APPENDICES

APPENDIX - I

SOURCE CODE

```
% M-File to estimate the channel and to develop the bit loading profile

clc;
clearall;
closeall;


% Inputs from user
pow = input('Enter the total power of the signal to be transmitted: ');
l(1)=input(' enter the length to be considered in Km : ');
margin= input('enter the margin to be given in dB: ');
gapfactordb=9.8+margin;
disp('taking a bit error tolerance of 10^(-6) and no coding gain the gap factor in dB is: ');
disp(gapfactordb);

%Define the properties of the transmitter.
tsym=3.333333333333334e-05                    %symbol duration to make
df=1875;
rsym=1/tsym;                                  %symbol rate
N=16;                                         %Number of parallel symbols
tnew=N*tsym;                                  %new symbol duration = 800 micro
seconds
rnew=1/tnew;
df=rnew;                                      %frequency spacing

%frequency values for the sixteen tones
fval=zeros([1 16]);
fori=1:16
```

```
fval(i)=1875+(i-1)*df;
fval(i)=floor(fval(i));
end

% QPSK Modulation of the bit pattern generated above
% 64 tones in total, inter-tone spacing of 500 Hz
seq = randi([0 1],32,1);                    % Random 32 bit sequence  - 16 tones in the 20KHz channel
mod = modem.pskmod(4,pi/4);
mod.inputtype = 'bit';
mod.SymbolOrder = 'gray';
q = modulate(mod,seq);
%figure(3)
disp(q);

% IFFT Definition
Fs = 120000;                                %so that the 20k component of freq exists.
T = 1/Fs;
L = 120000;
t = (0:L-1)*T;
freqdom=zeros([1 120000]);
q = [q;flipud(conj(q))];                    %complex conjugate of the signals added to fft
input('Press Enter to view the inputs to the IFFT block');
disp('Input to the IFFT Block');
%figure(1)
disp(q);

fori = 1:16
freqdom(fval(i)) = q(i);
freqdom(120000-fval(i)) = q(33-i);
```

```
end


% Displaying the transmitted signal
%figure(2)
subplot(2,3,1);
plot(abs(freqdom(1:30000)));                    %display only upto 30k
holdon;
xlabel('Frequency - Hz');
ylabel('Tone Amplitude in Watt');
title('Multi-Tone Structure Transmitted - Frequency Domain');
gridon;
freqdom1 = freqdom.*60000;                      %according to ifftdefiniton
realsig = ifft(freqdom1);


% Channel and Noise estimation
k=1.5;                                          % Spreading factor
fmax = 30500;                                           % Maximum value of frequency
required for calculation
 noise = ambient_noise(fmax);                   % Function call to calculate noise

% Use any one of the three channel models and comment the other two

%prop_const=thorp_model(fmax);
%prop_const=fisher_simmons_model(fmax);
prop_const=anslie_mccolm_model(fmax);

% Calculation of attenuation
L=l(1);
AdB1 = (k*10*log10(L) + L*prop_const);
C1=AdB1./10;
A1=10.^(C1);
```

```
% Calculating the effect of channel on the signal in frequency domain
freqdom_2(1:30000)=freqdom(1:30000)./A1(1:30000);
subplot(2,3,2);
plot(abs(freqdom_2));
xlabel('Frequency - Hz');
ylabel('Tone Amplitude in Watt');
title('Multi-Tone Structure at Receiver - Frequency Domain');
gridon;


% Convertion of power to voltage and then to dBreuPa %
pow = pow/16;                                          % Dividing the power in all tones
equally
imp = 2870;                                            % Refer data sheets for impedance
value
volts= sqrt (pow * imp);
convert = 31.6e6;    %1v=31.6e6 micropascal                          % standard for
sound projectors
gapfactor = 10^(gapfactordb/20);
noise1= zeros(1,16);

% SNR and CbyBcalculation
SNR=zeros(1,16);
CbyB=zeros(1,16);

for G=1:16
Signal_power_dbreupa= volts * convert;
fori=(G*1875)-500:1:(G*1875)+500;
        noise1(G) = noise1(G) + A1(i)*noise(i);        %Integrating the noise
end

SNR(G)=(Signal_power_dbreupa/(noise1(G)))^2;
CbyB(G)=(log2(1+(SNR(G)/gapfactor)));
SNR(G)= 10 * log10(SNR(G));
```

```
end
subplot(2,3,4);
stem(SNR);xlabel('TONE NUMBER');
ylabel('SNR in dB');
title(' SNR Profile');
gridon;
subplot(2,3,5);
stem(CbyB);
xlabel('TONE NUMBER');
ylabel('C/B in bits/Hz');
title(' C/B Profile');
gridon;


% Rounding off by calling the equalize function by using any one of the functions and comment the other two
CbyB_round = equalize_2(CbyB');                    % Mean Offset algorithm
subplot(2,3,6);
stem(CbyB_round);
xlabel('TONE NUMBER');
ylabel('C/B bits/Hz');
title(' C/B Profile after Rounding off');
gridon;

% Recalculation of Power and plotting it
pow_tones=pow*((2.^(CbyB_round))-1)./((2.^(CbyB'))-1);
subplot(2,3,3);
stem(pow_tones);
xlabel('TONE NUMBER');
ylabel('POWER in Watt');
title(' PSD AFTER ROUNDING');
gridon;
```

