**PROJECT CODE:**

% Recording, Playing and Write Audio File

clc;close all;clear all;

warning off

recObj = audiorecorder;% audiorecorder creates an 8000 Hz, 8-bit, 1 channel audiorecorder object.

% audiorecorder(Fs, NBITS, NCHANS) creates an audiorecorder object with

% sample rate Fs in Hertz, number of bits NBITS, and number of channels NCHANS.

% Common sample rates are 8000, 11025, 22050, 44100, 48000, and 96000 Hz.

% The number of bits must be 8, 16, or 24. The number of channels must

% be 1 or 2 (mono or stereo).

% audiorecorder(Fs, NBITS, NCHANS, ID) creates an audiorecorder object using

% audio device identifier ID for input. If ID equals -1 the default input

% device will be used.

recObj2 = audiorecorder;

recObj3 = audiorecorder;

Fs = 10000 ; % Sampling frequency in hertz8000, 11025, 22050, 44100, 48000, and 96000 Hz.

nBits = 16 ;% 8, 16, or 24

nChannels = 1 ; %Number of channels--2 options--1 (mono) or 2 (stereo)

ID = -1; % default audio input device like Microphone

recObj = audiorecorder(Fs,nBits,nChannels,ID);

recObj2 = audiorecorder(Fs,nBits,nChannels,ID);

recObj3 = audiorecorder(Fs,nBits,nChannels,ID);

disp('Start speaking.')

recordblocking(recObj,5);

disp('End of Recording.');

%play(recObj);

mySpeech = getaudiodata(recObj); % returns the recorded audio data as a double array

%Write audio file

audiowrite('m1.wav',mySpeech,Fs);

pause(1)

disp('Start speaking.')

recordblocking(recObj2,5);

% recordblocking(OBJ, T) records for length of time, T, in seconds;

% does not return until recording is finished.

disp('End of Recording.');

%play(recObj2);

mySpeech = getaudiodata(recObj2); % returns the recorded audio data as a double array

% getaudiodata(OBJ, DATATYPE) returns the recorded audio data in

% the data type as requested in string DATATYPE. Valid data types

% are 'double', 'single', 'int16', 'uint8', and 'int8'.

%Write audio file

audiowrite('m2.wav',mySpeech,Fs);

pause(1)

disp('Start speaking.')

recordblocking(recObj3,5);

% recordblocking(OBJ, T) records for length of time, T, in seconds;

% does not return until recording is finished.

disp('End of Recording.');

%play(recObj3); %PLAYS AUDIO BACK TO YOU

mySpeech = getaudiodata(recObj3); % returns the recorded audio data as a double array

% getaudiodata(OBJ, DATATYPE) returns the recorded audio data in

% the data type as requested in string DATATYPE. Valid data types

% are 'double', 'single', 'int16', 'uint8', and 'int8'.

%Write audio file

audiowrite('m3.wav',mySpeech,Fs);

%modulation stage:

%%%% Reading and Plotting Audio Signal with Noise %%%%%%%%%%%%%%%%%%%%%%%%%

clc;clear all; close all

[signal\_orignal,Fs] = audioread('m1.wav');

samples=24000; %i used 24000 instead of 48K samples as my computer would crash due to stack overflow at 48K

%noisy\_signal = signal\_orignal(1:samples)+0.05\*randn(1,samples)';

signal1 = signal\_orignal(1:samples);

f = -Fs/2:Fs/samples:Fs/2-(Fs/samples);

[signal\_orignal2,Fs] = audioread('m2.wav');

%noisy\_signal = signal\_orignal(1:samples)+0.05\*randn(1,samples)';

signal2 = signal\_orignal2(1:samples);

[signal\_orignal3,Fs] = audioread('m3.wav');

