

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 clearly some evidence of research - reading especially for task 2.	20	10
2. Knowledge and Understanding of Subject Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.		
You have gained a good grasp of this subject and demonstrate an interest in the subject.	25	20
3. Analysis Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence		
You have attempted to analyse both systems and would have gained a higher score if you had read more. Take a larger measurement sample in 2.	25	18
4. Practical Application and Deployment Deployment of methods, materials, tools and techniques; application of concepts; formulation of innovative and creative solutions to solve problems.		
I quite liked your idea of playing the - removed audio in task 1.	25	18
5. Skills for Professional Practice Attributes in professional practice: individual and collaborative working; deployment of appropriate media; presentation and organisation.		
More referencing would have been perfect otherwise a quite good report.	5	4

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)		70	%
	Up to one-week late			
	Over one week late			

HND Electrical and Electronic Engineering

Module: Digital Signal Processing Task 1

Lecturer: Linton Cook

Student Name: Gauvain Jay Chartier-Roberts

Student ID: CHA16832389

TOTAL
80%

Task 1.
No Referencing -
otherwise very good.

40/50

TASK 2

40/50

Contents

DSP – Task 1	3
Filter design process.....	3
Signal analysis and filter parameters	4
Filter application.....	5
Filter magnitude and phase response	6
Fast Fourier transform – filtered signal	7
Signal loss.....	8
Filter parameter testing.....	9
Conclusions.....	9

DSP – Task 1

For this task we were provided a digital file of some spoken dialogue that had been corrupted with a tone. The task was to identify the corrupting tone and construct a filter within Matlab that would reduce or remove the corrupting tone.

Filter design process

In order to identify the frequency of the corrupting tone, a fast Fourier transform was performed. The fast Fourier transform effectively allows us to analyse our signal in the time domain and produce an output in the frequency domain – a graphical output is produced that we can use to identify (by inspection) abnormal frequencies.

Once the likely corrupting frequency is identified, we use an appropriate window function to remove the tone. The parameters of the window function can be altered to provide a balance between filter complexity (the number of taps) and the range of frequency attenuation.

For our FIR filter design we will require a stop-band filter with minimum 50dB attenuation, so a Hamming window function is appropriate (53dB stop band attenuation). ✓

Signal analysis and filter parameters

The fast Fourier transform amplitude spectrum (Fig. 1) identified a strong response at 780Hz. ✓

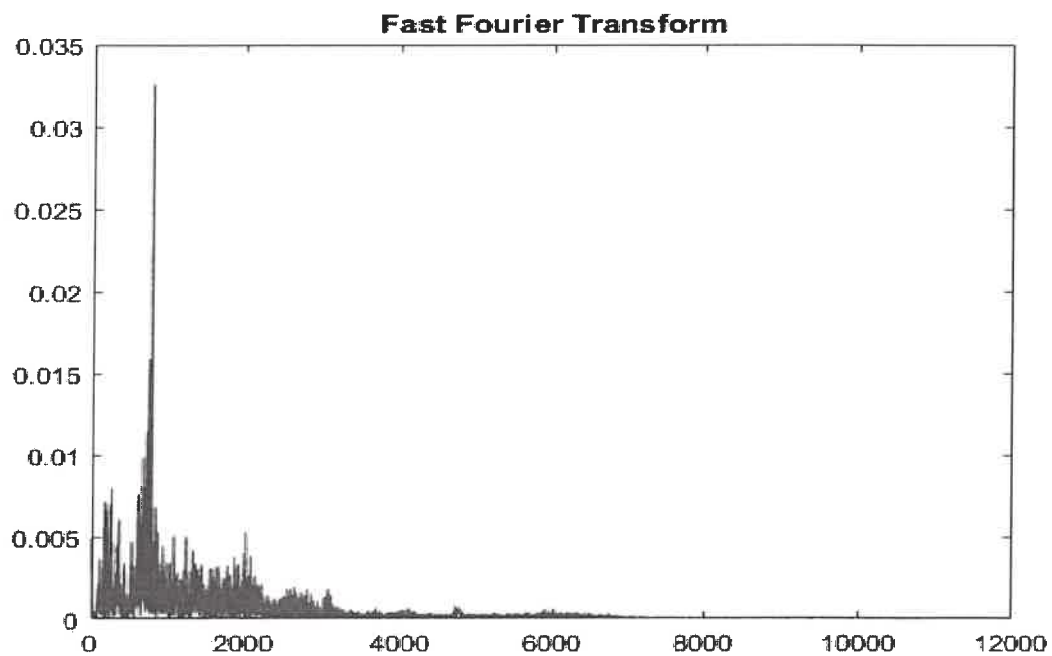


Figure 1: FFT on original signal

A stop-band filter was designed using a Hamming window function and the following filter parameters:

- Low cut off frequency: 650Hz
- High cut off frequency: 910Hz
- Transition band width: 100Hz
- Number of taps (N): 364+1

Filter application

The filter was applied and Figure 2 shows the spectrum of the initial tone and the filtered tone with a clear reduction in the signal magnitude after filtering.

