1. **Explanation of Algorithm:**

   1st code snippet:

```
clc;
clear all;
close all;
n=0:1023;
Ts=1/1000;
N=1000;
frame=zeros(N,length(n));

% Transmitter Side

%Band Pass Data
% for k=1:N
%      xi = 1344*cos(0.06*pi*n)+864*cos(0.18*pi*n)+8543*cos(0.38*pi*n)-
43*cos(0.8*pi*n);
%      xq = 1344*sin(0.06*pi*n)+864*sin(0.18*pi*n)+8543*sin(0.38*pi*n)-
43*sin(0.8*pi*n);
%     data=xi+j*xq;
%     power(k)=sum(abs(data.^2))/length(data);
%     signal_power_dBW(k)=10*log10(power(k));
%     frame(k,:)=data;
% end

V=zeros(1,4);
%Random Data
for k=1:N
   xi=-2^16+(2*2^16-1)*randn(1,1024);

    xq=-2^16+(2*2^16-1)*randn(1,1024);

    data=xi+j*xq;
    power(k)=sum(abs(data.^2))/length(data);
    signal_power_dBW(k)=10*log10(power(k));
    frame(k,:)=data;
end
```

In this part, random and bandpass data are generated, and all the data are passed through a frame that can contain 1024 data points. In this way, 1000 frames are generated. Also, signal power (in dBW) is also calculated for these two types of data.

2nd Code Snippet: (Transmitter Part)

```
h0_coeff_g = [-1 0 3 0 -8 0 21 0 -45 0 91 0 -191 0 643 1024 643 0 -191 0 91 0
-45 0 21 0 -8 0 3 0 -1];
h0_coeff = h0_coeff_g/2050;
n=0:[length(h0_coeff)-1];
```

```
h1_coeff = (-1).^n.*h0_coeff;
len=length(h0_coeff);

SNR=0:20;
%% Analysis Part

for k1=1:length(SNR)
    MSError_frame = 0;
for k=1:N
    % Channel 1
    MSError2=zeros(1,1024);

    noise_power_dBW=signal_power_dBW(k)-SNR(k1);
    h0_1=upsample(h0_coeff,2);
    h0_2=upsample(h0_coeff,4);
    h0_f= conv(conv(h0_coeff,h0_1),h0_2);
    h0 =downsample(h0_f,8);
    v0_a1=conv(h0_f,frame(k,:));
    v0_a = downsample(v0_a1,8);

    % Channel_2
    h1_1=upsample(h0_coeff,2);
    h1_2=upsample(h1_coeff,4);
    h1_f= conv(conv(h0_coeff,h1_1),h1_2);
    h1 =downsample(h1_f,8);
    v1_a1=conv(h1_f,frame(k,:));
    v1_a=downsample(v1_a1,8);

    % Channel_3
    h2_1=upsample(h1_coeff,2);
    h2_f= conv(h0_coeff,h2_1);
    h2 =downsample(h2_f,4);
    v2_a1=conv(h2_f,frame(k,:));
    v2_a=downsample(v2_a1,4);

    % Channel_4
    h3 =h1_coeff;
    v3_a1=conv(h3,frame(k,:));
    v3_a=downsample(v3_a1,2);
```

In this section, low-pass filter coefficients are defined, and high-pass filter coefficients are generated from the low-pass filter coefficients. In the analysis part, 4-channel polyphase filters are introduced from where V0, V1,V2,V3 coefficients are generated.

3rd Code Snippet:

```
V = [bitsll(k,10)+26 bitsll(k,11)+(3*26) bitsll(k,12)+(5*26)
bitsll(k,13)+(7*26)];
```

```
    for kc=1:4
        PN1(kc)=xor(xor(bitget(V(kc),1),bitget(V(kc),4)),bitget(V(kc),26));

PN2(kc)=xor(xor(xor(xor(bitget(V(kc),1),bitget(V(kc),2)),bitget(V(kc),3)),bit
get(V(kc),4)),bitget(V(kc),26));

        PN1_key(kc)=1-2*PN1(kc);
        PN2_key(kc)=1-2*PN2(kc);
    end
        V0_sc=real(v0_a)*PN1_key(1)+j*imag(v0_a)*PN2_key(1);
        V1_sc=real(v1_a)*PN1_key(2)+j*imag(v1_a)*PN2_key(2);
        V2_sc=real(v2_a)*PN1_key(3)+j*imag(v2_a)*PN2_key(3);
        V3_sc=real(v3_a)*PN1_key(4)+j*imag(v3_a)*PN2_key(4);

        %% Interleaving
        S = zeros(1,length(data));
        for count=1:length(data)
            if (rem(count,8)==1)
              S(count)=V0_sc(((count-1)/8)+1);
            elseif (rem(count,8)==2)
              S(count)=V3_sc(((count-2)/8)+1);
            elseif (rem(count,8)==3)
              S(count)=V1_sc(((count-3)/8)+1);
            elseif (rem(count,8)==4)
              S(count)=V3_sc(((count-4)/8)+1);
            elseif (rem(count,8)==5)
              S(count)=V2_sc(((count-5)/8)+1);
            elseif (rem(count,8)==6)
              S(count)=V3_sc(((count-6)/8)+1);
            elseif (rem(count,8)==7)
              S(count)=V2_sc(((count-7)/8)+1);
            else
              S(count)=V3_sc(((count-8)/8)+1);
            end
            end
```

