

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.		
<i>I can see evidence of reading in task 2. More background for task 1 would have been welcome</i>	20	10
2. Knowledge and Understanding of Subject Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.		
<i>You show a good understanding of the basic principles. I am a little concerned that some of your comments are restricted to explaining lines of code rather than concepts.</i>	25	15
3. Analysis Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence		
<i>More analysis in task 1 and a much larger, clearer caption in Task 2 was required.</i>	25	12
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>You have managed to use Matlab. You could have explored more of its capabilities to analyse your data.</i>	25	15
5. Skills for Professional Practice Attributes in professional practice: individual and collaborative working; deployment of appropriate media; presentation and organisation.		
<i>A Quite good effort - a section on conclusions - recommendations would have been useful.</i>	5	3

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)		55. %
	Up to one-week late	Over one week late	

HND Electrical and Electronic Engineering

DSP, Task 1

Lecturer: Linton Cook

Student Name: Emanuele Bonagura

Student ID: BO16830111

TASK 1 30/50
TASK 2 25/50
55 TOTAL

The aim of this task is to demonstrate an ability to apply filtering technique to a corrupted audio file, "DSPassignmenttask1".

The file contains a recording of spoken dialogue which has been corrupted with a tone.

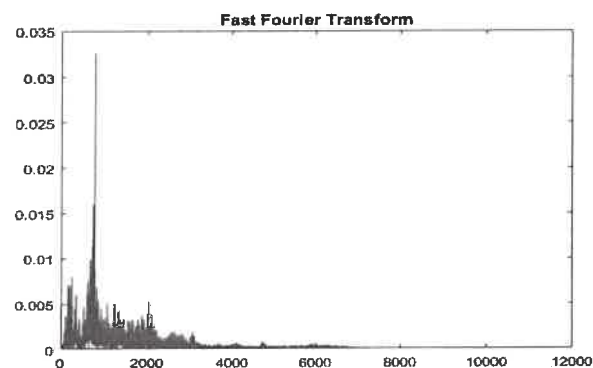
First the signal has been downloaded into MATLAB using the command `audioread`. The name has then been changed from "DSPassignmenttask1" to "AT1a"

```
path(path, 'Z:\DSP') % Path to files
[A,SR]=audioread('AT1a.wav'); % Command to add file to
Matlab (naming it A, with a sample rate named SR)
sound(A,SR) % Play sound
```

To figure out the tone frequency, a DFT (Discrete Fourier Transform) has been performed using the FFT (Fast Fourier Transform) command.

```
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
```

As showed from the figure below, a strong response has been identified at 780 Hz. ✓



To remove the tone, a stop-band filter has been created, using a Hamming window function.