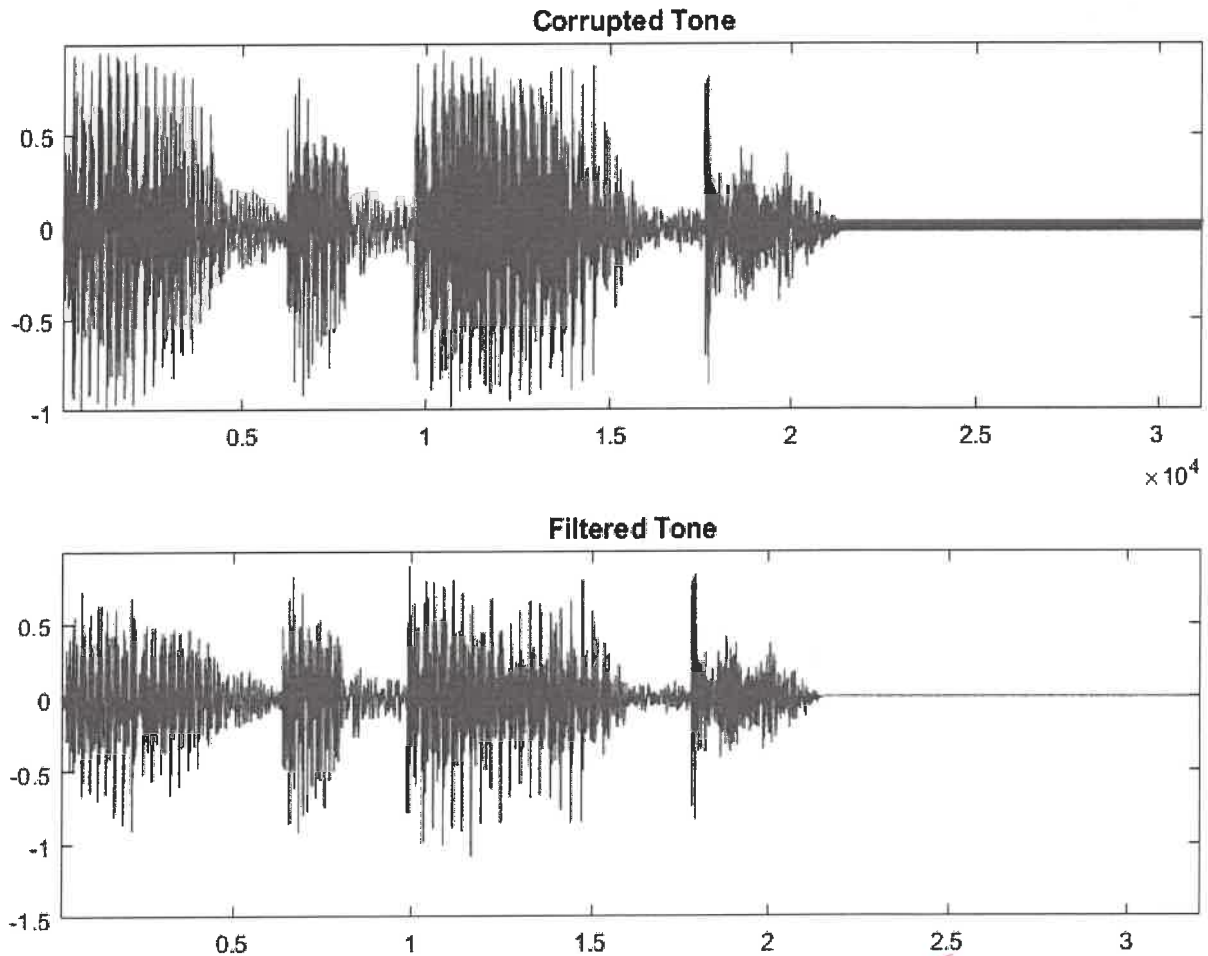


Figure 2: Comparison of the original and filtered signals

Filter magnitude and phase response

Figure 3 shows the magnitude of the filtering effect on the normalized frequency response. The point relating to the 780Hz tone is attenuated by the required amount and the sloping shoulders on the transition bands demonstrates the filter having a wider effect on the signal frequencies. The phase response is still linear except for a small hitch at the filtered frequency, which has the effect of slightly retarding the phase angle of the frequencies above the stop band.