In this section, scrambling and interleaving are performed. In the scrambling part, personal key is generated according to my Group ID (26), and in the interleaving part data is generated accordingly. **It is noted that the S array does not contain all the coefficients of V0, V1, V2, V3 as the size of the array is limited to 1024 as per the description of the project.**

4th Code Snippet: (Channel)

```
   %% Burst Formation
        freq_ID=60;
        freq_omega=(2*pi/128)*freq_ID;
        n=0:127;
        burst= cos(freq_omega*n)+j*sin(freq_omega*n);
        T=[burst S];
```

```
        np=0:(length(T)-1);
        freq_offset=-2*78.125;
        offset_signal=cos(2*pi*freq_offset*np)+j*sin(2*pi*freq_offset*np);
        corrupted=T.*offset_signal;
        noise=wgn(1,length(corrupted),noise_power_dBW);
        R(k1,:)=corrupted+noise;
```

In this section, burst signal of 60 Hz is generated to detect the frequency offset. Also, a gaussian noise (having specific power level) further corrupts the signal that will be received by the receiver. Both the transmitted and received signal have the size of 1152.

5th Code Snippet: (Receiver)

```
%% Receiver Data Side
        dft_value=fft(R(k1,:),128);
        [M,i(k1)]=max(dft_value);
        detct_freq_offset(k1)=(i(k1)-1)*(2*pi/128)-freq_omega;
        detect_freq_Hz(k1)=(detct_freq_offset(k1)*128)/(2*pi);
        offset_comp=cos(detct_freq_offset(k1)*np)-
j*sin(detct_freq_offset(k1)*np);
        Sp=R.*offset_comp;
```

In this section, the frequency offset is detected in the receiver side. The index of the maximum DFT value will be the frequency ID that will be further used to compensate for the offset signal.

6th Code Snippet:

```
  %% De-Interleaver
        for cout=1:length(Sp)
            if(rem(cout,8)==1)
                V0_scp(((cout-1)/8)+1)=Sp(cout);
            end
            if(rem(cout,8)==3)
                V1_scp(((cout-3)/8)+1)=Sp(cout);
            end
            if(rem(cout,8)==5)
                V2_scp(((cout-5)/8)+1)=Sp(cout);
            end
            if(rem(cout,8)==7)
                V2_scp(((cout-7)/8)+1)=Sp(cout);
            end
            if(rem(cout,2)==0)
                V3_scp(((cout-2)/2)+1)=Sp(cout);
            end
        end
        % Receiver Side_Descrambling and Systhesis
        V0_dsc=real(V0_scp)*PN1_key(1)+j*imag(V0_scp)*PN2_key(1);
        V1_dsc=real(V1_scp)*PN1_key(2)+j*imag(V1_scp)*PN2_key(2);
        V2_dsc=real(V2_scp)*PN1_key(3)+j*imag(V2_scp)*PN2_key(3);
```

```
V3_dsc=real(V3_scp)*PN1_key(4)+j*imag(V3_scp)*PN2_key(4);
```

In this section, de-interleaving and descrambling is performed on the receiver side.