Functions used:

Ambient noise function

```
function [noise]=ambient_noise(fmax)
f=(1:1:fmax)/1000;
sh=1;                                     % Shipping factor
w=25;                                     % Wind Speed

% Noise calculation for individual noise
Turbulence_noise = 10.^((17 - 30*log10(f))./20);
Shipping_noise = 10.^((40 + 20*(sh - 0.5) + 26*log10(f) - 60*log10(f + 0.3))./20);
Winddriven_noise = 10.^((50 + 7.5*sqrt(w) + 20*log10(f) - 40*log10(f+0.4))./20);
Thermal_noise = 10.^((-15 + 20*log10(f))./20);
noise=Turbulence_noise+Shipping_noise+Winddriven_noise+Thermal_noise;
end
```

Anslie model function

```
function[prop_const]=anslie_mccolm_model(fmax)
ph=8;                                     % ph value
s=35;                                     % Salinity in parts per trillion
d=1000;                                   % Depth in m
temp=10;                                  %Temperature in C
f=(1:1:fmax)/1000;
f1=0.78*sqrt(s/35)*exp(temp/26);
f2=42*exp(temp/17);
prop_const=(0.106*((f1*f.*f)./(f1*f1+f.*f))*exp((ph-
8)/0.56)+.52*(1+temp/43)*(s/35)*((f2*f.*f)./(f2*f2+f.*f))*exp(-d/6)+4.9*10^(-4)*f.*f*exp(-
(temp/27+d/17)));
```

```
end
```

Fisher simmons model function

```
function [prop_const]=fisher_simmons_model(fmax)
ph = 4;
S = 35;                                      % Salinity of water
d = 1000;                                    % depth of measurement
T = 10;                                      % temperature in degresscelsius
c = 1412 + 3.21*T + 1.19*S + 0.0167*d;       % speed of sound in m/s
t = 273 + T;
A1 = (8.86/c)*10^(0.78*ph - 5);              % Coefficients for the Equation
A2 = 21.44*(S/c)*(1 + 0.025*T);
A3 = 4.937*10^(-4) - 2.59*10^(-5)*T + 9.11*10^(-7)*T*T - 1.5*10^(-8)*T*T*T;
f1 = 2.8*sqrt(S/35)*10^( 4- 1245/t);
f2 = (8.17*10^(8-1990/t))/(1 + 0.0018*(S-35));
P1 = 1;
P2 = 1 - 1.37*(10^-4)*d + 6.2*(10^-9)*d*d;
P3 = 1 - 3.84*(10^-5)*d + 4.9*(10^-10)*d*d;
f=(1:1:fmax)/1000;
prop_const = (A1*P1*f1*f.*f)./(f1*f1 + f.*f) + (A2*P2*f2*f.*f)./(f2*f2 + f.*f) +
A3*P3*f.*f;
end
```

Thorp model function

```
function [prop_const]=thorp_model(fmax)
f=(1:1:fmax)/1000;
prop_const=(0.11*(f.*f)./(1+f.*f))+(44*(f.*f)./(4100+f.*f))+2.75*10.^(-4)*f.*f+0.003;
end
```

Round off algorithm function

```
function [q1] = equalize_2(q)
q=q';
diff_easy = 0;
fori=1:16
if ( diff_easy> 0 )
q1(i) = floor(q(i));
elseif ( diff_easy< 0 )
q1(i) = ceil(q(i));
else
q1(i) = round(q(i));
end
diff_easy =diff_easy + q1(i) - q(i);
end
q1=q1';
end
```