%noisy\_signal = signal\_orignal(1:samples)+0.05\*randn(1,samples)';

signal3 = signal\_orignal(1:samples);

figure

subplot (2,2,1)

plot(f,abs(fftshift(fft(signal1)))) % FFT (not normalized)

title('Magnitude Spectrum of Signal 1 m1'), grid

sound(signal1,Fs)

subplot (2,2,2)

plot(f,abs(fftshift(fft(signal2)))) % FFT (not normalized)

title('Magnitude Spectrum of Signal 2 m2'), grid

sound(signal2,Fs)

subplot (2,2,3)

plot(f,abs(fftshift(fft(signal3)))) % FFT (not normalized)

title('Magnitude Spectrum of Signal 3 m3'), grid

sound(signal3,Fs)

%part 2

N = size(signal1,1); % Row Length Of ‘y’

Ts = 1/Fs; % Sampling Interval (seconds)

t = linspace(0, 1, N)'\*Ts; % Time Column Vector

%%%%%% Implementing Low Pass Filter %%%%%%%%%%%%%%

f\_cut = 3000; % LPF cutoff frequency in Hz

ord = 4; % LPF filter order

[b,a]=butter(ord,f\_cut/(Fs/2));

% [b,a]=butter(4,20/500);

env = filter(b,a,signal1);

env2 = filter(b,a,signal2);

env3 = filter(b,a,signal3);

y = env - mean(env)

y2 = env2 - mean(env2)

y3 = env3 - mean(env3)

Zk = fft(y)/N; % Computing fft for signal1

Z = fftshift(abs(Zk));

Zk2 = fft(y2)/N; % Computing fft for signal2

Z2 = fftshift(abs(Zk2));

Zk3 = fft(y3)/N; % Computing fft for signal3

Z3 = fftshift(abs(Zk3));

figure

subplot(241)

plot(t,y,'linewidth',1);

% ylim([-1 1])

title('TASK PART 2: signal 1 in time domain');

grid on;

subplot(242)

fz = -Fs/2:Fs/length(Z):Fs/2-(Fs/length(Z));

stem(fz,Z,'linewidth',1);

% xlim([-20 20])

title('TASK PART 2: signal 1 in frequency domain');

grid on;

subplot(243)

plot(t,y2,'linewidth',1);

% ylim([-1 1])

title('TASK PART 2: signal 2 in time domain');

grid on;

subplot(244)

fz2 = -Fs/2:Fs/length(Z2):Fs/2-(Fs/length(Z2));

stem(fz2,Z2,'linewidth',1);

% xlim([-20 20])

title('TASK PART 2: signal 2 in freq domain');

grid on;

subplot(245)

plot(t,y3,'linewidth',1);

% ylim([-1 1])

title('TASK PART 2: signal 3 in time domain');

grid on;

subplot(246)

fz3 = -Fs/2:Fs/length(Z3):Fs/2-(Fs/length(Z3));

stem(fz3,Z3,'linewidth',1);

% xlim([-20 20])

title('TASK PART 2');

grid on;

%part 3:

% USSB AM modulation for each message with a 1 kHz guard band

Fc1 = 5000; % Carrier frequency for message 1 (5 kHz)

Fc2 = 9000; % Carrier frequency for message 2 (9 kHz)

Fc3 = 13000; % Carrier frequency for message 3 (13 kHz)

guard\_band = 1e3; % Guard band (1 kHz)

% Modulate messages using USSB AM

modulated\_signal1 = signal1 .\* cos(2\*pi\*(Fc1 + guard\_band/2)\*t);

modulated\_signal2 = signal2 .\* cos(2\*pi\*(Fc2 + guard\_band/2)\*t);

modulated\_signal3 = signal3 .\* cos(2\*pi\*(Fc3 + guard\_band/2)\*t);

% Combine modulated signals with guard band

combined\_signal = modulated\_signal1 + modulated\_signal2 + modulated\_signal3;

% SSB-FDM modulation using USSB AM

Fc\_final = 400000; % Carrier frequency for final modulation (400 kHz)

ssb\_fdm\_signal = combined\_signal .\* cos(2\*pi\*(Fc\_final)\*t);