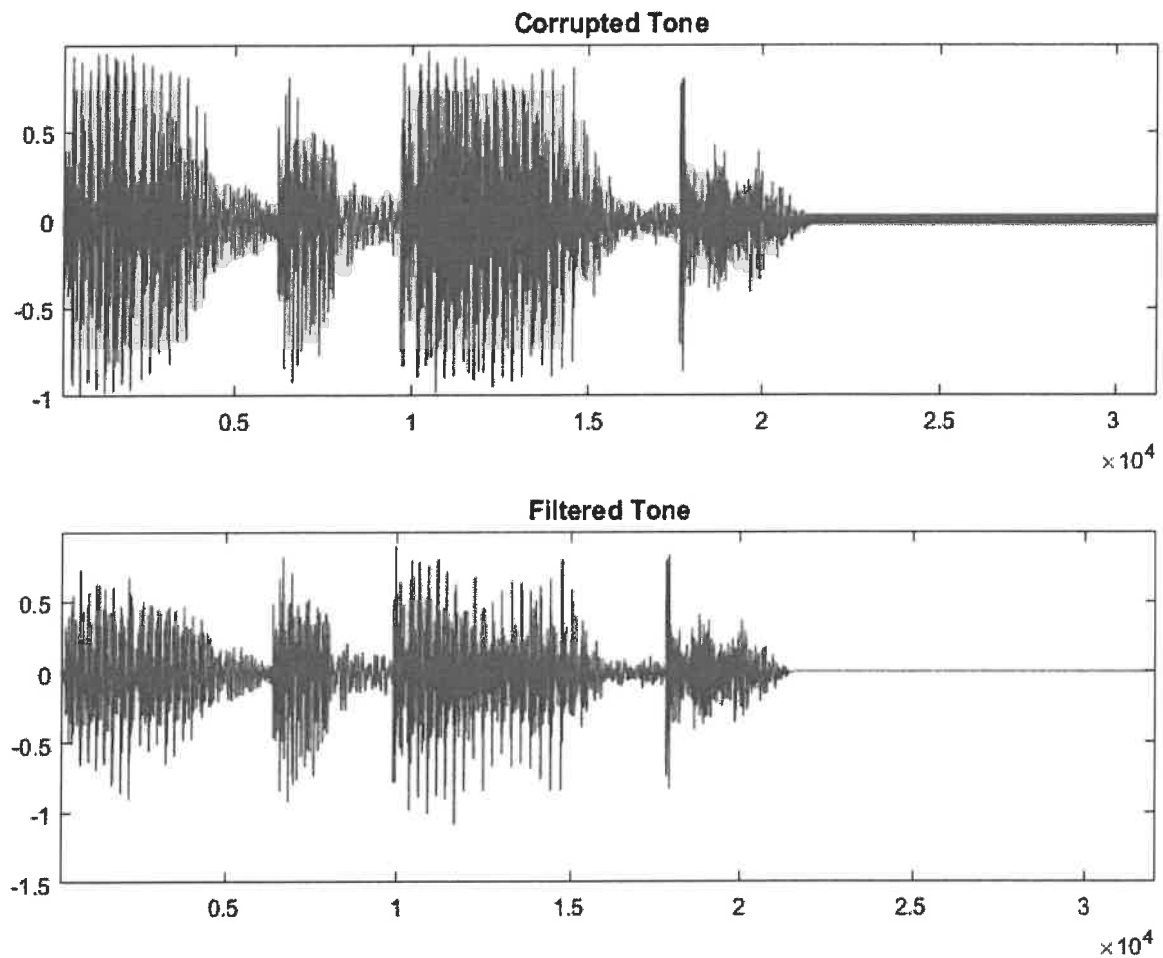
The FIR Filter parameters are:

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

```
fs=SR; % Sampling Frequency
fn=fs/2; % Nyquist Frequency
flowcut=650; % Low frequency cutoff
fhighcut=910; % High frequency cutoff
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 (Hamming)
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
```

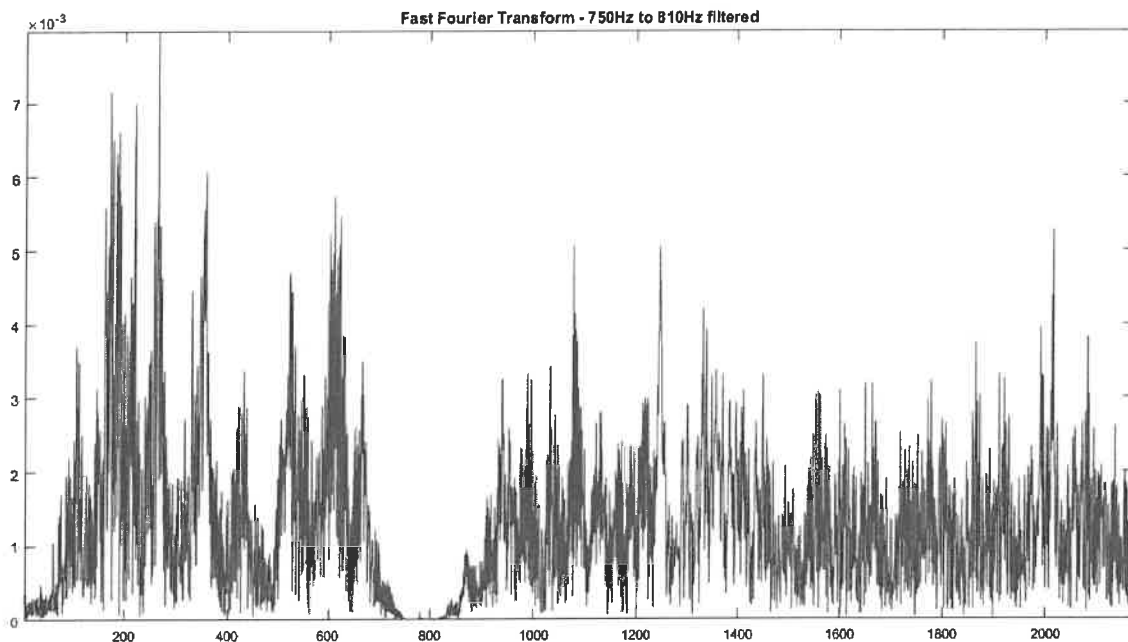
Then, the filtered file has been compared with the corrupted one.

```
% 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')
```