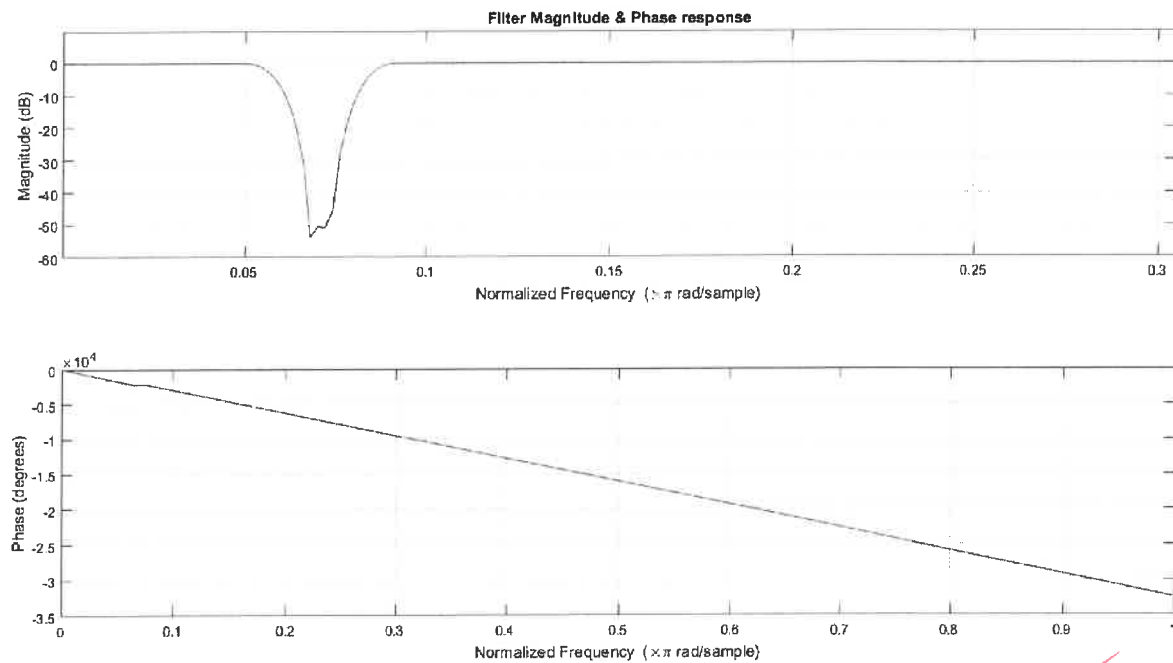


Figure 3: Filter magnitude and phase response

Fast Fourier transform - filtered signal

Running the fast Fourier transform again now shows (Fig. 4) the removal of the 780Hz frequency (and those between 750Hz and 810Hz), and reduction in frequencies in the transition bands either side. ✓

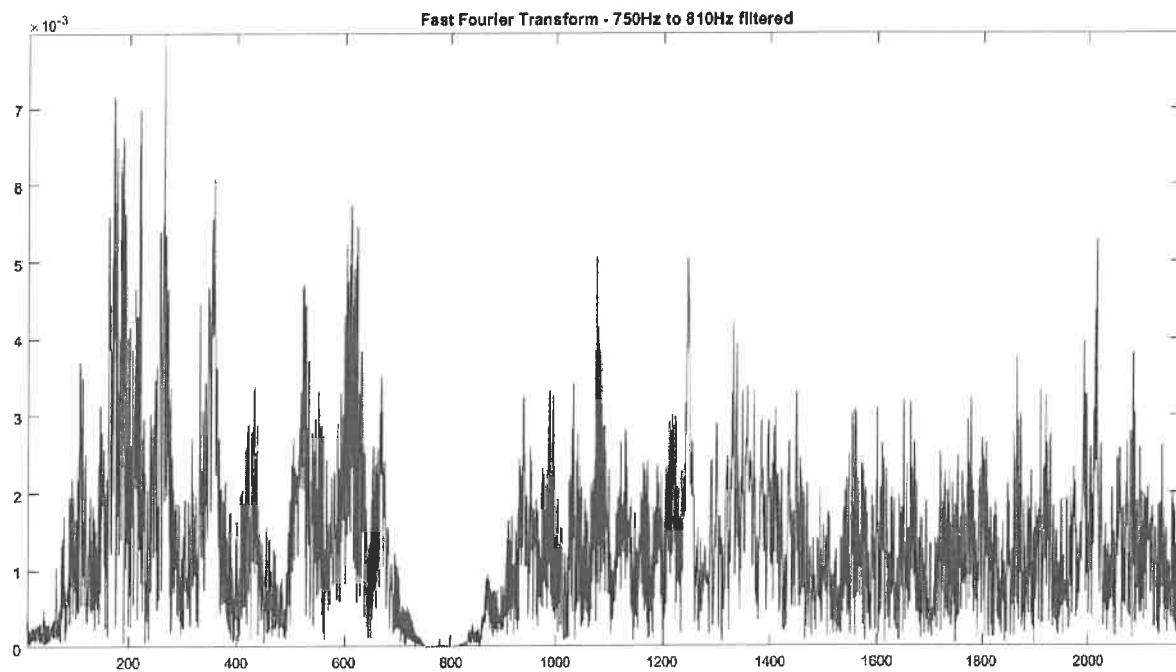


Figure 4: FFT of our filtered signal

Signal loss

By aligning the original and filtered signals¹ we can subtract the difference – giving us the signal that has been lost as a result of applying the filter. Playing back this lost signal allows us to qualitatively assess the degree to which our filter has removed aspects of the original signal we might wish to preserve, and Figure 5 allows us to inspect the waveform to see that not only the 780Hz tone has been removed. This is where the trade-off between filter complexity and audio losses becomes apparent – how much of the original signal are we willing to sacrifice and how much are we willing to invest in complicated signal processing?

