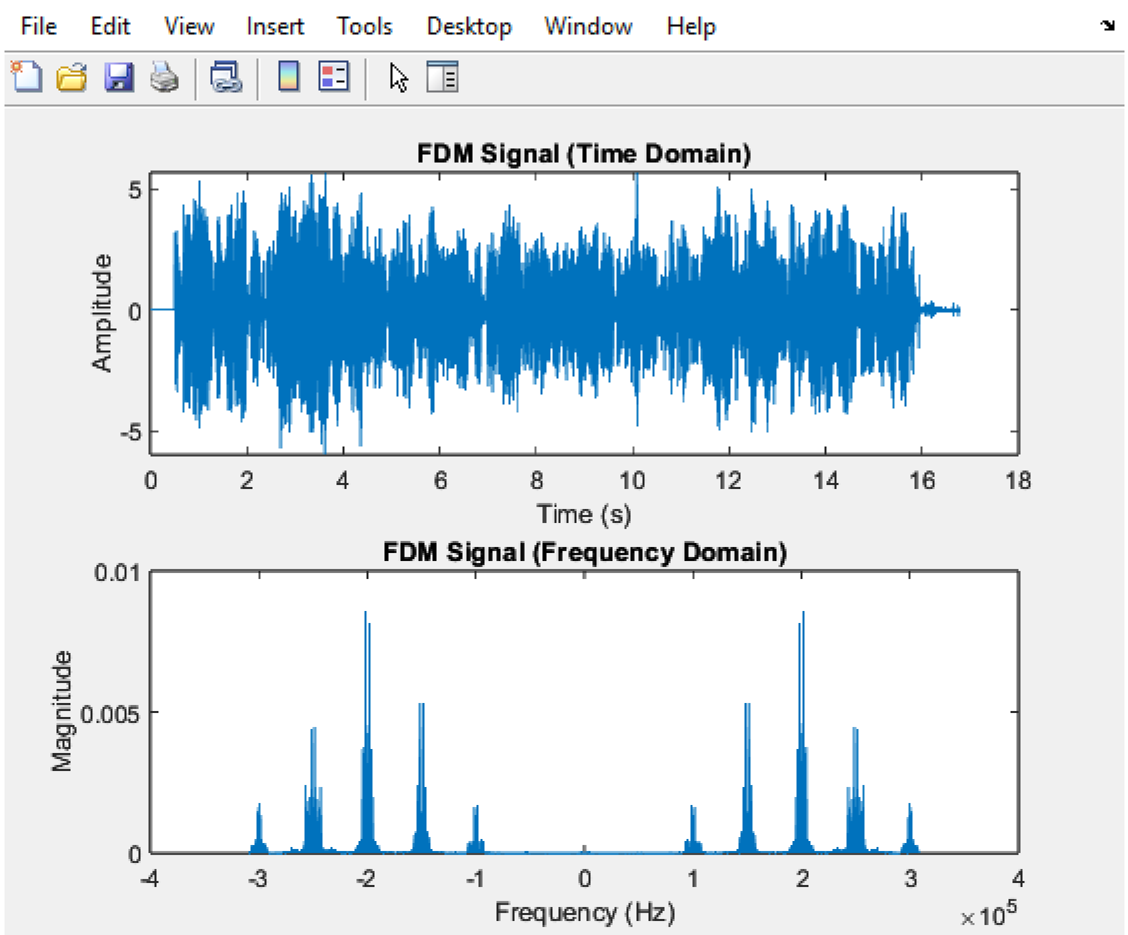
- The first stage is that I extracted the data from the signals getting important parameters like sampling frequency and BWs
- Then it came time for modulating which was as straight forward as just multiplying the signal by .*cos(2pi*Fc*t) in an element wise multiplication
- 

Fig. 1-1 TIme and Freq domain of all sent signals (modulated)

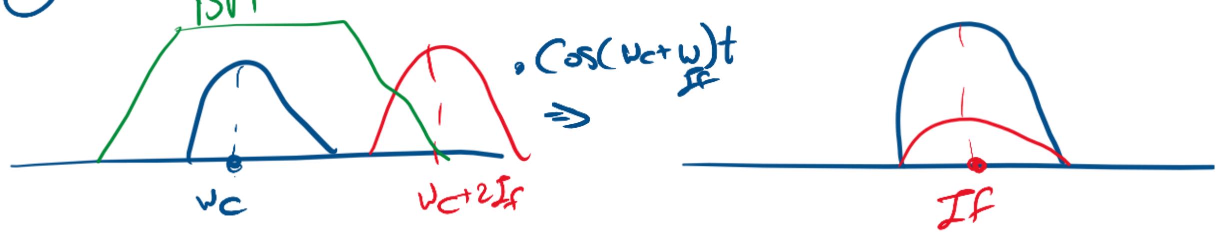
- Then the another function takes the whole receiver part
- At RF stage the user selects the required signal then a BPF is placed at the intended frequency to minimize the image interference
- Then a mixer simply multiplies with. *Cos(2pi*(Fc+IF) *t) to get the band in intermediate frequency which could be an input directly to an ADC (but more BW overhead for ADC)
- Then we put another BPF filter in IF stage to get rid of any interference and other signals
- For the last stage we modulate again with IF only to get the signal to the baseband, then finally we put a LPF to get rid of all noise and interference and hear the signal clearly

$$m(t)\cos(w_c t)\cdot\cos(w_c+w_{If})t$$

$$=\frac{1}{2}m(t)\left[\cos(2w_c+w_{If})t+\cos(I_f)t\right]$$

very far Not a concern

①



BPF specs → can't be very sharp

$w_c+B_c<F_c<W_c+2I_f-B_{img}$

& How much it rejects the image is determined by <u>SIR</u>

b. Answer the following:

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 detector selects the desired station from multiple signals when input signal is at Wc
  IF detector converts any signal to a fixed intermediate frequency regardless of the original carrier frequency. which simplifies the hardware and improves filtering performance
- Baseband detector is used to demodulate the signal from IF to the original Baseband

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. Hint: use the 'fft' command in MATLAB. You may also find the 'fftshift' command useful.