The filtered signal has been analyzed using a Fast Fourier Transform.

```
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')
```



Comparing the original and filtered waveform signals we can outline the signal that has been lost as a result of applying the filter.

```
% 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)

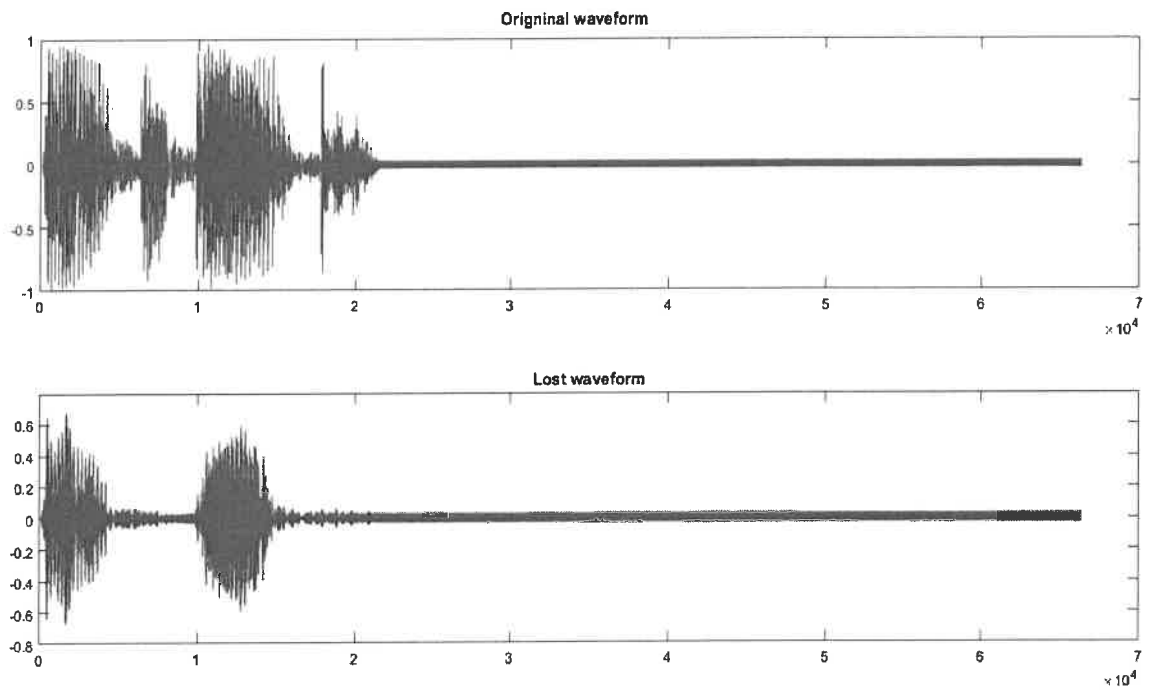
% Pad out the original file according to the delay
for z=length(Orig):length(Afilt) % Making Orig the same length as
    Afilt
    Orig(z)=0; % Fills the gap with zeros
end

% 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)
```



To determine the minimum size filter that can be applied such that the corruption is removed to a level that renders it inaudible different filter parameters have been tested.

```
% Play the original signal
hplayer = audioplayer(y, Fs);
play(hplayer);

% Play the noise-reduced signal
hplayer = audioplayer(yf, Fs);
play(hplayer);
```

Listening to the signal before and after processing we noted that from a qualitative point of view, all of the conditions generated an acceptable filtered sound.

References

References: J.G. Proakis and D. G. Manolakis, "Digital Signal Processing. Principles, Algorithms, and Applications", Prentice-Hall, 1996.