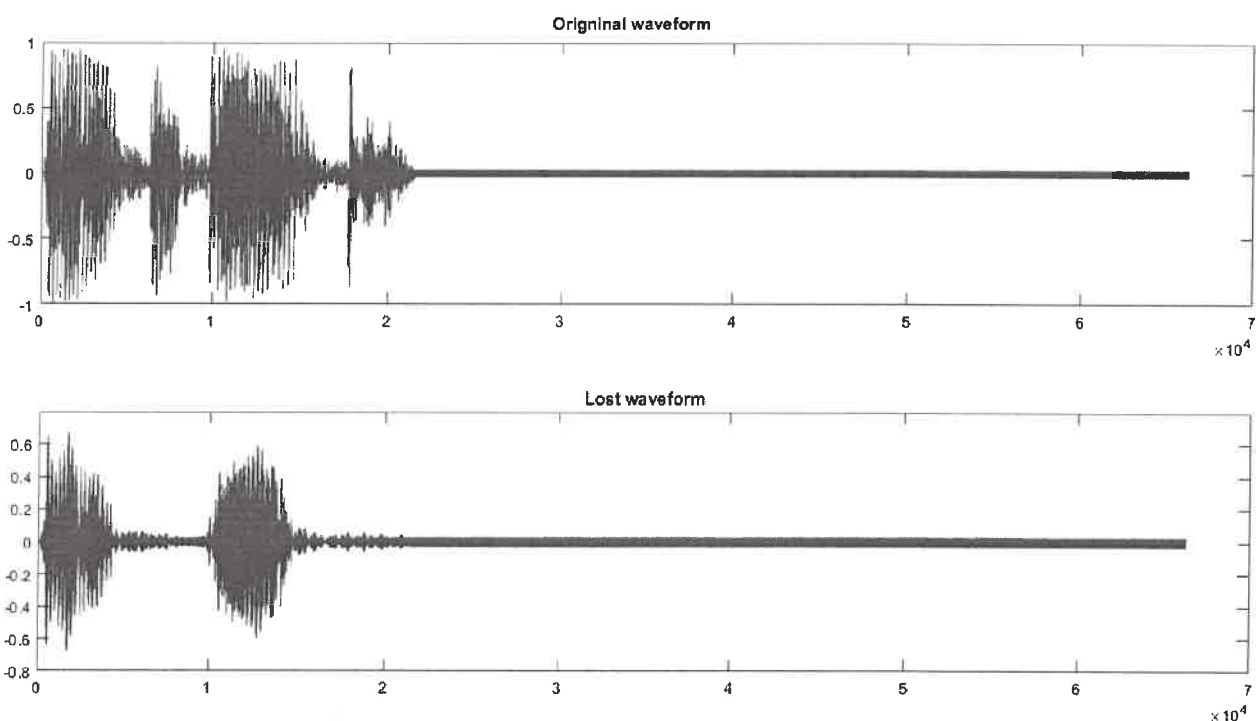


Figure 5: Lost audio signal due to filtering

¹ The two files are not in alignment as the filtered version has a delay at the beginning and is subsequently slightly larger. To ensure the files are the same size for alignment we pad out one of the files with zeros.

Filter parameter testing

A range of filter parameters were tested in order to get a picture of how the complexity of the filter impacted on the quality of the final signal. Table 1 shows the additional filter parameters that were tested.

Parameters	Low cut off	High cut off	Transition band	Taps
Narrow TB	700	860	50	728
Wide TB	600	960	150	243
Tight	750	810	20	1819

Table 1: Additional test parameters

Conclusions

From a qualitative point of view, all of the conditions above created a clear, acceptable filtered sound, and to my ear, there was nothing to separate them. Listening to the lost waveforms, there was a noticeable difference – the wide filter producing the most clearly definable sound, and the tight filter producing a much more muted sound.

It is apparent that the complexity of the filter using these parameters does not have a qualitative effect on the final sound in this case, making it impossible to recommend producing a filter with a narrow/tight stop band in this case.

FIR Filter Code for Matlab

```
path(path, 'Z:\DSP') % Path to files
[A,SR]=audioread('ATla.wav'); % Command to add file to Mlab
(naming it A, with a sample rate named SR)
sound(A,SR) % Play sound

% Use Fast Fourier Transform to analyse a mixed/corrupt signal
L=length(A); % Length of longest file
p=fft(A); % Fast fourier transform on file A
P2=abs(p/L); % Absolute values for all the
values of p
P1=P2(1:L/2+1);
P1(2:end-1)=2*P1(2:end-1);
f=SR*(0:(L/2))/L;
plot(f,P1) % Plot the frequency spectrum of
the FFT
title('Fast Fourier Transform')
pause

% Define the filter and variables to remove the corrupting variable
% Corrupting frequency identified as 780hz
fs=SR; % Sampling Frequency
fn=fs/2; % Nyquist Frequency
flowcut=650; % Low frequency cutoff (650Hz)
fhighcut=910; % High frequency cutoff (910Hz)
ft=100; % Transition band
Flowcut=flowcut/fn; % Normalised frequency low
Fhighcut=fhighcut/fn; % Normalised frequency high
Ft=ft/fn; % Normalised transition band
Ap=0.1; % Pass band gain in dB (the ripple) ✓
AS=55; % Stop band attenuation in dB
N=3.3/Ft; % Find number of taps (Hanning)
NR=round(N); % Round taps to whole number
a=fir1(NR, [Flowcut, Fhighcut], 'stop'); % Get filter multipliers (for each
tap), stop band filter
Afilt=filter(a,1,A); % Apply filter to the signal

% Plot our unfiltered and filtered tones
subplot(2,1,1); % Here we are setting up 2 plots,
this is the first
x=linspace(0,10); % A command for helping plot the
graph
plot(A) % Plot the original combined tones
title('Corrupted Tone')
subplot(2,1,2);
plot(Afilt) % Plot the filtered tones
title('Filtered Tone')

% Play filtered sound
pause
sound(Afilt,SR)

% Plot the magnitude and phase of the filter
freqz(a,1,500)
plot(freqz) % plot the magnitude and phase of
the filter
title('Filter Magnitude and Phase')
pause
```

```

% Use Fast Fourier Transform to analyse the filtered signal
L=length(Afilt); % Length of longest file
p=fft(Afilt); % Fast fourier transform on file
Afilt
P2=abs(p/L); % Absolute values for all the
values of p
P1=P2(1:L/2+1);
P1(2:end-1)=2*P1(2:end-1);
f=SR*(0:(L/2))/L;
plot(f,P1) % Plot the frequency spectrum of
the FFT
title('Fast Fourier Transform - 750Hz to 810Hz filtered')

% Export filtered file
filename='ATlafiltered.wav';
audiowrite(filename,A,SR);

% Now we combine the filtered with the original using the following
% Load in the original file
path(path,'Z:\DSP') % Path to files
[Orig,SR]=audioread('Arnie.wav'); % Add file to Mlab (naming it Orig,
with a sample rate named SR)

% Pad out the original file according to the delay
for z=length(Orig):length(Afilt) % Making Orig the same length as Afilt
    Orig(z)=0; % Fills the gap with zeros
end

% Align signals, auto calculates the delay, gives new names [new name, new
name]
[AfiltC,OrigC]=alignsignals(Afilt,Orig);

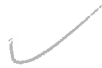
% Pad out filtered file according to the delay
for zz=length(AfiltC):length(OrigC) % Making AfiltC the same length as
OrigC
    AfiltC(zz)=0; % Fills the gap with zeros
end

% Determine the difference between the samples in each file
Difference=AfiltC-OrigC;

% Play difference
pause
sound(Difference,SR)

% Export difference file
filename='Difference.wav';
audiowrite(filename,A,SR);