7th Code Snippet:

```
  %% Synthesis Part

    f0_coeff = h0_coeff;
    f1_coeff = -h1_coeff;
    % Channel 1
    f0_u1=upsample(f0_coeff,4);
    f0_u2=upsample(f0_coeff,2);
    f0 = conv(conv(f0_u1,f0_u2),f0_coeff);
    in_0=upsample(V0_dsc,8);
    x0_rcon=conv(in_0,f0);

    % Channel 2
    f1_u1 = upsample(f1_coeff,4);
    f1_u2 = upsample(f0_coeff,2);
    f1 = conv(conv(f1_u1,f1_u2),f0_coeff);
    in_1=upsample(V1_dsc,8);
    x1_rcon=conv(in_1,f1);

    % Channel 3
    f2_u1=upsample(f1_coeff,2);
    f2 = conv(f2_u1,f0_coeff);
    in_2=upsample(V2_dsc,4);
    x2_rcon=conv(in_2,f2);
    x2_rcon_zero=[x2_rcon, zeros(1,(length(x1_rcon)-length(x2_rcon)))];

    % Channel 4
    in_3=upsample(V3_dsc,2);
    x3_rcon=conv(in_3,f1_coeff);
    x3_rcon_zero=[x3_rcon, zeros(1,(length(x1_rcon)-length(x3_rcon)))];

    xr =x0_rcon+x1_rcon+x2_rcon_zero+x3_rcon_zero;
    %% MSE calculation
    sum=0;
    for k=1:1024
        MSEr2(k)=abs(data(k)-xr(k))^2/abs(data(k))^2;
        sum=sum+MSEr2(k);
    end
    MSError2=sqrt(sum)/1024;
    MSError_frame=MSError_frame+MSError2 ;
    end
    MSError_avg(k1) = MSError_frame/N;
end
    plot(SNR,MSError_avg,'--g');
```

```
xlabel('SNR in dB'); ylabel('MSE')
```

This synthesis part is performed in the receiver section that helps to reconstruct the signal. After reconstruction, MSE is computed for each frame and for each SNR level. The final MSE at a specific SNR level will be arithmetic mean of the MSE for all frames. Finally, MSE vs SNR is plotted.

One of the weak points of the design is to discard the coefficient values of V0, V1, V2, V3 while constructing the S frame of 1152 size. This action leads to significant loss of data information. The frequency offset is not properly detected and hence not removed in this design. So the reconstructed signal deviates from the original signal.

One of the drawbacks of the design is the higher runtime of the simulation due to the execution of multiple loops in a single script.

2. (a) For Band-Pass Data:

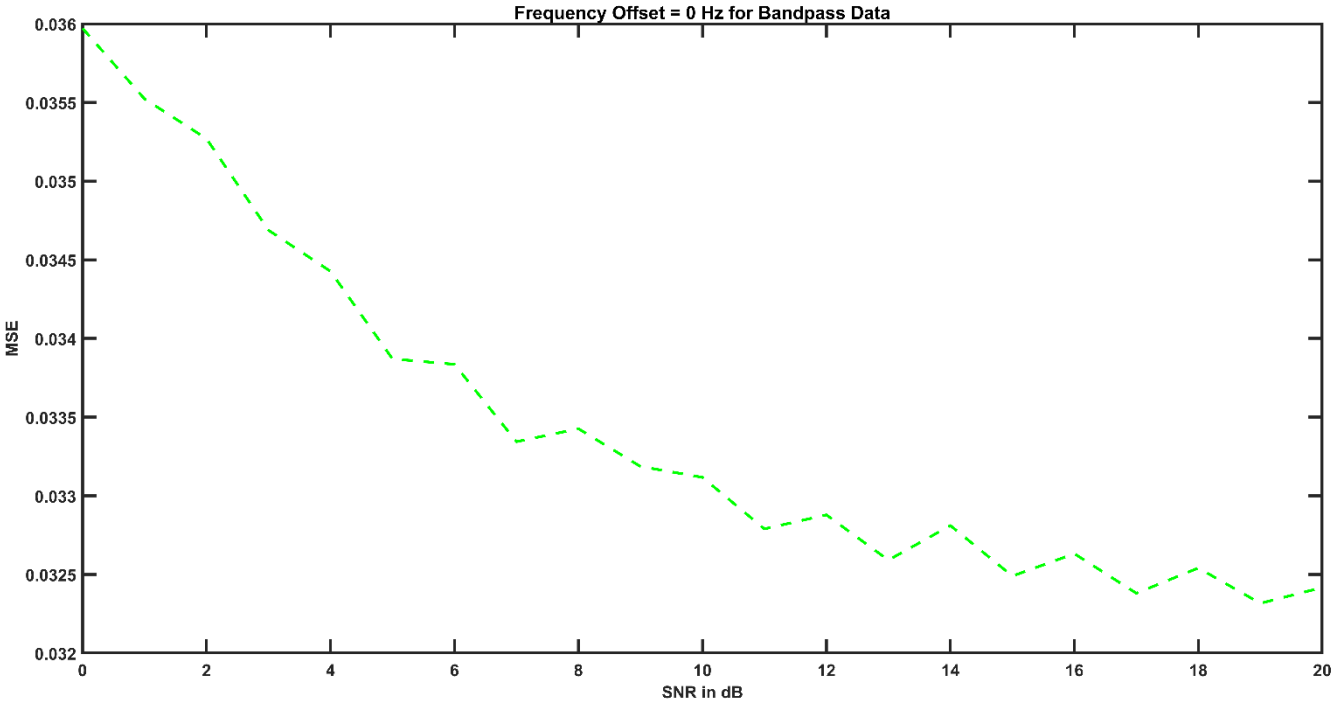

Figure 1: MSE vs SNR plot for Bandpass data for offset = 0 Hz

(b)