S. J. Orfanidis, "Introduction To Signal Processing", Prentice-Hall, 1996.

www.mathworks.com

HND Electrical and Electronic Engineering

DSP, Task 2

Lecturer: Linton Cook

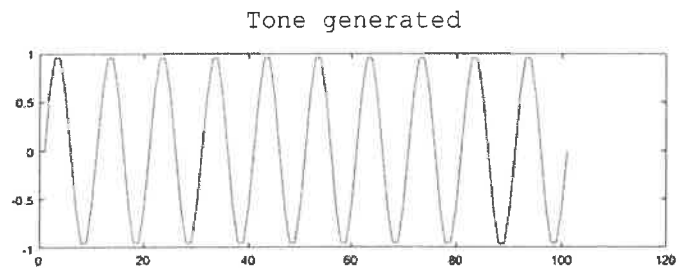
Student Name: Emanuele Bonagura

Student ID: BO16830111

a) Generate a 1 KHz tone sampled at 10 KHz with a duration of 10 cycles.

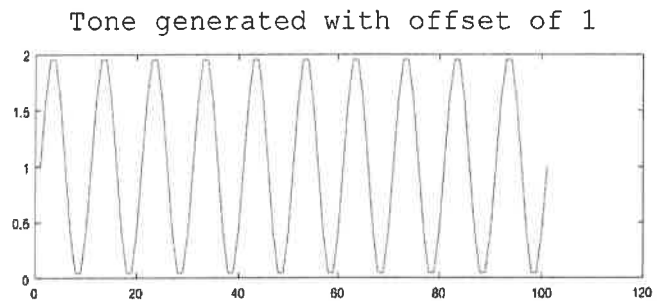
Using the function “clear all” all the existing data and workspace has been clean. Then, a 1000Hz tone with a duration of 10 cycles has been created. At the end, the tone has been sampled at 10KHz

```
clc;  
clear all;  
frequency = 1000;  
period = 1/frequency;  
cycles = 10;  
t=[0:(period/10):(cycle  
s*period)];  
yg =
```



Then an offset of 1 has been added to the sin wave so that all values are positive, and the maximum value of the sin wave is 2.

```
y1 = yg+1;  
sigmax = 2;
```



Then an 8 bits quantizer, with nvals of 2^8 , has been generated, space delimiters have been inserted and extra space removed.

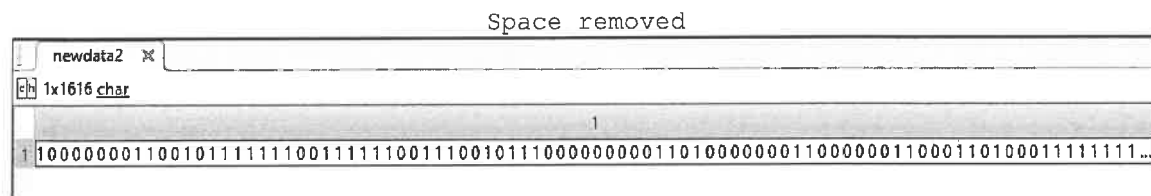
```
AtoD = 8;
nvals = 2^AtoD;
samplevals = 2/nvals; % this will then be the step size Note 2 is
the max valuenumErrors
sampls = fix(y1/samplevals);
samplelen = length(sampls);
datstr = strings([1,samplelen]);
for (getsample = 1:samplelen);
datstr(getsample) = dec2bin(sampls(getsample),8);
end
datain11=join(datstr);
;
datain=char(datain11);
;
newdata1 = regexprep(datain,'0','0 ');
newdata1a = regexprep(newdata1,'1','1 ');
newdata2 = regexprep(newdata1a,' ',' ');
[sepbit,rembits] = strtok(newdata2);
for(i=1:(getsample*AtoD))
    bits(i) = str2num(sepbit);
    [sepbit,rembits] = strtok(rembits);
end
```

8 bits data

```
newdata1 x
1x1303 char
1 10000000 11001011 11111001 11111001 11001011 10000000 00110100 00000110 00000110 00110100 01111111 11001011 111110...
```