```



HND Electrical and Electronic Engineering

Module: Digital Signal Processing Task 2

Lecturer: Linton Cook

Student Name: Gauthain Jay Chartier-Roberts

Student ID: CHA16832389

Contents

Task 2 – QPSK transmissions.....	3
The process	3
Waveforms	4
Results	7
Analysis & Conclusions.....	8
Matlab code	9
Works Cited.....	13

Task 2 – QPSK transmissions

For this task we shall use Matlab to simulate the performance of a QPSK demodulator when receiving a modulated QPSK signal combined with varying levels of randomly generated noise.

Quadrature Phase Shift Keying is a signal transmission method which utilises four carrier signals (of the same amplitude) to carry 8 bits of information (in pairs). However, transmitted signals are subject to various conditions which cause the received signal to differ (Desai & Kulkarni, 2016), and in digital transmissions the quality of the received signal is determined by the bit error rate (BER). The presence of noise contaminates the original signal so that the received signal differs, and for our purposes, white Gaussian noise will be added to our signal at varying levels in order to assess the quality of our QPSK transmission.

The process

Generate a tone in Matlab which we can then sample at a certain rate and feed into the QPSK modulator. A simulated white Gaussian noise signal can then be generated and combined with the modulated signal. The combined signal is processed by the demodulator and any errors can be highlighted. By altering the level of noise we can ascertain the acceptable error limits for QPSK transmission. Our bit error rate (BER) is the number of bits which have been altered from the original signal (808 bits) as a percentage.

The noise signal is composed of two separate noise signals one with positive components and one with negative components. The randomly generated numbers have values between 0 and 1, and are then multiplied to alter the level of noise. This combining of equal magnitude positive and negative amplitudes gives us our Gaussian distribution with equal power density at all frequencies – hence our additive white Gaussian noise.

For each noise level tested, the number of errors was calculated 3 times and then averaged before producing a percentage bit error rate.

Waveforms

Figure 1 shows a section of the original signal in binary form.

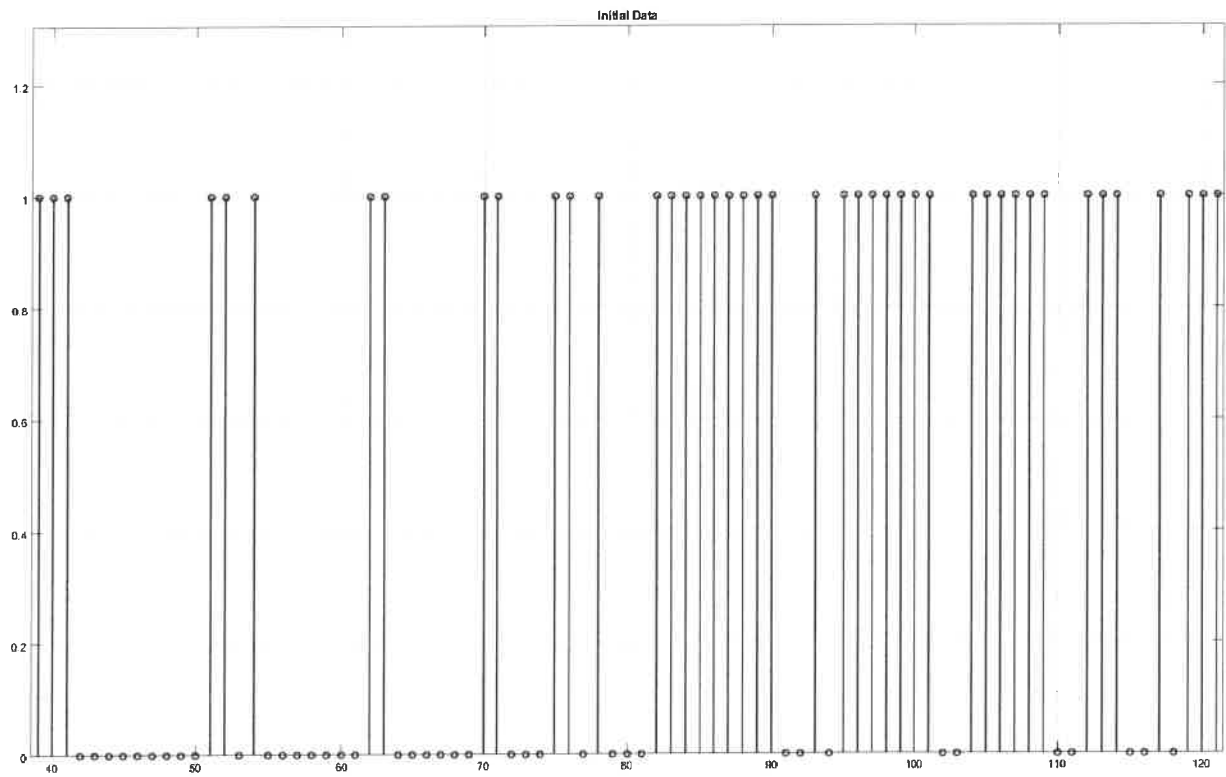


Figure 1: Original signal in binary form

Figure 2 shows the components created for our QPSK signal. We have our In-phase component, our quadrature phase component and their combined QPSK signal. The noise component is then added to give the final waveform, which is then demodulated to recover the initial data.