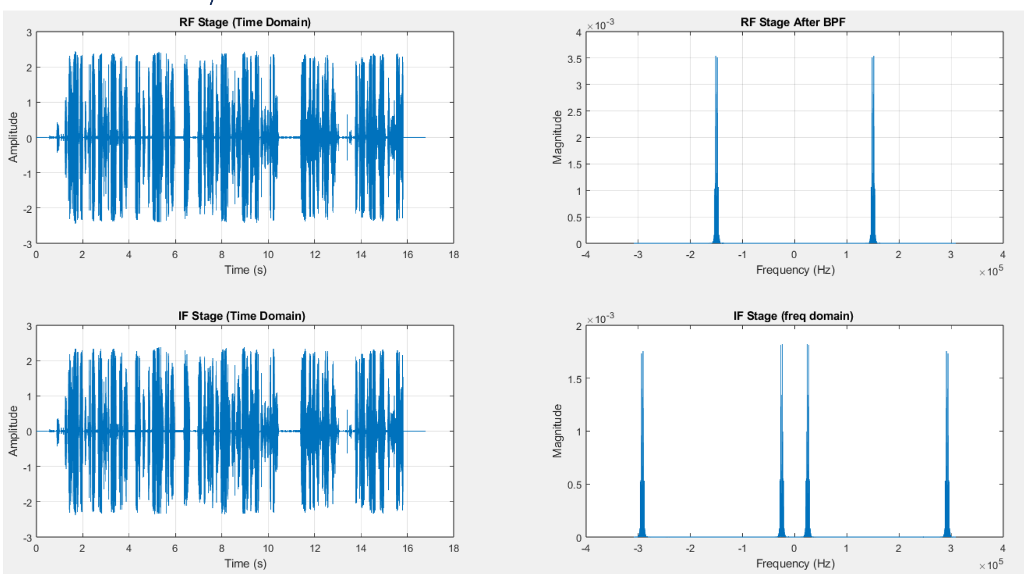


Fig. 1-2 RF and IF Stages time and freq domain signal

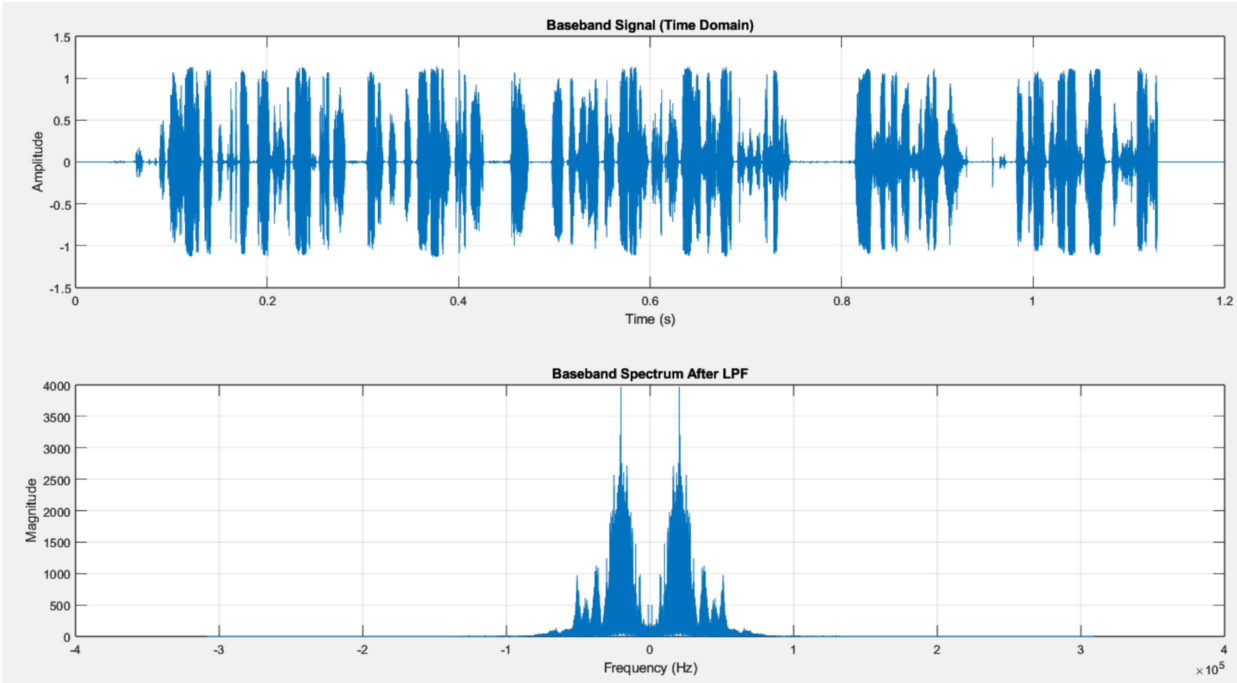


Fig. 1-3 baseband signal time and freq

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

- Yes, I can hear it clearly and I had to down sample again to hear it

4) Add “noise” to your signal and then play “sound” the signal. What is the effect of the noise?

- Using RECEIVED SIGNAL = AWGN(SENTSIGNAL,10);
- I heard a white noise with the signal وشوشة

5) 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 $\omega 0$?