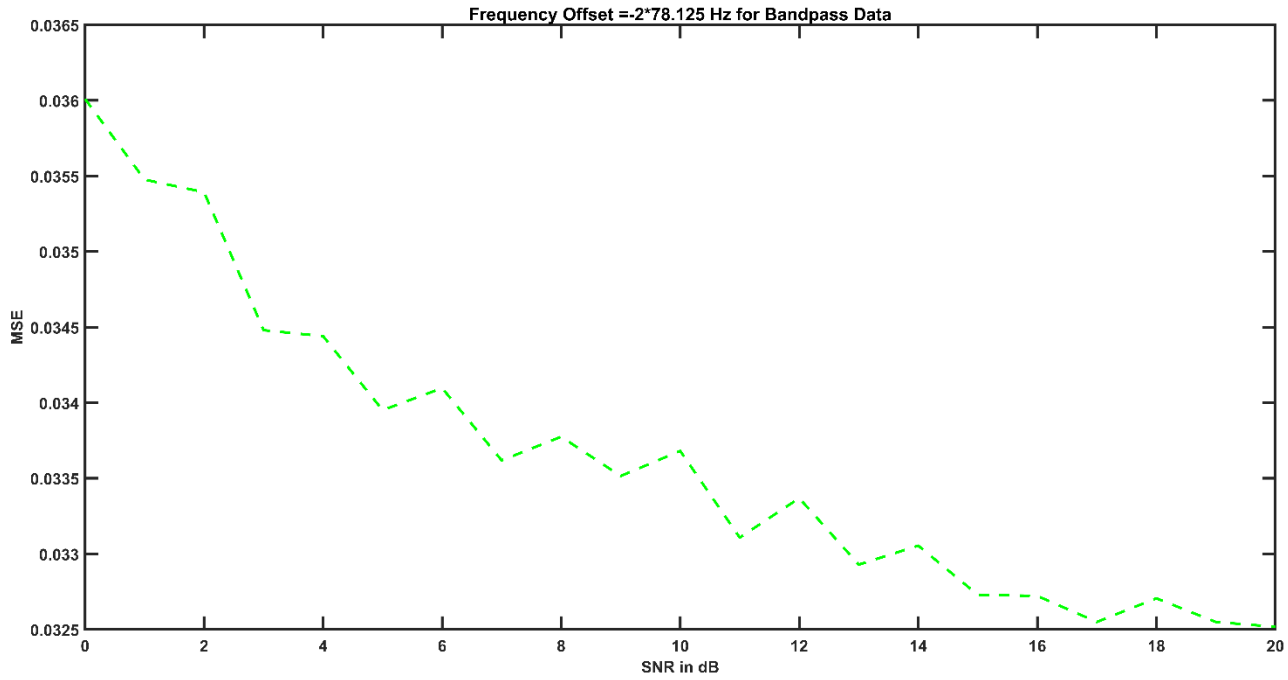

Figure 2: MSE vs SNR plot for Bandpass data for offset = -2*78.125 Hz

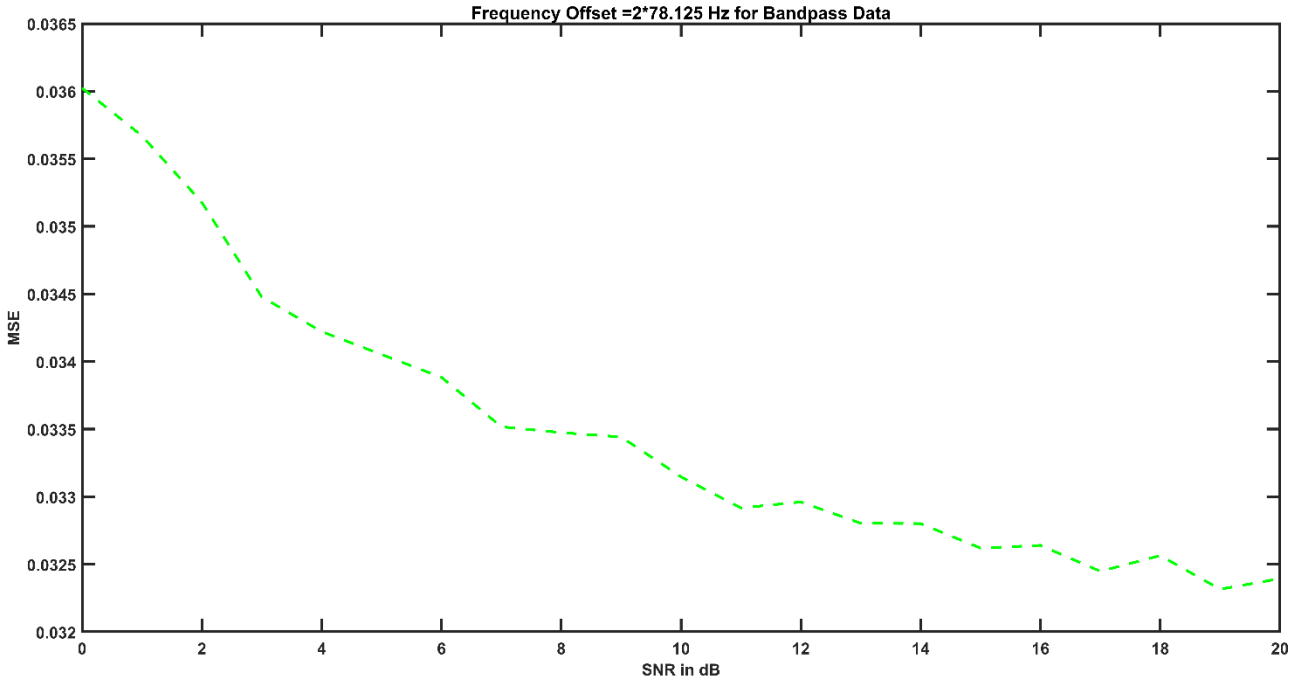