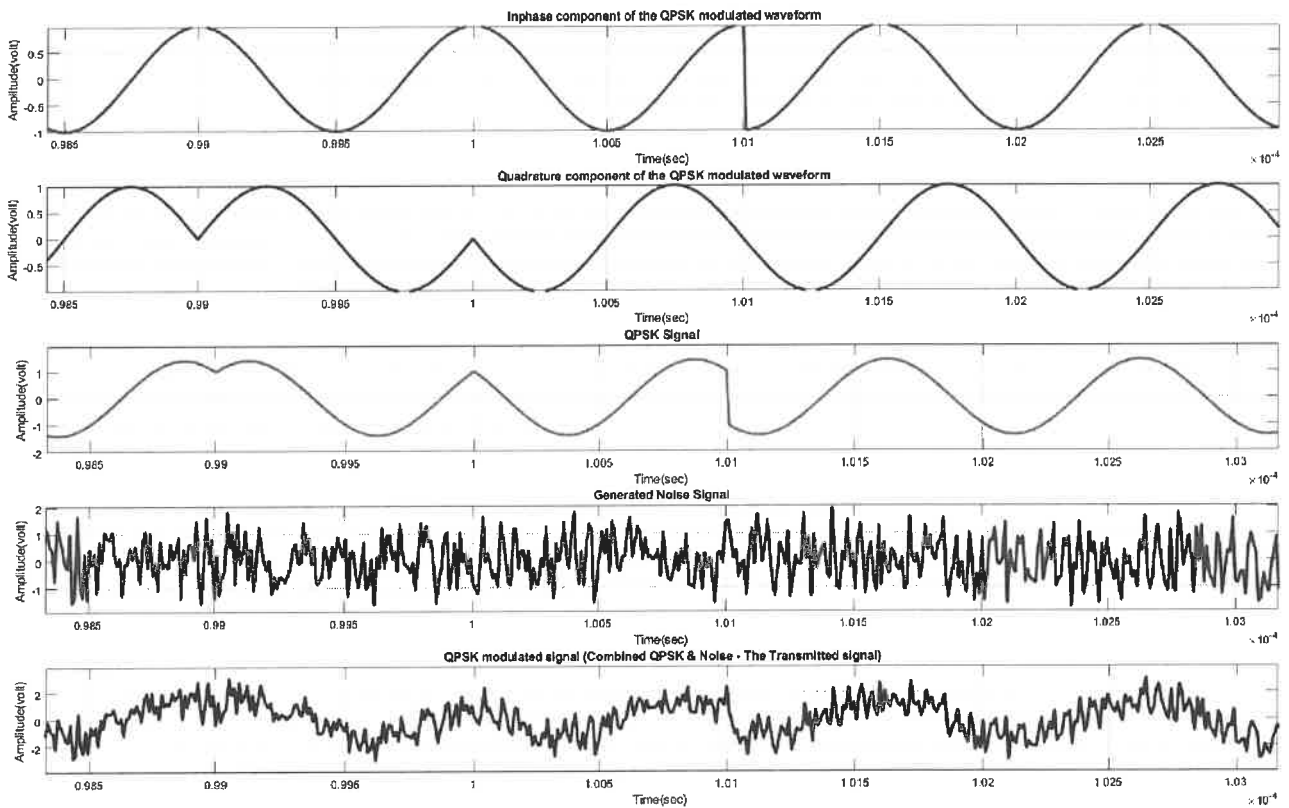


Figure 2: QPSK signal components

Figure 3 shows a section of the demodulated signal, and it is this that is compared with the initial data to determine whether errors have been introduced.