- I only heard the image signal at 150khz completely overlapping and shadowing the intended signal
- And it’s visible in graph 2 IF stage that most of the interfering signals are still there
- I also would like to mention the I didn’t hear any image noise from signal no.5 as there’s no signal at “2*IF” from it

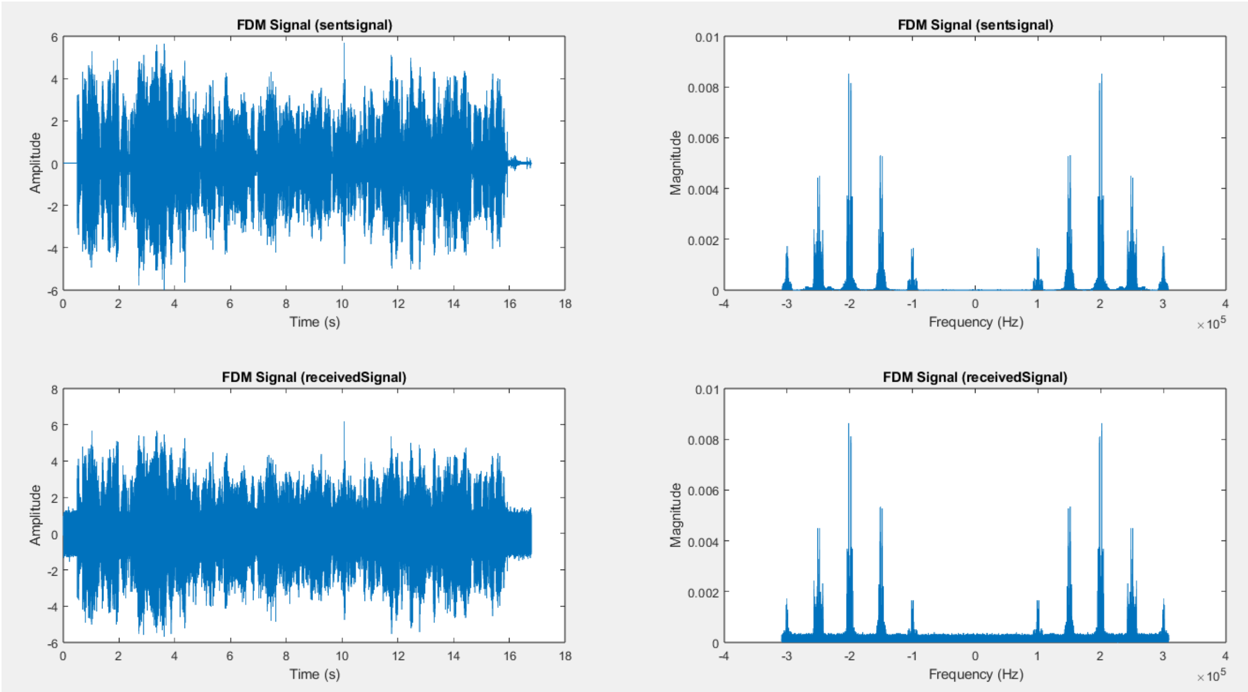


Fig. 1-4 after adding noise to received signal

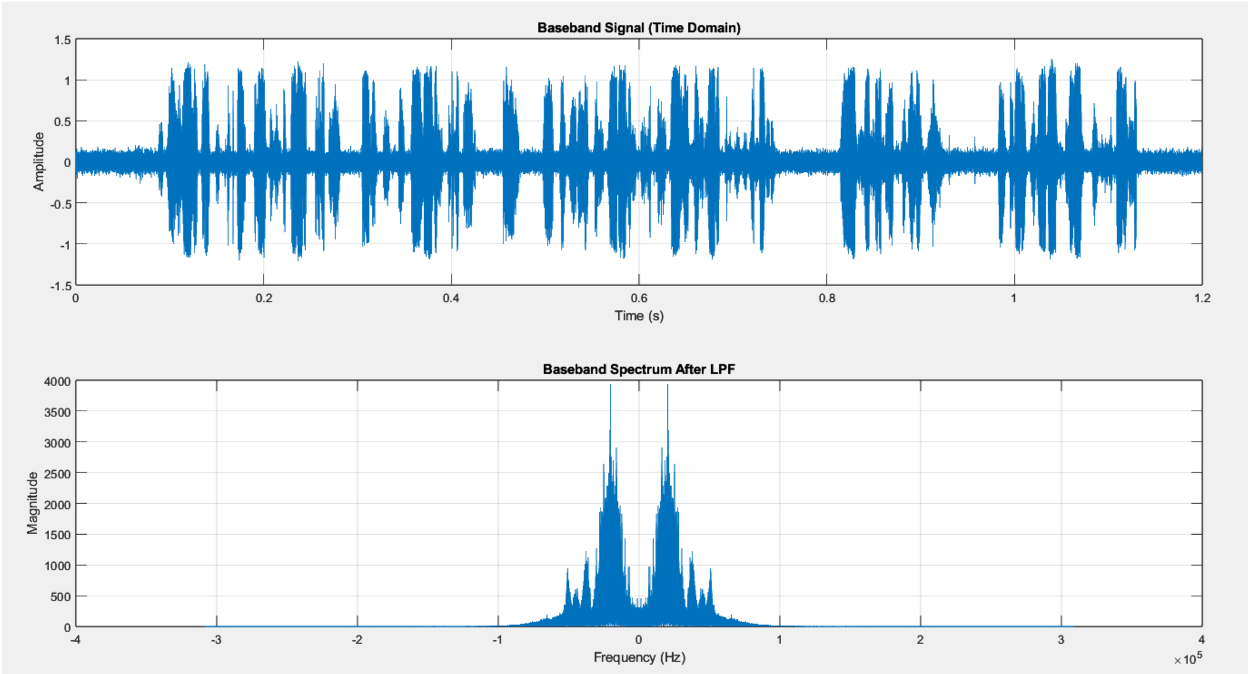


Fig. 1-5 baseband after adding noise

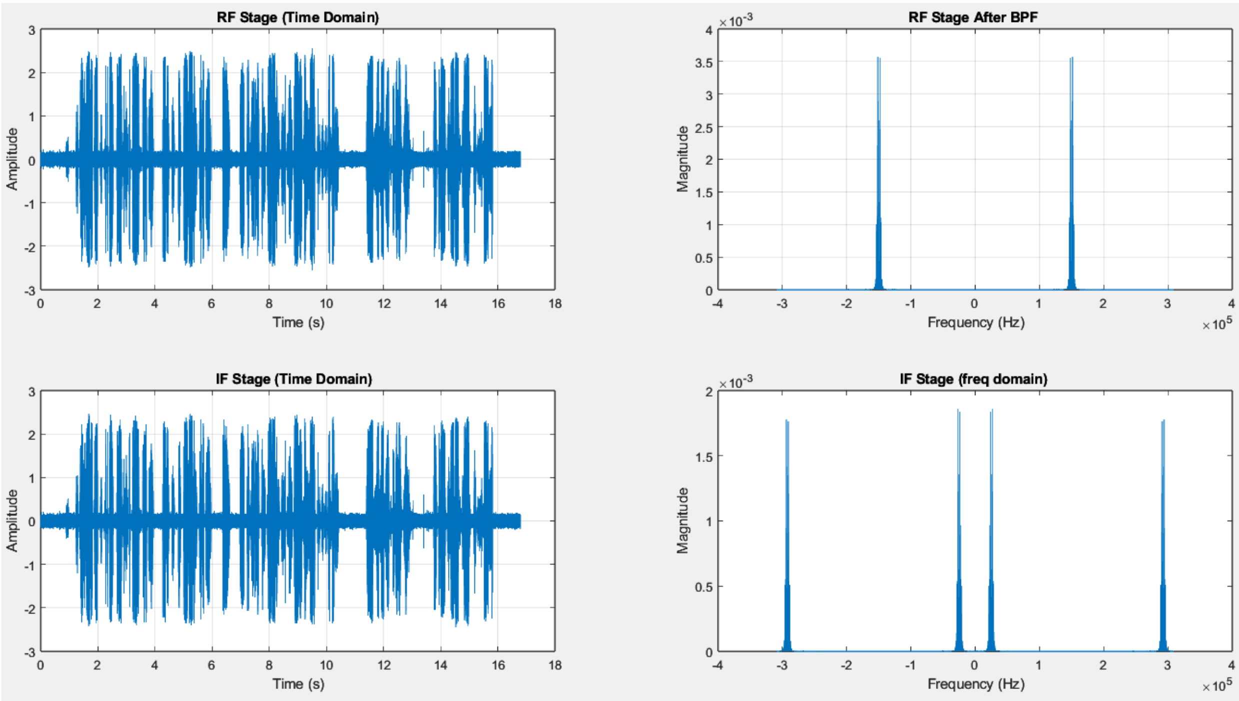


Fig. 1-6 RF And IF Stages after noise added