Space delimiters inserted

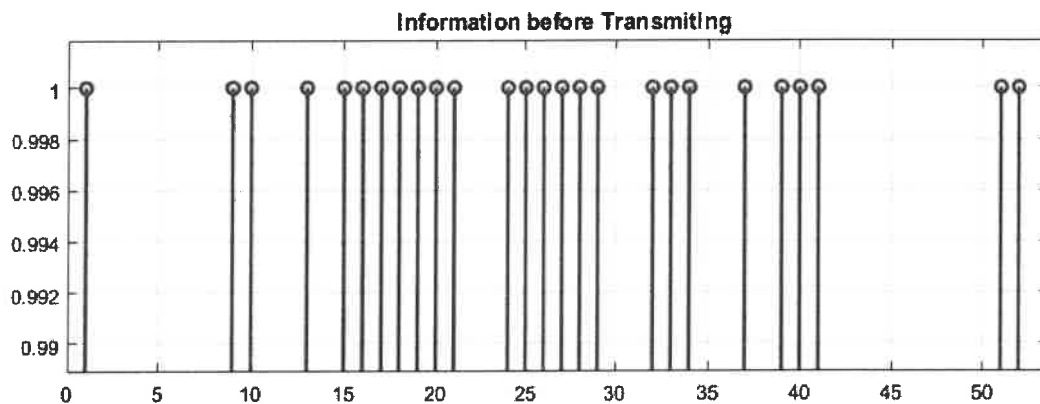
```
newdata1a x
1x1716 char
1 10000000 11001011 11111001 11111001 11001011 10000000 00110100 00000110 00000110 00110100 01111111 ...
```



At this point, a hamming code could be added.

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

"Information before transmitting" have been generated as showed in the figure below.



b) Deliver the sampled wave to the QPSK modulator in a suitable form.

The sampled wave, previously generated, has now been transformed in NZR form for QPSK (Quadrature phase shift keying) modulation.

QPSK modulation is a technique for achieving bandwidth reduction.

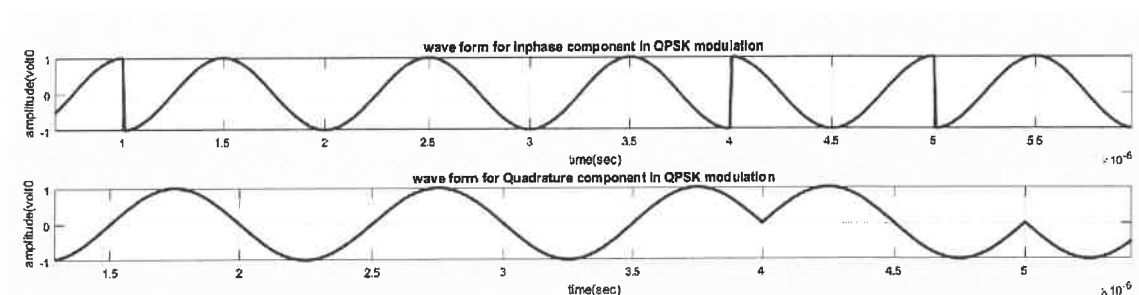
```
data_NZR=2*data-1;
s_p_data=reshape(data_NZR,2,length(data)/2);

% 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

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)');
```

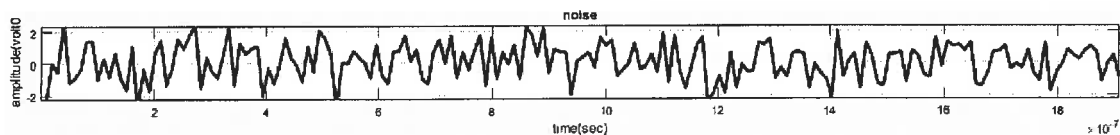
The QPSK modulator has a carrier frequency of 10^6 .



c) Generate an interfering 'random' noise signal (simulated white Gaussian).

```
rnoise1 = rand(1,length(y))*3;  
rnoise2 = rand(1,length(y))*-3;  
rnoise = rnoise1+rnoise2;  
  
subplot(5,1,4);  
plot(tt,rnoise,'r','linewidth',3), grid on;  
title('noise');  
xlabel('time(sec)');  
ylabel(' amplitude(volt0)');
```