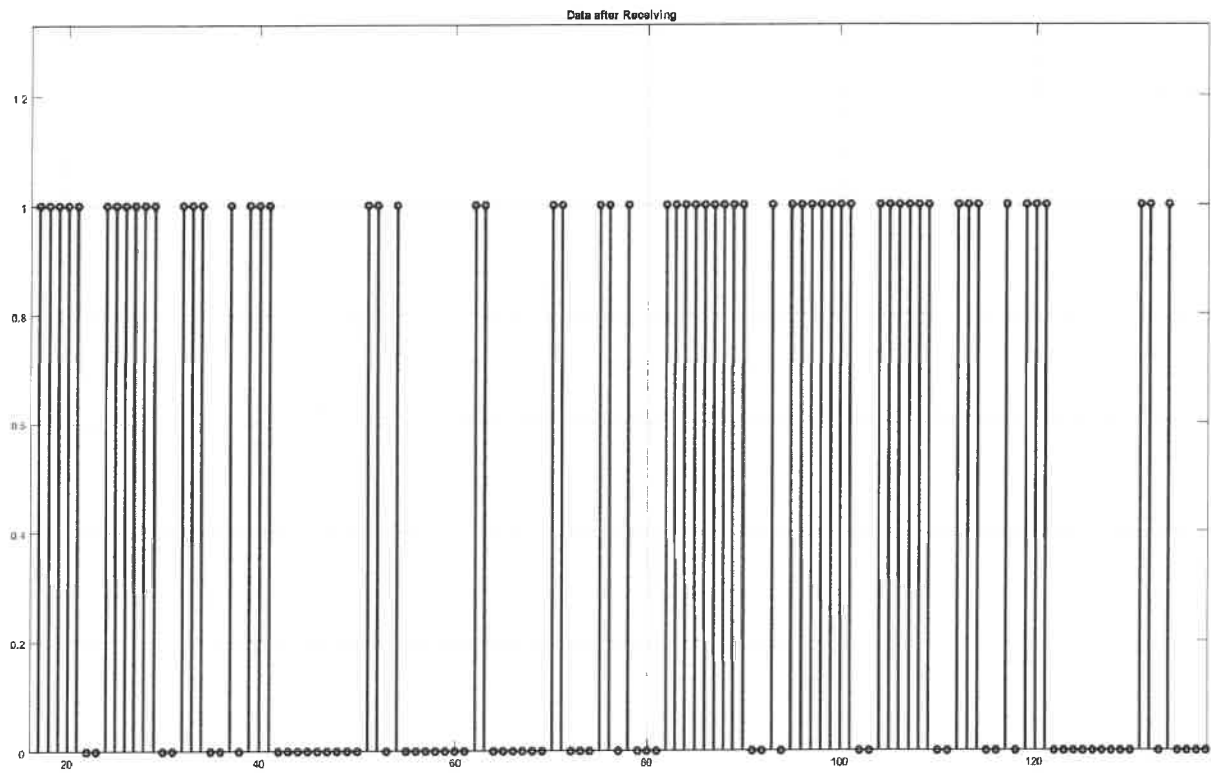


Figure 3: Received data signal

Results

Noise Signal 1 level	Noise Signal 2 level	Total Errors Trial 1	Total Errors Trial 2	Total Errors Trial 3	Average Bit Error Rate (%)
2	-2	0	0	0	0.00
3	-3	0	0	0	0.00
5	-5	0	2	1	0.12
7	-7	5	9	4	0.74
8	-8	14	12	10	1.49
10	-10	31	42	34	4.41
13	-13	71	92	66	9.45
16	-16	122	111	119	14.52
20	-20	159	165	164	20.13
24	-24	195	181	187	23.23
28	-28	234	219	225	27.97
32	-32	236	251	255	30.61
35	-35	245	278	253	32.01
42	-42	273	282	276	34.28
50	-50	294	285	310	36.67
57	-57	321	326	314	39.65
65	-65	339	337	343	42.04
72	-72	342	327	338	41.54
80	-80	375	335	336	43.15
87	-87	345	350	356	43.36
95	-95	351	371	336	43.65
120	-120	355	359	363	44.43

Figure 4: Table of Results

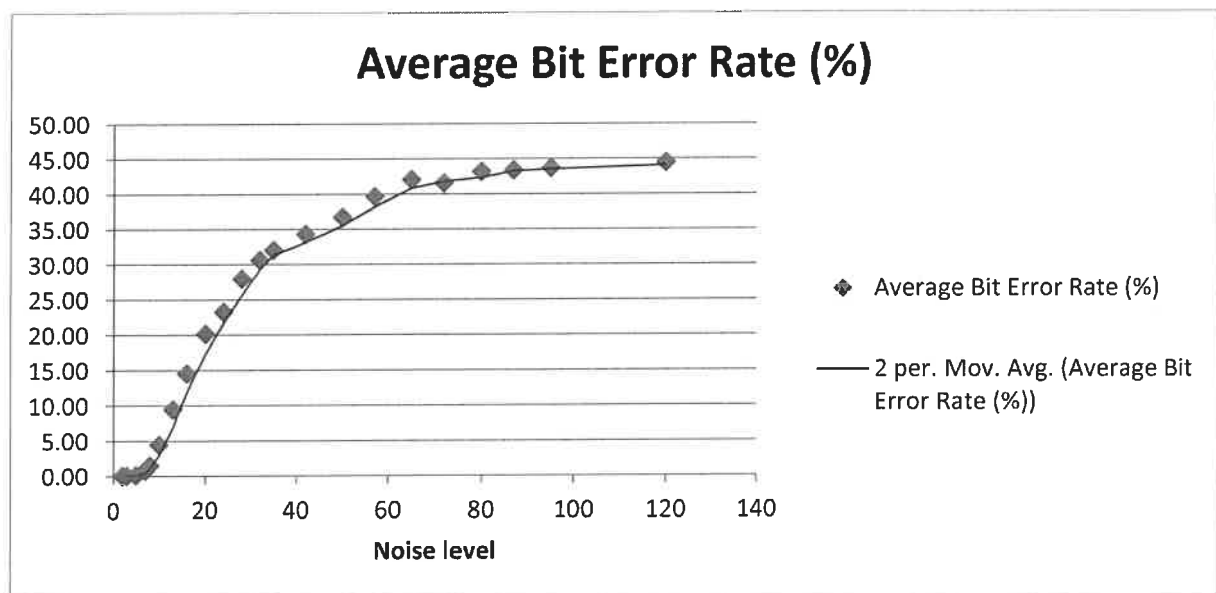


Figure 5: Graph of average BER with increasing noise level

Analysis & Conclusions

The results demonstrate a roughly sigmoid response to increasing noise level, reaching a plateau at approximately 45%. By increasing the signal transmission power we could decrease the effect of noise, effectively shifting the curve to the right. We could also introduce error-correction factors such as a Hamming code to reduce the BER. Another option would be to apply filters to the signal to deal with known sources/frequencies of noise.

We could improve the interpretation of the BER by determining the signal energy per bit to noise power density ratio (given by E_b/N_0). When noise is present there is a limit to the bit rate that a channel can carry, so with some additional information such as the channel bandwidth, signal power, noise power we can also analyse channel capacity in the presence of noise.