6) What happens (in terms of the spectrum and the sound quality) if the receiver oscillator has frequency offset by 0.2 KHz and 1.2 KHz?

- `carrier_LO = cos (2 * pi * (fc_mod + IF + offset) * t); % Local oscillator signal`
- `at 0.2khz I could barely hear the signal with very annoying sound effect`

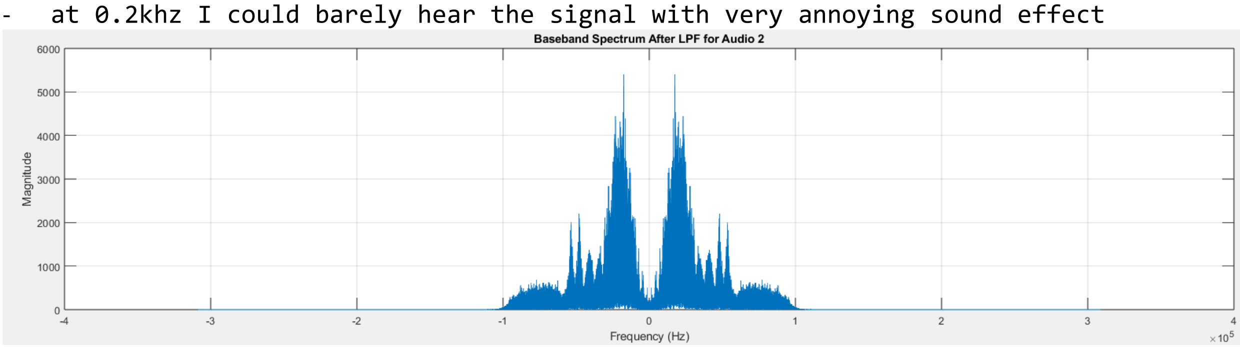


Fig. 1-7 effect of offset on baseband signal (0.2kHz)

- at 1.2khz I couldn’t hear the signal at all

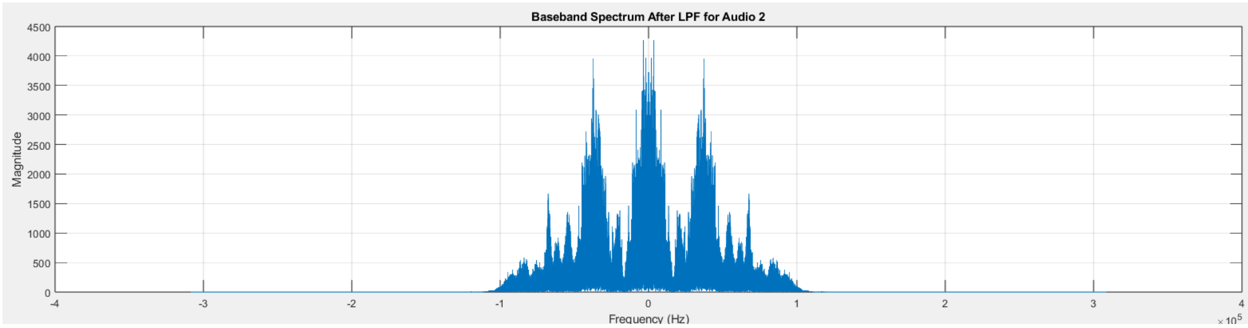


Fig. 1-8 effect of offset on baseband signal (1.2kHz)