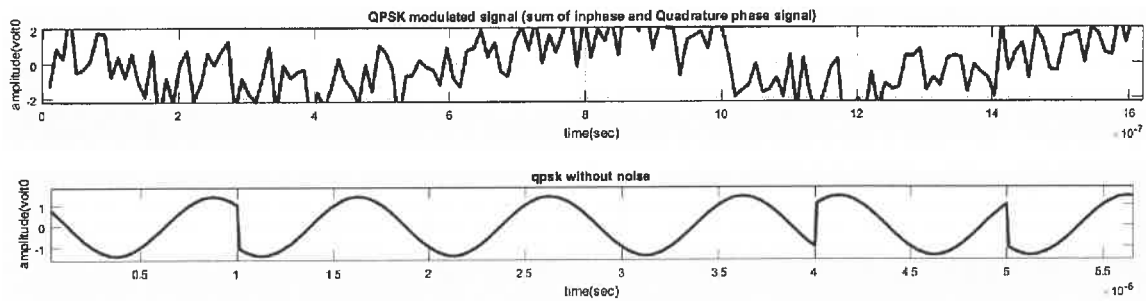
Two separate random noise signals have been generated. A final noise has been obtained adding noise 1 and noise 2.



d) Combine the noise signal with the modulated QPSK signal and feed to the QPSK demodulator.

```
Tx_sig=(y+rnoise); % transmitting signal after modulation  
tt=T/99:T/99:(T*length(data))/2;  
  
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(5,1,5);  
plot(tt,y,'r','linewidth',3), grid on;  
title('qpsk without noise');  
xlabel('time(sec)');  
ylabel(' amplitude(volt0)');
```

At this point, the final noise signal has been combined with the modulated QPSK signal. The QPSK signal is the sum "inphase" and "quadrature phase" signal obtained previously.



e) Examine the output of the QPSK demodulator and detect any errors in transmission.

```

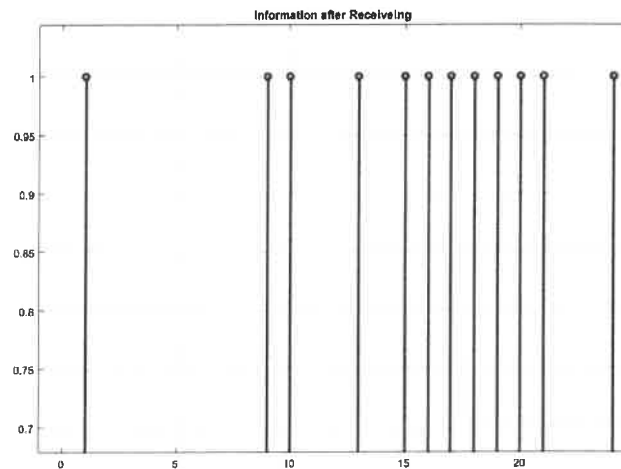
% QPSK Demodulation
Rx_data=[];
Rx_sig=Tx_sig; % Received signal
for(i=1:1:length(data)/2)
    %%XXXXXXX inphase coherent dector XXXXXXXX
    Z_in=Rx_sig((i-1)*length(t)+1:i*length(t)).*cos(2*pi*f*t);
    % above line indicat multiplication of received & inphase
    carried signal

    Z_in_intg=(trapz(t,Z_in))*(2/T);% integration using trapezoidal
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 trapezoidal
rule
    if (Z_qd_intg>0)% Decision Maker
        Rx_qd_data=1;
    else
        Rx_qd_data=0;
    end
    Rx_data=[Rx_data Rx_in_data Rx_qd_data]; % Received Data
end
vector
End
figure(3)
stem(Rx_data,'linewidth',3)
title('Information after Receiveing ');
axis([ 0 (length(Rx_data)) 0 1.5]), grid on;

```



The errors have been analysed using the trapezoidal rule.

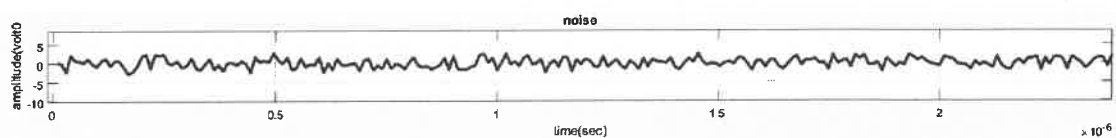
Every time that the program starts, the values of “Z_in_intg” and “Z_qd_intg” will change.

Z_in_intg	0.7547
Z_qd	1x99 double
Z_qd_intg	0.7887

Z_in_intg	1.1840
Z_qd	1x99 double
Z_qd_intg	1.0047

- f) Vary the level of noise signal generated in step C and determine the maximum level that would be allowed such that the data could be reliably transmitted.

```
noiseVariance = 10^(-snr/10);
noise = sqrt(noiseVariance)* randn( 1, length(Tx_Sig_fil));
plot(noise+Tx_Sig_fil);
```



g) Analysis of noise vs BER

The presence of noise degrades the signal quality. For the analog system the signal quality is measured by the parameter, signal-to-noise ratio (SNR), and for a digital system the signal quality is measured by the parameter, bit error ratio (BER).