Figure 3: MSE vs SNR plot for Bandpass data for offset = 2*78.125 Hz

(c)

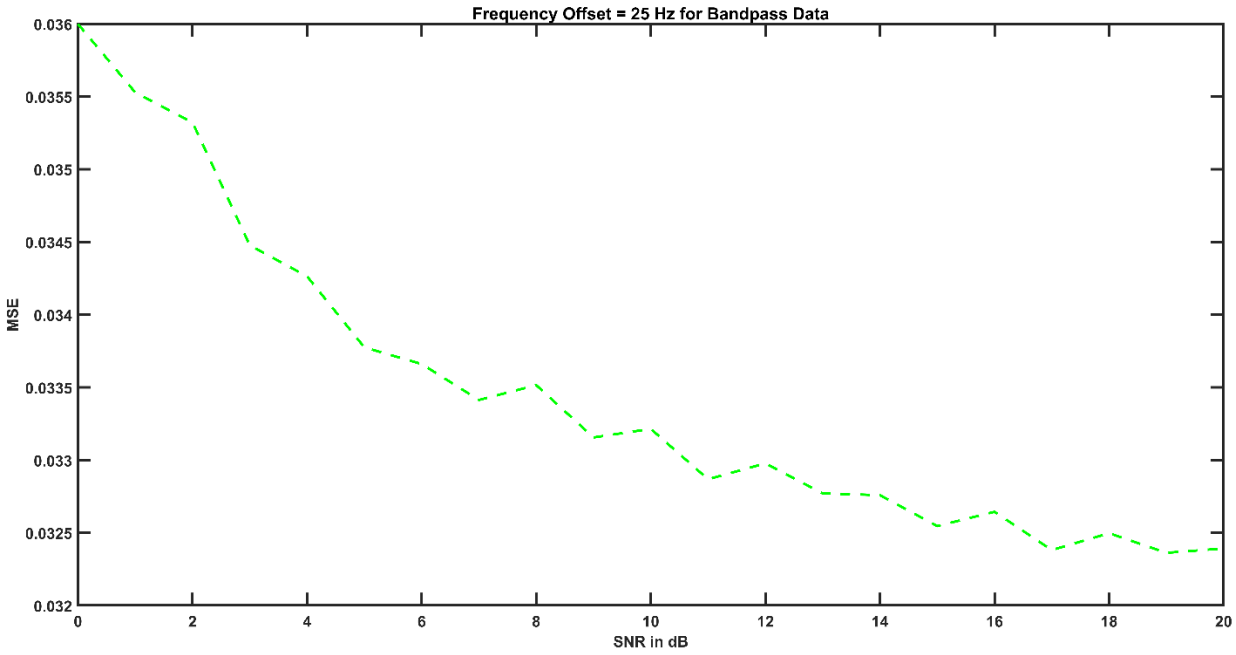

Figure 4: MSE vs SNR plot for Bandpass data for offset = 25 Hz

For Random Data:

(a)