1.1 Filter order

Command Window

```
>>
>> lpf_order= filtord(lowpass_filter);
fprintf('The order of the LPF is %d\n', lpf_order);
The order of the LPF is 8
>> filter_order_IF = filtord(if_bandpass_filter);
fprintf('The order of the IF bandpass filter is %d\n', filter_order_IF);
The order of the IF bandpass filter is 20
>> filter_order = filtord(rf_bandpass_filter);
fprintf('The order of the RF bandpass filter is %d\n', filter_order);
The order of the RF bandpass filter is 26
>>
```

2 CODE appendix

2.1 Samples (First to run)

```
%% Section 1: Load Data
addpath 'Audio Signals'\
[c1, fs1] = audioread("Short_BBCArabic2.wav");
[c2, fs2] = audioread("Short_FM9090.wav");
[c3, fs3] = audioread("Short_QuranPalestine.wav");
[c4, fs4] = audioread("Short_RussianVoice.wav");
[c5, fs5] = audioread("Short_SkyNewsArabia.wav");
audioSignals = {c1, c2, c3, c4, c5};
samplingRates = [fs1, fs2, fs3, fs4, fs5];
Left_channel = cell(1, length(audioSignals));
Right_channel = cell(1, length(audioSignals));
monoSignals = cell(1, 5);
paddedSignals = cell(1, 5);
maxLength = 0;
% Extract left, right, and mono channels; calculate max signal length
for i = 1:length(audioSignals)
    Left_channel{i} = audioSignals{i}(:, 1);
    Right_channel{i} = audioSignals{i}(:, 2);
    monoSignals{i} = sum(audioSignals{i}, 2);  % Combine left and right channels
    maxLength = max(maxLength, length(monoSignals{i}));
end
% Pad signals to match the maximum length
for i = 1:length(monoSignals)
    paddedSignals{i} = [monoSignals{i}; zeros(maxLength - length(monoSignals{i}), 1)];
end
%% Section 2: Plot and Extract Data
BWs = zeros(1, 5);  % Initialize array for bandwidths
% Plot frequency spectra and calculate bandwidths
for i = 1:length(audioSignals)
    x = audioSignals{i};
    fs = samplingRates(i);
    % FFT for frequency spectrum
    N = length(x);
```

```
    X = fft(x, N);

    f = (-N/2:N/2-1) * fs / N;

    % Calculate bandwidth

    BWs(i) = obw(monoSignals{i}, fs);

    % Plot frequency spectrum

    figure;

    plot(f, abs(fftshift(X)) / N);

    title(['Frequency Spectrum of Signal ', num2str(i)]);

    xlabel('Frequency (Hz)');

    ylabel('Magnitude');

    grid on;

end

% Convert bandwidths to Hz and kHz

BWs_Hz = BWs;

BWs_kHz = BWs_Hz / 1000;

clearvars -except BWs paddedSignals audioSignals
```

2.2 Modulation (second to run)

```
% Constants
fc_mod = zeros(1, 5);  % Carrier frequencies
modulated_signal = cell(1, 5);  % Modulated signals
sentsignal = [];  % Combined FDM signal
interp_factor = 14;  % Interpolation factor
fs = 44100;  % Sampling frequency
fs_interpolated = interp_factor * fs;  % Interpolated sampling frequency

% Modulation process
for i = 1:length(audioSignals)
    % Retrieve the padded mono signal
    x = paddedSignals{i};

    % Take the first portion of the signal
    x = x(1:floor(length(x) / 1));

    % Carrier frequency (in Hz)
    fc_mod(i) = 100e3 + (i - 1) * 50e3;

    % Interpolate the signal
    x_inter = interp(x, interp_factor);

    % Time vector for the interpolated signal
    t = (0:1/fs_interpolated:(length(x_inter) - 1) / fs_interpolated)';

    % Generate the carrier signal and perform modulation
    Mod_carrier = cos(2 * pi * fc_mod(i) * t);
    modulated_signal{i} = x_inter .* Mod_carrier;
end

% Combine all modulated signals into the FDM signal
for i = 1:length(modulated_signal)
    if isempty(sentsignal)
        sentsignal = modulated_signal{i};
    else
```

```
        sentsignal = sentsignal + modulated_signal{i};
    end
end

% Plot the combined FDM signal in the time domain
figure;
subplot(2, 1, 1);
t_total = (0:length(sentsignal) - 1) / fs_interpolated;  % Time vector for FDM signal
plot(t_total, sentsignal);
title('FDM Signal (Time Domain)');
xlabel('Time (s)');
ylabel('Amplitude');

% Plot the combined FDM signal in the frequency domain
subplot(2, 1, 2);
N = length(sentsignal);
f = (-N/2:N/2-1) * fs_interpolated / N;  % Frequency axis
spectrum_fdm = fftshift(fft(sentsignal, N));  % Compute and shift the FFT
plot(f, abs(spectrum_fdm) / N);
title('FDM Signal (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Play the FDM signal
%sound(sentsignal, fs_interpolated);
clearvars -except BWs   sentsignal interp_factor fs
```

