

MARKING CRITERIA AND STUDENT FEEDBACK

This section details the assessment criteria. The extent to which these are demonstrated by you determines your mark. The marks available for each criterion are shown. Lecturers will use the space provided to comment on the achievement of the task(s), including those areas in which you have performed well and areas that would benefit from development/improvement.

Common Assessment Criteria Applied	Marks available	Marks awarded
1. Research-informed Literature Extent of research and/or own reading, selection of credible sources, application of appropriate referencing conventions.		
There is no evidence of Research	20	0
2. Knowledge and Understanding of Subject Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.		
you show a reasonable knowledge of the Subject in task 1 in that you have managed to design a filter that works. In task 2 your treatment is rather superficial.	25	12
3. Analysis Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence		
There is little evidence of analysis apart from task 2. Although there are poor conclusions	25	5
4. Practical Application and Deployment Deployment of methods, materials, tools and techniques; application of concepts; formulation of innovative and creative solutions to solve problems.		
you have managed to use Matlab correctly	25	15
5. Skills for Professional Practice Attributes in professional practice: individual and collaborative working; deployment of appropriate media; presentation and organisation.		
A Quite Poor report - rather disappointing.	5	2

Assignment Mark Overall (Assessment marks are subject to ratification at the Exam Board. These comments and marks are to give feedback on module work and are for guidance only until they are confirmed.)	Late Submission Penalties (tick if appropriate)		34. %
	Up to one-week late		
	Over one week late		

Andrew Shannon

Digital Signal Processing

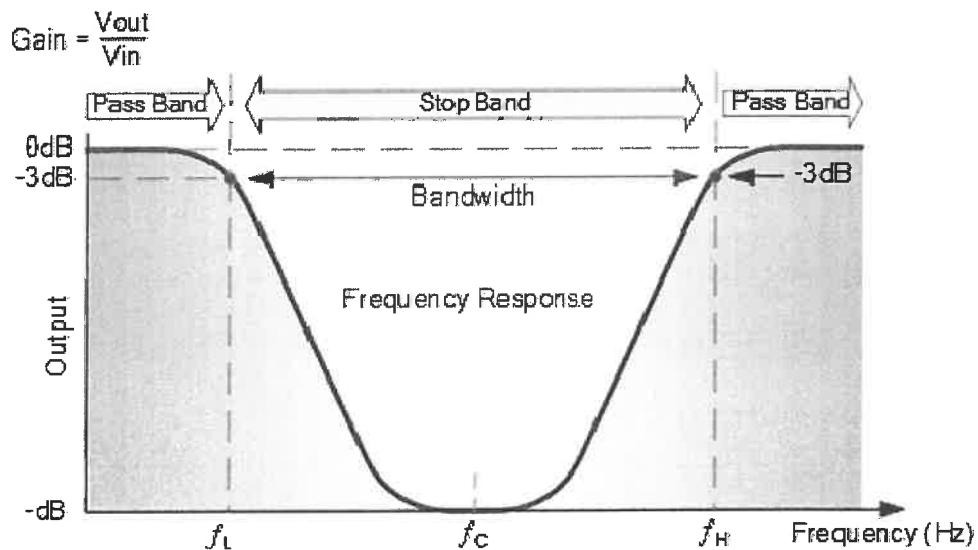
Assignment

Task one requires construction of a filter to remove an unwanted tone from an audio signal.

An audio file was given with a corrupt tone in it. Using the knowledge of DSP's a filter must be created to filter out the unwanted noise.

A band stop filter must be used to filter the unwanted frequency out of the audio file.

A band stop filter allows you to accurately cut unwanted frequencies out of sounds samples.



The image above shows an example of a band stop filter. The area labelled stop band are the frequencies that will get filtered out of the audio file and the areas labelled pass band are acceptable frequencies.

The filter must be made using matlab and must do the following:

1. Import corrupted tone
2. Play corrupted tone
3. Sample the corrupted tone
4. Identify frequencies
5. Filter (Band Stop)
6. Play new audio
7. Show graph of filter

Below is the matlab program showing the bandstop filter:

```
path(path,'Z:\My Music')
[y,Fs1] = audioread('Assignment.wav');

sound(y,Fs1)

pause

fs = Fs1; % Sampling Frequency
fn = fs/2; %Nyquist frequency
flowcut = 1000; % low frequency
fhighcut = 1300; % High Frequency
ft = 200; % Transition Band
Flowcut = flowcut/fn; % Normalised Frequency
Fhighcut = fhighcut/fn;
Ft = ft/fn;
Fc = (Flowcut + .5*Ft);
Ap = 0.1; %dB
AS = 50; %Attenuation in dB
N = 3.3/Ft;
yfilt = filter(a,1,y);
sound(yfilt,Fs1)

subplot(2,1,1);
x = linspace(0,10);
plot(y)
subplot(2,1,2);
%plot(yfilt)
L = length(y);
p = fft(y);
P2 = abs(p/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs1*(0:(L/2))/L;
plot(f,P1)

pause

subplot(2,1,1);
x = linspace(0,10);
plot(yfilt)
subplot(2,1,2);
%plot(yfilt)
L = length(yfilt);
p = fft(yfilt);
P2 = abs(p/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs1*(0:(L/2))/L;
plot(f,P1)

freqz(a,1,500)
```

Task 2

Task two requires analysis of the effect of noise in a QPSK transmission system. A QPSK modulator and demodulator are provided.