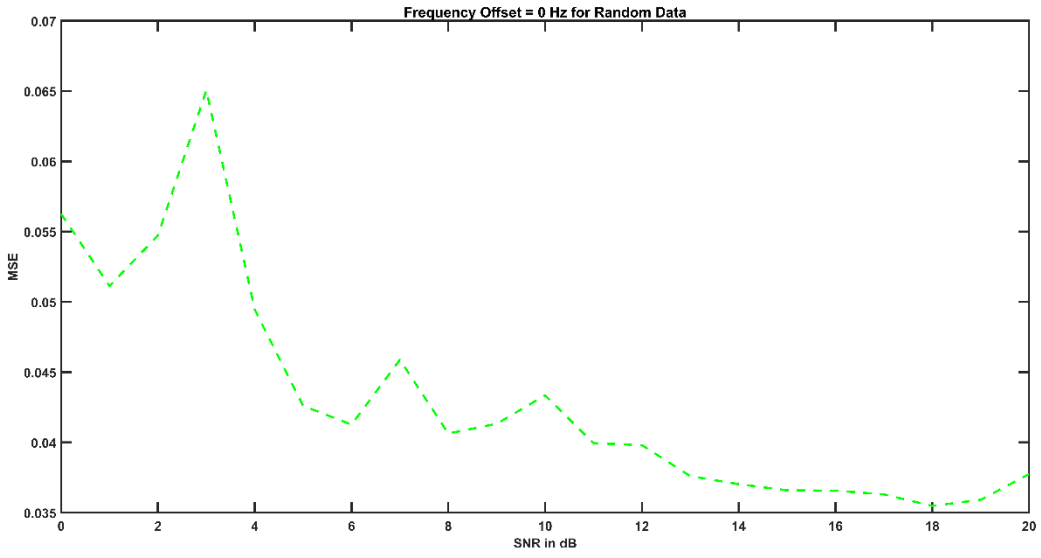

Figure 5: MSE vs SNR plot for Random data for offset = 0 Hz

(b)