2.3 Demodulation

```
% Constants
offset = 0;
IF = 25e3;  % Intermediate frequency (25 kHz)
fs_interpolated = interp_factor * fs;  % Interpolated sampling rate
receivedSignal = sentsignal;  % Received FDM signal
Rf_enable = 0;  % Enable/Disable RF filter

% Relaxed Filter Constraints
A_pass = 2;
A_stop1 = 20;
A_stop2 = 20;

fprintf("\nSignal Processing Initialized.\n");
while true
    % User input for channel selection
    fprintf("\nChoose one of these channels (Enter 0 to Exit):\n");
    fprintf("1. Short_BBCArabic2\n");
    fprintf("2. Short_FM9090\n");
    fprintf("3. Short_QuranPalestine\n");
    fprintf("4. Short_RussianVoice\n");
    fprintf("5. Short_SkyNewsArabia\n");
    i = input("Choose: ");

    % Exit condition
    if i == 0
        fprintf("Exiting program...\n");
        break;
    elseif i < 1 || i > 5
        fprintf("Invalid input! Please choose a number between 1 and 5, or 0 to exit.\n");
        continue;
    end

    % Carrier frequency and bandwidth for the selected channel
    fc_mod = 100e3 + (i - 1) * 50e3;  % Carrier frequency for the selected channel
    bw = BWs(i);  % Bandwidth (fixed for all channels)

    % Define proper increasing frequencies and cap at Nyquist
    F_stop1 =  fc_mod - bw ;  % Stopband start
    F_pass1 = fc_mod - bw/2;                % Passband start
```

```
    F_pass2 = fc_mod + bw/2;                % Passband end
    F_stop2 =  fc_mod + bw ;  % Stopband end

    % RF Bandpass Filter
    if Rf_enable == 1
        if F_stop2 > fs_interpolated / 2
            error("Filter frequencies exceed Nyquist limit. Increase fs_interpolated.");
        end
        rf_bandpass_filter = designfilt('bandpassiir', ...
            'StopbandFrequency1', F_stop1, 'PassbandFrequency1', F_pass1, ...
            'PassbandFrequency2', F_pass2, 'StopbandFrequency2', F_stop2, ...
            'StopbandAttenuation1', A_stop1, 'PassbandRipple', A_pass, ...
            'StopbandAttenuation2', A_stop2, 'SampleRate', fs_interpolated);
        fprintf("RF Filter Order: %d\n", filtord(rf_bandpass_filter));
        filtered_signal = filter(rf_bandpass_filter, receivedSignal);
    else
        filtered_signal = receivedSignal;
    end

    % Plot RF Stage Spectrum
    figure(1); clf;
    rf_spectrum = fftshift(fft(filtered_signal, 2^nextpow2(length(filtered_signal))));
    N = length(rf_spectrum);
    f = (-N/2:N/2-1) * fs_interpolated / N;
    subplot(2, 1, 1);
    plot(f, abs(rf_spectrum));
    title("RF Stage After BPF for Audio " + num2str(i));
    xlabel("Frequency (Hz)"); ylabel("Magnitude"); grid on;

    % Step 2: Mix with Local Oscillator to Shift to IF
    t = (1:length(filtered_signal))' / fs_interpolated;  % Time vector
    carrier_LO = cos(2 * pi * (fc_mod + IF) * t);  % Local oscillator signal
    mixed_signal_IF = filtered_signal .* carrier_LO;  % Mixer output

    % Plot IF Stage Spectrum
    IF_spectrum = fftshift(fft(mixed_signal_IF, 2^nextpow2(length(mixed_signal_IF))));
    subplot(2, 1, 2);
    plot(f, abs(IF_spectrum));
    title("IF Stage for Audio " + num2str(i));
    xlabel("Frequency (Hz)"); ylabel("Magnitude"); grid on;

    % Step 3: Bandpass Filter to Isolate IF
    F_stop1_IF = max(0, IF - bw - 5e3);  % Stopband start
    F_pass1_IF = IF - bw;                % Passband start
    F_pass2_IF = IF + bw;                % Passband end
    F_stop2_IF = min(fs_interpolated / 2 - 1, IF + bw + 5e3);  % Stopband end

    if_bandpass_filter = designfilt('bandpassiir', ...
        'StopbandFrequency1', F_stop1_IF, 'PassbandFrequency1', F_pass1_IF, ...
        'PassbandFrequency2', F_pass2_IF, 'StopbandFrequency2', F_stop2_IF, ...
        'StopbandAttenuation1', A_stop1, 'PassbandRipple', A_pass, ...
        'StopbandAttenuation2', A_stop2, 'SampleRate', fs_interpolated);
    fprintf("IF Filter Order: %d\n", filtord(if_bandpass_filter));
    filtered_signal_IF = filter(if_bandpass_filter, mixed_signal_IF);

    % Step 4: Demodulate IF Signal
    carrier_IF = cos(2 * pi * IF * t);  % IF carrier signal
    baseband_signal = filtered_signal_IF .* carrier_IF;  % Demodulation

    % Step 5: Apply Low-Pass Filter to Recover Baseband Signal
    F_pass = bw; F_stop = bw + 5e3;
    lowpass_filter = designfilt('lowpassiir', ...
        'PassbandFrequency', F_pass, 'StopbandFrequency', F_stop, ...
        'PassbandRipple', A_pass, 'StopbandAttenuation', A_stop1, ...
        'SampleRate', fs_interpolated);
    Base_Band_received_signal_LPF = filter(lowpass_filter, baseband_signal);

    % Resample and Playback the Demodulated Signal
    Base_Band_received_signal_LPF = 4 * resample(Base_Band_received_signal_LPF, 1,
interp_factor);
    sound(Base_Band_received_signal_LPF, fs);
```

```
    % Plot Baseband Spectrum After LPF
    figure(2); clf;
    baseband_spectrum = fftshift(fft(Base_Band_received_signal_LPF,
2^nextpow2(length(Base_Band_received_signal_LPF))));
    N = length(baseband_spectrum);
    f = (-N/2:N/2-1) * fs_interpolated / N;
    plot(f, abs(baseband_spectrum));
    title("Baseband Spectrum After LPF for Audio " + num2str(i));
    xlabel("Frequency (Hz)"); ylabel("Magnitude"); grid on;

    fprintf("Audio %d processed successfully.\n", i);
end
fprintf("Signal Processing Completed.\n");
```