The task requires the following:

1. Generate tone
2. Sample tone
3. Digitise samples
4. QPSK Modulation
5. Add noise
6. Demodulate QPSK
7. Detect Errors
8. Report

Below is the matlab program to represent this:

```
% Demonstration of a sampled sin wave
%transmitted using QPSK Modulation and Demodulation
%in the presence of noise

%first generate a signal
clc; % clear the existing data and workspace
clear all;
frequency = 1000; %Make a 1kHz tone
period = 1/frequency;
cycles = 10; % generate cycles
t=[0:(period/10):(cycles*period)]; % so will be sampled at 10 times
frequency
yg = sin(2*pi*frequency*t);
y1 = yg+1;
% I will add an offset of 1 to the sin wave so that all values are positive
sigmax = 2; % 2 hence is the maximum value of the sin wave
%-----
% now generate sampled data

AtoD = 8; % 8 bit quantizer
nvals = 2^AtoD; % this gives nvals discret values

samplevals = 2/nvals; % this will then be the step size Note 2 is the max
value

sampls = fix(y1/samplevals);
samplelen = length(sampls);
datstr = strings([1,samplelen]);
for (getsample = 1:samplelen);
datstr(getsample) = dec2bin(sampls(getsample),8); % selects data samples
and ensures its 8 bits

end
%datain = dec2bin(sampls(2),8); % selects data samples and ensures its 8
bits

datain11=join(datstr);
```