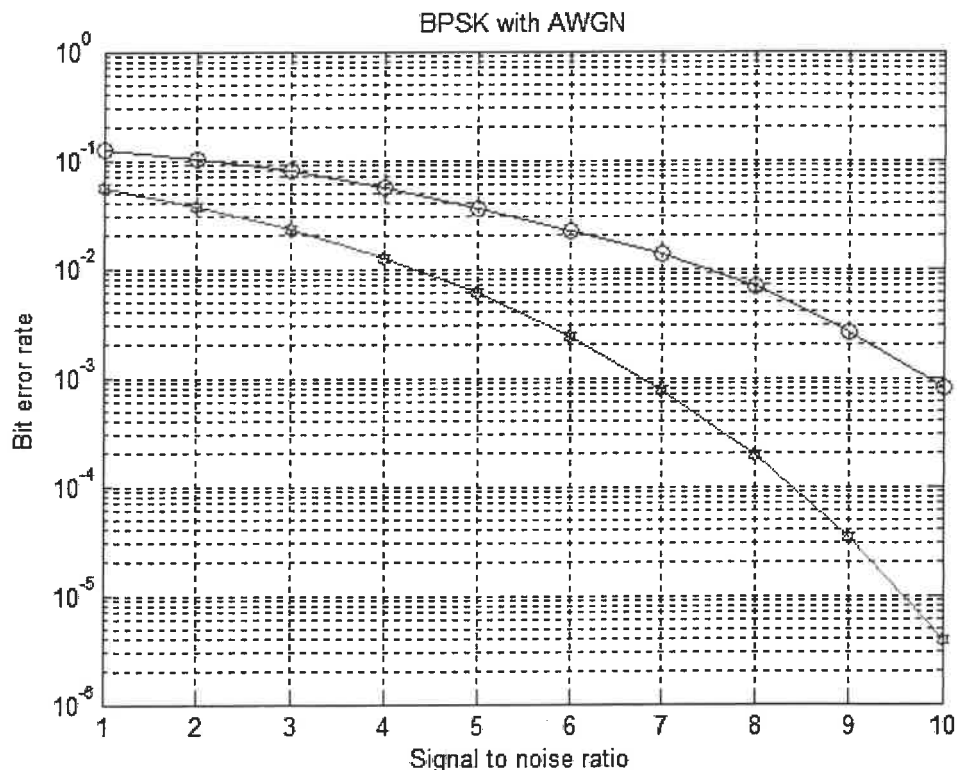
Bit Error Rate is a measure of signal to noise ratio in a digital modulation system. In order to calculate the BER, a known repeating pattern must be transmitted to the radio. The receiver must demodulate the data and compare it to the known data pattern and determine the number of bits that are errors. The BER is then the ratio of bits in error to total bits received.

Signal-to-noise ratio (abbreviated SNR or S/N) compares the level of a desired signal to the level of background noise.

SNR is defined as the ratio of signal power to the noise power, often expressed in decibels. A ratio higher than 1:1 (greater than 0 dB) indicates more signal than noise.

SINAD is a term used for the Signal to Noise and Distortion ratio and is a type of audio signal to noise ratio. In an analog FM system, demodulated audio signal to noise ratio is an indication of RF signal quality. In order to measure the audio signal to noise ratio, typically test equipment measures total audio power (Signal plus Noise plus Distortion) and then notch filters the audio signal tone (typically 1 kHz) and measures the audio power again (Noise plus Distortion) and takes the ratio in decibels.

$$SINAD(dB) = 10 * \log_{10} \left(\frac{Signal+Noise+Distortion}{Noise+Distortion} \right)$$



References

The MathWorks 1997, *MATLAB The Language of Technical Computing*

https://ub.cbm.uam.es/publications/teaching/master_biofisica_2011_2012/MATLAB_REFBOOK.pdf

Holly Moore, 2007 *MATLAB for Engineers*

Nicholas Higham, Desmond Higham, 2000 *MATLAB Guide*

www.mathworks.co.uk