2.4 Whole code

```
%% Section 1: Load Data
addpath 'Audio Signals'\
[c1, fs1] = audioread("Short_BBCArabic2.wav");
[c2, fs2] = audioread("Short_FM9090.wav");
[c3, fs3] = audioread("Short_QuranPalestine.wav");
[c4, fs4] = audioread("Short_RussianVoice.wav");
[c5, fs5] = audioread("Short_SkyNewsArabia.wav");

audioSignals = {c1, c2, c3, c4, c5};
samplingRates = [fs1, fs2, fs3, fs4, fs5];
Left_channel = cell(1, length(audioSignals));
Right_channel = cell(1, length(audioSignals));
monoSignals = cell(1, 5);
paddedSignals = cell(1, 5);
maxLength = 0;

% Extract left, right, and mono channels; calculate max signal length
for i = 1:length(audioSignals)
    Left_channel{i} = audioSignals{i}(:, 1);
    Right_channel{i} = audioSignals{i}(:, 2);
    monoSignals{i} = sum(audioSignals{i}, 2);  % Combine left and right channels
    maxLength = max(maxLength, length(monoSignals{i}));
end

% Pad signals to match the maximum length
for i = 1:length(monoSignals)
    paddedSignals{i} = [monoSignals{i}; zeros(maxLength - length(monoSignals{i}), 1)];
end
% Section 2: Plot and Extract Data
BWs = zeros(1, 5);  % Initialize array for bandwidths
% Plot frequency spectra and calculate bandwidths
for i = 1:length(audioSignals)
    x = audioSignals{i};
    fs = samplingRates(i);

    % FFT for frequency spectrum
    N = length(x);
    X = fft(x, N);
    f = (-N/2:N/2-1) * fs / N;

    % Calculate bandwidth
    BWs(i) = obw(monoSignals{i}, fs);
end

% Convert bandwidths to Hz and kHz
BWs_Hz = BWs;
BWs_kHz = BWs_Hz / 1000;
clearvars -except BWs paddedSignals audioSignals samplingRates

%%  section2: Modulation
```

```
% Constants
fc_mod = zeros(1, 5);  % Carrier frequencies
modulated_signal = cell(1, 5);  % Modulated signals
sentsignal = [];  % Combined FDM signal
interp_factor = 14;  % Interpolation factor
fs = 44100;  % Sampling frequency
fs_interpolated = interp_factor * fs;  % Interpolated sampling frequency

% Modulation process
for i = 1:length(audioSignals)
    % Retrieve the padded mono signal
    x = paddedSignals{i};

    % Take the first portion of the signal
    x = x(1:floor(length(x) / 1));

    % Carrier frequency (in Hz)
    fc_mod(i) = 100e3 + (i - 1) * 50e3;

    % Interpolate the signal
    x_inter = interp(x, interp_factor);

    % Time vector for the interpolated signal
    t = (0:1/fs_interpolated:(length(x_inter) - 1) / fs_interpolated)';

    % Generate the carrier signal and perform modulation
    Mod_carrier = cos(2 * pi * fc_mod(i) * t);
    modulated_signal{i} = x_inter .* Mod_carrier;
end

% Combine all modulated signals into the FDM signal
for i = 1:length(modulated_signal)
    if isempty(sentsignal)
        sentsignal = modulated_signal{i};
    else
        sentsignal = sentsignal + modulated_signal{i};
    end
end

% Subplot 1: Time domain of sentsignal
subplot(2, 2, 1);
t_total = (0:length(sentsignal) - 1) / fs_interpolated;
plot(t_total, sentsignal);
title('FDM Signal (sentsignal)');
xlabel('Time (s)');
ylabel('Amplitude');

% Subplot 2: Frequency domain of sentsignal
subplot(2, 2, 2);
N = length(sentsignal);
f = (-N/2:N/2-1) * fs_interpolated / N;
spectrum_fdm = fftshift(fft(sentsignal, N));
plot(f, abs(spectrum_fdm) / N);
title('FDM Signal (sentsignal)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Play the FDM signal
%sound(sentsignal, fs_interpolated);
clearvars -except BWs   sentsignal interp_factor fs audioSignals samplingRates

%% Section3: Receiver
% Constants
offset = 0;
IF = 25e3;  % Intermediate frequency (25 kHz)
fs_interpolated = interp_factor * fs;  % Interpolated sampling rate
receivedSignal = sentsignal;  % Received FDM signal
%receivedSignal = awgn(sentsignal,10);  % Received FDM signal

% Subplot 3: Time domain of receivedSignal
subplot(2, 2, 3);
```

```
t_total = (0:length(receivedSignal) - 1) / fs_interpolated;
plot(t_total, receivedSignal);
title('FDM Signal (receivedSignal)');
xlabel('Time (s)');
ylabel('Amplitude');

% Subplot 4: Frequency domain of receivedSignal
subplot(2, 2, 4);
N = length(receivedSignal);
f = (-N/2:N/2-1) * fs_interpolated / N;
spectrum_fdm = fftshift(fft(receivedSignal, N));
plot(f, abs(spectrum_fdm) / N);
title('FDM Signal (receivedSignal)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

Rf_enable = 1;  % Enable/Disable RF filter

% Relaxed Filter Constraints
A_pass = 2;
A_stop1 = 20;
A_stop2 = 20;

fprintf("\nSignal Processing Initialized.\n");
while true
    % User input for channel selection
    fprintf("\nChoose one of these channels (Enter 0 to Exit):\n");
    fprintf("1. Short_BBCArabic2\n");
    fprintf("2. Short_FM9090\n");
    fprintf("3. Short_QuranPalestine\n");
    fprintf("4. Short_RussianVoice\n");
    fprintf("5. Short_SkyNewsArabia\n");
    i = input("Choose: ");

    % Exit condition
    if i == 0
        fprintf("Exiting program...\n");
        break;
    elseif i < 1 || i > 5
        fprintf("Invalid input! Please choose a number between 1 and 5, or 0 to exit.\n");
        continue;
    end

    % Carrier frequency and bandwidth for the selected channel
    fc_mod = 100e3 + (i - 1) * 50e3;  % Carrier frequency for the selected channel
    bw = BWs(i);  % Bandwidth (fixed for all channels)

    % Define proper increasing frequencies and cap at Nyquist