Andrew Shannon

```
;
datain=char(datain11);
;
newdata1 = regexprep(datain,'0','0 ');
newdata1a = regexprep(newdata1,'1','1 ');% put space delimiters in
newdata2 = regexprep(newdata1a,' ',' ');% remove extra spaces
[sepbit,rembits] = strtok(newdata2);

for(i=1:(getsample*AtoD)) % send
    %[sepbit,rembits] = strtok(newdata2);
    bits(i) = str2num(sepbit);

    [sepbit,rembits] = strtok(rembits);

end

% here we could add coding eg hamming

data = bits;
%Number_of_bit=1024;
%data=randint(Number_of_bit,1);

figure(1)
stem(data, 'linewidth',2), grid on;
title(' Information before Transmitting ');
axis([ 0 (length(bits)) 0 1.5]);

data_NZR=2*data-1; % Data Represented at NZR form for QPSK modulation
s_p_data=reshape(data_NZR,2,length(data)/2); % S/P conversion of data

% QPSK modulator

br=10.^6;
f= br; %carrier frequency
T=1/br; %bit duration
t=T/99:T/99:T;
y=[]; %time space for 1 bit
y_in=[];
y_qd=[];
for(i=1:length(data)/2)
    y1=s_p_data(1,i)*cos(2*pi*f*t); % inphase component
    y2=s_p_data(2,i)*sin(2*pi*f*t) ;% Quadrature component
    y_in=[y_in y1]; % inphase signal vector
    y_qd=[y_qd y2]; %quadrature signal vector
    y=[y y1+y2]; % modulated signal vector
end
rnoise1 = rand(1,length(y))*5;
rnoise2 = rand(1,length(y))*-5;
rnoise = rnoise1+rnoise2;
Tx_sig=(y+rnoise); % transmitting signal after modulation
tt=T/99:T/99:(T*length(data))/2;

figure(2)

subplot(5,1,1);
```

```
plot(tt,y_in,'linewidth',3), grid on;
title(' wave form for inphase component in QPSK modulation ');
xlabel('time(sec)');
ylabel(' amplitude(volt0)');

subplot(5,1,2);
plot(tt,y_qd,'linewidth',3), grid on;
title(' wave form for Quadrature component in QPSK modulation ');
xlabel('time(sec)');
ylabel(' amplitude(volt0)');

subplot(5,1,3);
plot(tt,Tx_sig,'r','linewidth',3), grid on;
title('QPSK modulated signal (sum of inphase and Quadrature phase
signal)');
xlabel('time(sec)');
ylabel(' amplitude(volt0)');

subplot(5,1,4);
plot(tt,rnoise,'r','linewidth',3), grid on;
title('noise');
xlabel('time(sec)');
ylabel(' amplitude(volt0)');

subplot(5,1,5);
plot(tt,y,'r','linewidth',3), grid on;
title('qpsk without noise');
xlabel('time(sec)');
ylabel(' amplitude(volt0)');

%                               QPSK Demodulation
Rx_data=[];
Rx_sig=Tx_sig; % Received signal

for(i=1:length(data)/2)

    %%XXXXXXX inphase coherent dector XXXXXXXX
    Z_in=Rx_sig((i-1)*length(t)+1:i*length(t)).*cos(2*pi*f*t);
    % above line indicat multiplication of received & inphase carred signal

    Z_in_intg=(trapz(t,Z_in))*(2/T);% integration using trapizodial rule
    if(Z_in_intg>0) % d
        Rx_in_data=1;
    else
        Rx_in_data=0;
    end

    %       QPSK detector
    Z_qd=Rx_sig((i-1)*length(t)+1:i*length(t)).*sin(2*pi*f*t);
    %above line indicates multiplication of received & Quadphase carred
    signal

    Z_qd_intg=(trapz(t,Z_qd))*(2/T);%integration using trapizodial rule
    if (Z_qd_intg>0)% Decission Maker
        Rx_qd_data=1;
    else
        Rx_qd_data=0;
    end
end
```

```

end

Rx_data=[Rx_data Rx_in_data Rx_qd_data]; % Received Data vector
end

figure(3)
stem(Rx_data,'linewidth',3)
title('Information after Receiveing ');
axis([ 0 (length(Rx_data)) 0 1.5]), grid on;

bbbbbb= Rx_data - data;
biterr= num2str(bbbbb);
noerr=count(biterr,'0');
errors=length(bbbbb)-noerr

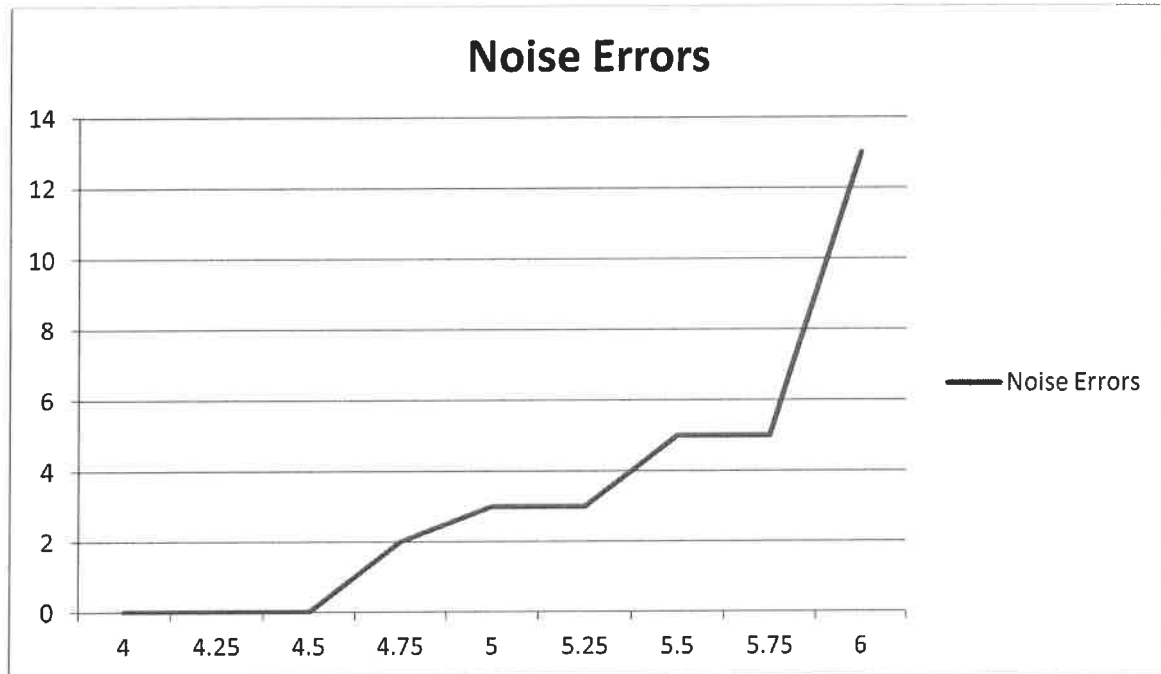
```

Adding noise to the system affects the amount of errors that occur. A test was carried out to show the effect of noise on the system. The more noise in a system the more errors that should occur.

Noise	Attempt number					Total
	1	2	3	4	5	
4.00	0	0	0	0	0	0
4.25	0	0	0	0	0	0
4.50	0	0	0	0	0	0
4.75	0	1	1	0	0	2
5.00	1	0	1	1	0	3
5.25	0	0	2	1	0	3
5.50	1	1	0	3	0	5
5.75	1	2	1	1	0	5
6.00	1	5	4	2	1	13

Above shows the results for the test. The table shows that the higher number of noise inputted, the higher amount of errors occurred.

Using the results from the table, a line chart can be created. The line chart shows the increase in errors as the amount of noise increases.



A hamming code could be used within this system to fix small errors from occurring.