In reality, signals are subject to a whole manner of different factors which introduce “noise” of varying types (Gaussian/Rician/Rayleigh). These can be deterministic to a degree (so we can filter them) or random, so modeling can help us predict if and how signals can be received in an acceptable condition.

Matlab code

```
% Generate a 1000Hz signal, set up our parameters
frequency = 1000;           % Tone frequency
period = 1/frequency;       % The period
cycles = 10;                % Required cycles
t=0:(period/10):(cycles*period); % Sample rate set to 10x
                             % frequency
wave = sin(2*pi*frequency*t);
A = wave+1;                  % Offset the wave giving
                             % all positive values for A
sigma_max = 2;               % The maximum value of
                             % the sin wave

% Generate the data sample
AtoD = 8;                    % Set 8 bit quantizer
numberofvalues = 2^AtoD;     % Sets 256 values
samplevalues = 2/numberofvalues; % Sample step size
samples = fix(A/samplevalues); % Setting number of
samples and rounding
samplelength = length(samples); % Setting sample length
data1 = strings([1,samplelength]); % Create the data1 array
for sample = 1:samplelength
    data1(sample) = dec2bin(samples(sample),8); % Select samples,
    convert to 8 bit binary
end

% We basically need our data to be one long single string with no
spaces

datacomb=join(data1);        % Combining data1
strings into one
dataconv=char(datacomb);     % Converting data1 into
characters

% Now we juggle the character data to space it
nd1 = regexp(dataconv,'0','0 '); % Spacing the zeros
nd2 = regexp(nd1,'1','1 ');      % Spacing the ones
nd3 = regexp(nd2,' ',' ');       % Removing any extra
spaces to leave the whole character string in one cell
[separate,remain] = strtok(nd3); % Parsing the character
string using whitespaces to delimit

% Set up a new array which separates out the individual bits into
cells

for i=1:(sample*AtoD)
```

```

    senddata(i) = str2num(separate);           % Convert the characters
into a numeric array
    [separate,remain] = strtok(remain);

end

% Plot our sample data that we intend to transmit
figure(1)
stem(senddata, 'linewidth',2), grid on;
title('Initial Data');
axis([ 0 (length(senddata)) 0 1.5]);

% Manipulate our senddata for non-return-to-zero and reshape the
array
NRZdata=2*senddata-1;                        % Converts all
zeros to -1 for QPSK modulation
sp_data=reshape(NRZdata,2,length(senddata)/2); % S/P conversion
of data

% The 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(senddata)/2
    A=sp_data(1,i)*cos(2*pi*f*t);          % Inphase component
    y2=sp_data(2,i)*sin(2*pi*f*t) ;       % Quadrature component
    y_in=[y_in A];                         % Inphase signal vector
    y_qd=[y_qd y2];                       % Quadrature signal vector
    y=[y A+y2];                            % Modulated signal vector
end

% Create and add white Gaussian noise - change these parameters for
testing
rand_noise1 = rand(1,length(y))*2;         % Generate random values
between 0 and 1 and multiply
rand_noise2 = rand(1,length(y))*-2;        % Generate random values
between 0 and 1 and multiply
rand_noise = rand_noise1+rand_noise2;      % Combine noise values
Tx_signal=(y+rand_noise);                  % Transmitted signal =
modulated signal plus noise
tt=T/99:T/99:(T*length(senddata))/2;

figure(2)

subplot(5,1,1);

```

```

plot(tt,y_in,'m','linewidth',3), grid on;
title('Inphase component of the QPSK modulated waveform');
xlabel('Time(sec)');
ylabel('Amplitude(volt)');

subplot(5,1,2);
plot(tt,y_qd,'linewidth',3), grid on;
title('Quadrature component of the QPSK modulated waveform');
xlabel('Time(sec)');
ylabel('Amplitude(volt)');

subplot(5,1,3);
plot(tt,y,'g','linewidth',3), grid on;
title('QPSK Signal');
xlabel('Time(sec)');
ylabel('Amplitude(volt)');

subplot(5,1,4);
plot(tt,rand_noise,'r','linewidth',3), grid on;
title('Generated Noise Signal');
xlabel('Time(sec)');
ylabel('Amplitude(volt)');

subplot(5,1,5);
plot(tt,Tx_signal,'k','linewidth',3), grid on;
title('QPSK modulated signal (Combined QPSK & Noise - The Transmitted signal)');
xlabel('Time(sec)');
ylabel('Amplitude(volt)');

% The QPSK Demodulation
Rx_data=[];
Rx_signal=Tx_signal; % Received signal

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

    % Inphase coherent detector
    Z_in=Rx_signal((i-1)*length(t)+1:i*length(t)).*cos(2*pi*f*t); %
multiply received & inphase carried signal
    Z_in_intg=(trapz(t,Z_in))*(2/T); %
integration using trapizodial rule
    if(Z_in_intg>0)
        Rx_in_data=1;
    else
        Rx_in_data=0;
    end

    % QPSK detector
    Z_qd=Rx_signal((i-1)*length(t)+1:i*length(t)).*sin(2*pi*f*t); %
multiply received & quadrature carried signal

```

```

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

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

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

data_diff=Rx_data - senddata; % The difference between
sent and received data
data_errors=num2str(data_diff); % Convert the difference
into a character string
error_count=count(data_errors,'0'); % Looking for zeros in
the difference
Total_errors=length(data_diff)-error_count; % The final number of
errors

```

Works Cited

Desai, J. A. & Kulkarni, S. N., 2016. Effect of Noise on BER of BPSK, QPSK, DPSK, and QAM Modulation Techniques. *International Journal of Research and Scientific Innovation (IJRSI)*, July, III(VII), pp. 71-74.