    F_stop1 =  fc_mod - bw ;  % Stopband start
    F_pass1 = fc_mod - bw/2;              % Passband start
    F_pass2 = fc_mod + bw/2;              % Passband end
    F_stop2 =  fc_mod + bw ;  % Stopband end

    % RF Bandpass Filter
    if Rf_enable == 1
        if F_stop2 > fs_interpolated / 2
            error("Filter frequencies exceed Nyquist limit. Increase fs_interpolated.");
        end
        rf_bandpass_filter = designfilt('bandpassiir', ...
            'StopbandFrequency1', F_stop1, 'PassbandFrequency1', F_pass1, ...
            'PassbandFrequency2', F_pass2, 'StopbandFrequency2', F_stop2, ...
            'StopbandAttenuation1', A_stop1, 'PassbandRipple', A_pass, ...
            'StopbandAttenuation2', A_stop2, 'SampleRate', fs_interpolated);
        fprintf("RF Filter Order: %d\n", filtord(rf_bandpass_filter));
        filtered_signal = filter(rf_bandpass_filter, receivedSignal);
    else
        filtered_signal = receivedSignal;
    end

% Plot RF Stage Spectrum and Time Domain in a New Figure
figure;
```

```
% Subplot 1: Time Domain
subplot(2, 2, 1);
t_total = (0:length(filtered_signal) - 1) / fs_interpolated;
plot(t_total, filtered_signal);
title('RF Stage (Time Domain)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Subplot 2: Frequency Domain of filtered_signal (RF Spectrum)
rf_spectrum = fftshift(fft(filtered_signal, 2^nextpow2(length(filtered_signal))));  %
Shifted FFT
N = length(rf_spectrum);  % Length of the signal
f = (-N/2:N/2-1) * fs_interpolated / N;  % Correct frequency axis scaling
subplot(2, 2, 2);
plot(f, abs(rf_spectrum) / N);  % Normalize by the length of the signal
title('RF Stage After BPF ');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

    % Step 2: Mix with Local Oscillator to Shift to IF
    t = (1:length(filtered_signal))' / fs_interpolated;  % Time vector
    carrier_LO = cos(2 * pi * (fc_mod + IF) * t);  % Local oscillator signal
    mixed_signal_IF = filtered_signal .* carrier_LO;  % Mixer output

% Plot IF Stage Spectrum in a Different Figure with Its Own Subplot

% Subplot 1: Time Domain of mixed_signal_IF (Optional if you want to plot it)
subplot(2, 2, 3);
t_total_IF = (0:length(mixed_signal_IF) - 1) / fs_interpolated;
plot(t_total_IF, mixed_signal_IF);
title('IF Stage (Time Domain)  ');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Subplot 2: Frequency Domain of mixed_signal_IF (IF Spectrum)
IF_spectrum = fftshift(fft(mixed_signal_IF, 2^nextpow2(length(mixed_signal_IF))));  %
Shifted FFT
subplot(2, 2, 4);
plot(f, abs(IF_spectrum) / N);  % Normalize by the length of the signal
title("IF Stage (freq domain) " );
xlabel("Frequency (Hz)");
ylabel("Magnitude");
grid off;

    % Step 3: Bandpass Filter to Isolate IF
    F_stop1_IF = max(0, IF - bw - 5e3);  % Stopband start
    F_pass1_IF = IF - bw;                % Passband start
    F_pass2_IF = IF + bw;                % Passband end
    F_stop2_IF = min(fs_interpolated / 2 - 1, IF + bw + 5e3);  % Stopband end

    if_bandpass_filter = designfilt('bandpassiir', ...
        'StopbandFrequency1', F_stop1_IF, 'PassbandFrequency1', F_pass1_IF, ...
        'PassbandFrequency2', F_pass2_IF, 'StopbandFrequency2', F_stop2_IF, ...
        'StopbandAttenuation1', A_stop1, 'PassbandRipple', A_pass, ...
        'StopbandAttenuation2', A_stop2, 'SampleRate', fs_interpolated);
    fprintf("IF Filter Order: %d\n", filtord(if_bandpass_filter));
    filtered_signal_IF = filter(if_bandpass_filter, mixed_signal_IF);

    % Step 4: Demodulate IF Signal
    carrier_IF = cos(2 * pi * IF * t);  % IF carrier signal
    baseband_signal = filtered_signal_IF .* carrier_IF;  % Demodulation

    % Step 5: Apply Low-Pass Filter to Recover Baseband Signal
    F_pass = bw; F_stop = bw + 5e3;
    lowpass_filter = designfilt('lowpassiir', ...
        'PassbandFrequency', F_pass, 'StopbandFrequency', F_stop, ...
        'PassbandRipple', A_pass, 'StopbandAttenuation', A_stop1, ...
```

```
        'SampleRate', fs_interpolated);
    Base_Band_received_signal_LPF = filter(lowpass_filter, baseband_signal);

    % Resample and Playback the Demodulated Signal
    Base_Band_received_signal_LPF = 4 * resample(Base_Band_received_signal_LPF, 1,
interp_factor);
    sound(Base_Band_received_signal_LPF, fs);

% Create a new figure
figure;

% Subplot 1: Time Domain of Base_Band_received_signal_LPF
subplot(2, 1, 1);  % First position in a 2x2 grid
t_total = (0:length(Base_Band_received_signal_LPF) - 1) / fs_interpolated;
plot(t_total, Base_Band_received_signal_LPF);
title('Baseband Signal (Time Domain)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Subplot 3: Frequency Domain of Base_Band_received_signal_LPF (Baseband Spectrum)
baseband_spectrum = fftshift(fft(Base_Band_received_signal_LPF,
2^nextpow2(length(Base_Band_received_signal_LPF))));  % Shifted FFT
N = length(baseband_spectrum);  % Length of the signal
f = (-N/2:N/2-1) * fs_interpolated / N;  % Frequency axis scaling
subplot(2, 1, 2);  % Third position in a 2x2 grid
plot(f, abs(baseband_spectrum));
title('Baseband Spectrum After LPF');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

    fprintf("Audio %d processed successfully.\n", i);
end
fprintf("Signal Processing Completed.\n");
```