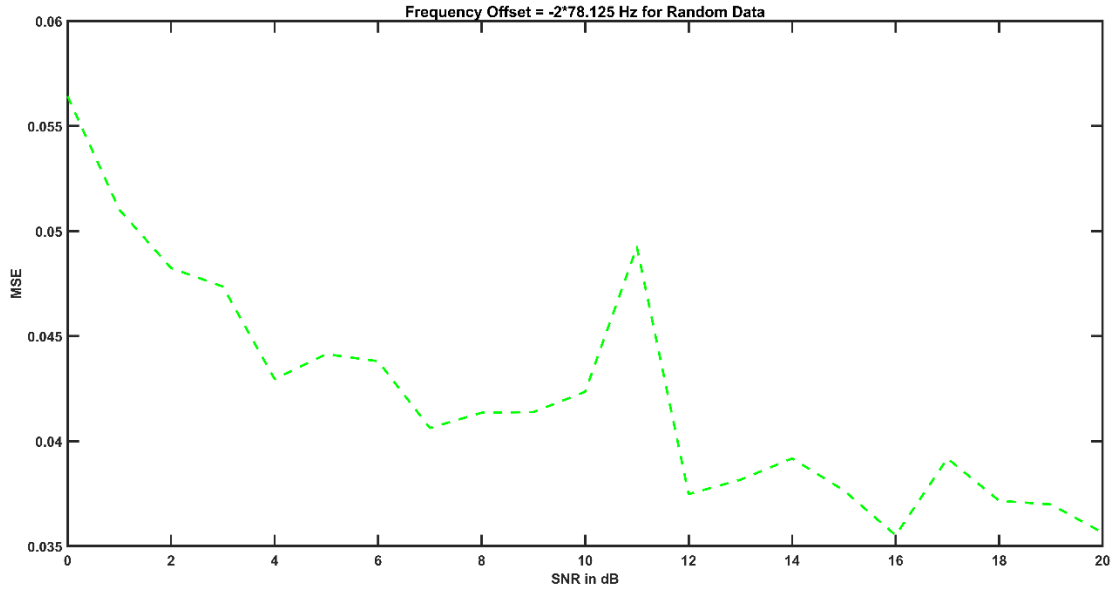

Figure 6: MSE vs SNR plot for Random data for offset = -2*78.125 Hz

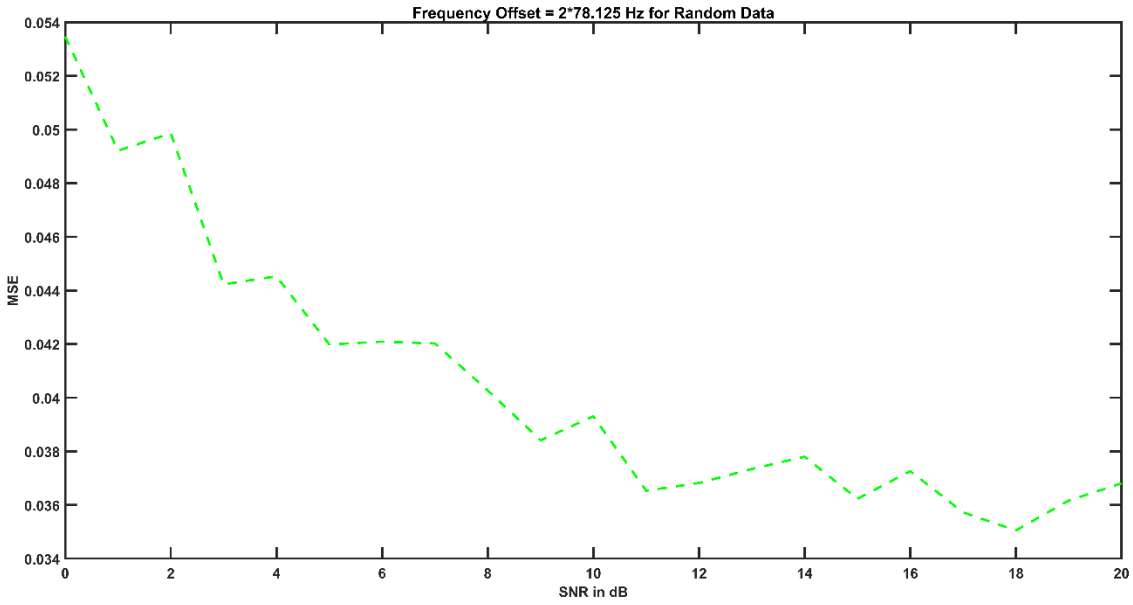

Figure 7: MSE vs SNR plot for Random data for offset = 2*78.125 Hz

(c)

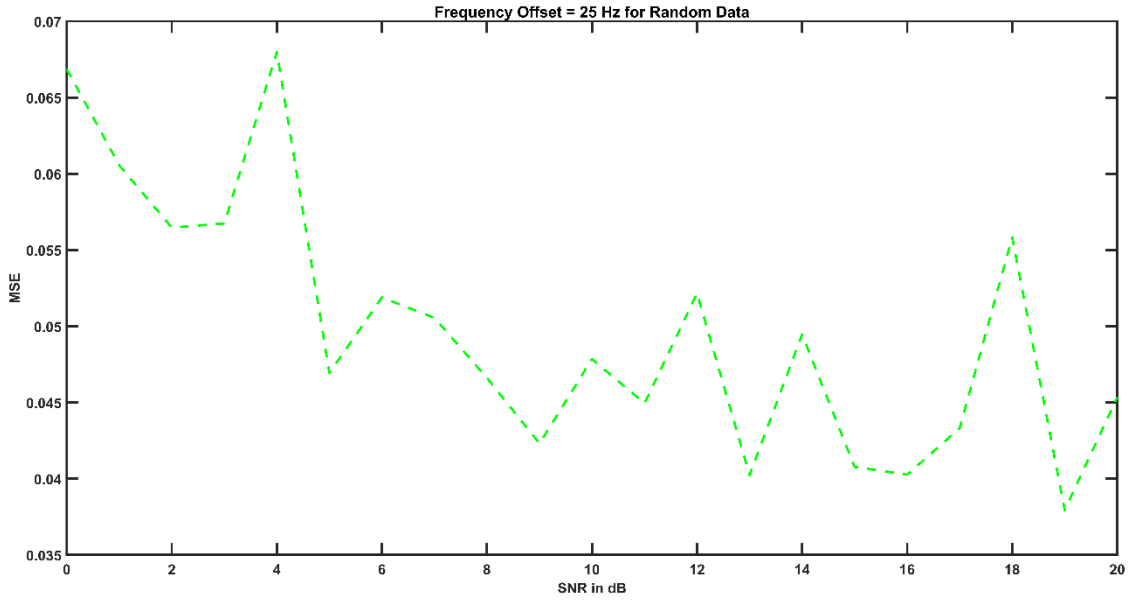

Figure 8: MSE vs SNR plot for Random data for offset = 25 Hz

(d) For both the random and band-pass data the MSE is kind of decreasing in nature. With the increase of SNR, the dominance of noise decreases in the received signal and it becomes convenient for the receiver to reconstruct the signal from the corrupted signal. Hence, the MSE reduces with the increase of SNR. In case of band-pass data, the decreasing trend is lower than the random data and the MSE can be assumed constant with the increase of SNR, although the graph shows decreasing trend due to the limit of y axis. For both random and band-pass data, the signal power is much higher than the noise power, so the 20dB increase of SNR doesn't reduce the MSE much.