% Plot the original messages, modulated signals, and final SSB-FDM signal

figure;

subplot(4,1,1);

plot(t, signal1);

title('Message 1');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(4,1,2);

plot(t, signal2);

title('Message 2');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(4,1,3);

plot(t, signal3);

title('Message 3');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(4,1,4);

plot(t, ssb\_fdm\_signal);

title('SSB-FDM Signal');

xlabel('Time (s)');

ylabel('Amplitude');

% part 4

% Standard Hilbert Transform

m1\_hat = imag(hilbert(signal1));

figure;

subplot(2,2,1)

plot(t,signal1), title('m(t)'), xlabel('t (sec)'), grid

subplot(2,2,2)

plot(t,m1\_hat), title('Hilbert Transform of m1(t)'), xlabel('t (sec)'), grid

Fc = 400000 %400KHz

fc1 = 5000 % 5KHz multiplexing for m1

ct = cos(2\*pi\*fc1\*t)

ct\_phaseshift = sin(2\*pi\*fc1\*t)

ut1 = (signal1.\*ct)-m1\_hat.\*ct\_phaseshift

subplot(2,2,3)

plot(t,ut1), title('multiplexed c1'), xlabel('t (sec)'), grid

% Simulation loop for different SNR values

SNR\_values = [5, 10, 20];

for snr\_dB = SNR\_values

% Add AWGN to the SSB-FDM signal

received\_signal = awgn(ssb\_fdm\_signal, snr\_dB, 'measured');

% Demodulation for Message 1

downconverted\_signal1 = received\_signal .\* cos(2\*pi\*(Fc1 + guard\_band/2)\*t);

filtered\_signal1 = lowpass(downconverted\_signal1, guard\_band, Fs);

demodulated\_signal1 = abs(hilbert(filtered\_signal1));

% Demodulation for Message 2

downconverted\_signal2 = received\_signal .\* cos(2\*pi\*(Fc2 + guard\_band/2)\*t);

filtered\_signal2 = lowpass(downconverted\_signal2, guard\_band, Fs);

demodulated\_signal2 = abs(hilbert(filtered\_signal2));

% Demodulation for Message 3

downconverted\_signal3 = received\_signal .\* cos(2\*pi\*(Fc3 + guard\_band/2)\*t);

filtered\_signal3 = lowpass(downconverted\_signal3, guard\_band, Fs);

demodulated\_signal3 = abs(hilbert(filtered\_signal3));

figure;

plot(t, demodulated\_signal1, 'r', t, demodulated\_signal2, 'g', t, demodulated\_signal3, 'b');

title(['Demodulated Signals (SNR = ' num2str(snr\_dB) ' dB)']);

xlabel('Time (s)');

ylabel('Amplitude');

legend('Message 1', 'Message 2', 'Message 3');

f\_demod = -Fs/2:Fs/length(demodulated\_signal1):Fs/2-(Fs/length(demodulated\_signal1));

figure;

subplot(3,1,1);

stem(f\_demod, abs(fftshift(fft(demodulated\_signal1))), 'r');

title('Magnitude Spectrum of Demodulated Signal 1');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

subplot(3,1,2);

stem(f\_demod, abs(fftshift(fft(demodulated\_signal2))), 'g');

title('Magnitude Spectrum of Demodulated Signal 2');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

subplot(3,1,3);

stem(f\_demod, abs(fftshift(fft(demodulated\_signal3))), 'b');

title('Magnitude Spectrum of Demodulated Signal 3');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

legend('Message 1', 'Message 2', 'Message 3');

% Listen to recovered audio messages

soundsc(demodulated\_signal1, Fs);

pause(2); % Wait for the audio to finish playing

soundsc(demodulated\_signal2, Fs);

pause(2); % Wait for the audio to finish playing

soundsc(demodulated\_signal3, Fs);

pause(2); % Wait for the audio to finish playing

end

%---------------------------------------------------------------