In random input, the data varies at every frame level due to the randomness, but in case of band-pass input, the data remain same. This leads to some fluctuating nature of MSE in case of random data.

Bonus Part

(a)

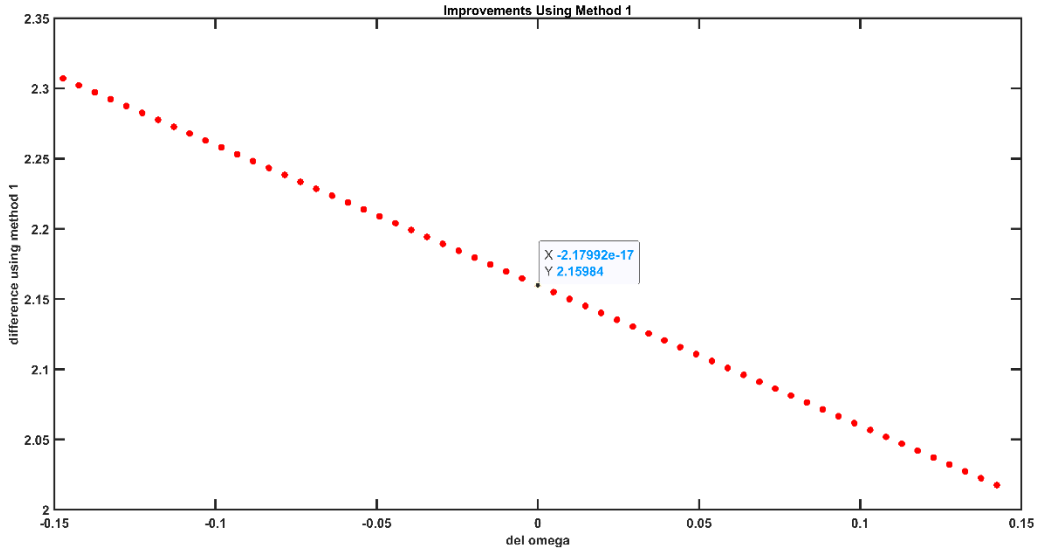

Figure 9: Frequency Compensation Technique Using Method 1

(b)

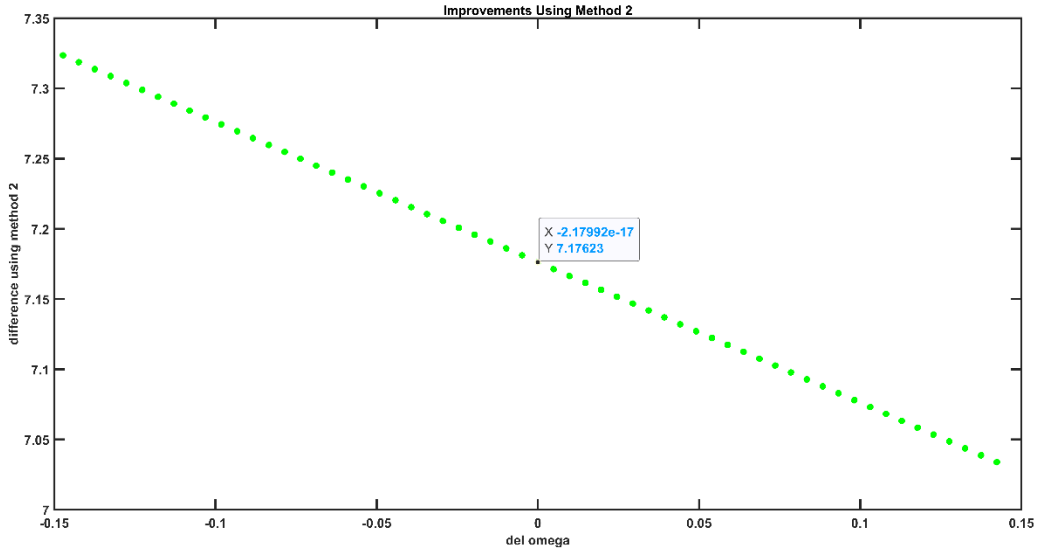

Figure 10: Frequency Compensation Technique Using Method

(c) Using this method 1 and 2, the suitable frequency offset can be estimated. The point where the value of y axis is zero needs to be utilized here and from that, the approximate correct ID can be retrieved. It shows that, with the increase of the value of k, the graphs for both methods become downward.