Part II

Signals Circuits and Systems

Workshop 2: Signal Analysis in Frequency Domain

Objective: To analyze continuous-time and discrete-time signals in frequency domain.

Outcome: After successful completion of this session, the student would be able to

1. Analyze and find the frequency components in a continuous time periodic signal.
2. Synthesize a periodic signal using the Fourier series.
3. Identify the difference between ideal filters and actual filters.
4. Use filters in simple applications.

Equipment Required:

1. A personal computer.
2. Python with NumPy, SciPy, snd Matplotlib

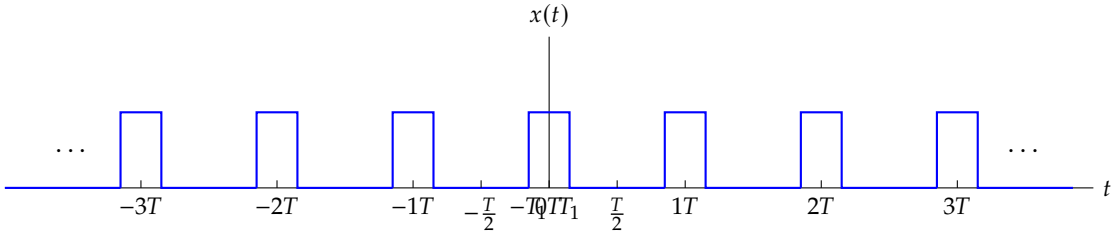
Components Required: None.

2.1 Fourier Series Approximation

The Fourier series analysis equation for a continuous time periodic signal $x(t)$, having the period T, angular frequency ω_0, $(= 2\pi/T)$ can be given as follows.

$$a_k = \frac{1}{T}\int_T x(t)e^{-jk\omega_0 t}\,dt \tag{2.1}$$

Find the Fourier Series coefficients for the square wave given in Figure 1. [Graded] Hint: Take $T_1 = \frac{T}{4}$ (Please note: Include an image of your work.)



Let's import required packages first.

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft,fftshift,ifft
from scipy import signal
```

Q: Taking $A = 1$ V, $T = 1s$ complete the function $a(k)$ to return the Fourier series coefficients of the square wave for given any integer value of k. [Graded]

The Fourier series synthesis equation is given as

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}. \tag{2.2}$$

The following is the equation for the Fourier series approximation of original periodic signal with N number of harmonics.

$$x(t) \simeq x_N(t) = \sum_{k=-N}^{N} a_k e^{jk\omega_0 t}. \tag{2.3}$$

```
# Square pulse
def square(t):
    if t % 1 < 0.25 or t % 1 > 0.75:
        s = 1
    elif t % 1 == 0.25 or t % 1 == 0.75:
        s = 0.5
    else:
        s = 0
    return s

# Fourier series coefficients
def a(k):
    # Your code goes here
    a_k = 1

    return a_k
```

Q: Complete the function fs_approx(t,N) to return the value of a Fourier series approximated periodic signal, at any given time. (Graded)

```
def fs_approx(t, N):
    # Your code goes here
    x_t = 0

    return x_t
```

Q: Update the python script according to the following guidelines. [Graded]

1. Create the array t with equally spaced 1000 elements in the interval $[-2.5, 2.5]$
2. Use the square(t) function to fill the array x with the values of square wave at each time instant in the array t.
3. Use the function fs_approx(t,N) to fill the array y with the function values of the Fourier series approximated square wave.

```
# Fourier series approximation of the square wave
x = []
y = []
N = 5  # CHANGE HERE

time = <————> # Your code goes here
for t in time:
    # Your code goes here
    <————————————
    ————————————>
```

Q: Write a Python script to plot the original signal, $x(t)$ and the approximated signal, $x_N(t)$ in the same figure for $N = 5$. [Graded] Q: Plot the original signal, $x(t)$ and the approximated signal, $x_N(t)$ in the same figure for $N = 50$. [Graded] Q: Comment on your observations. (i.e. for $N = 5$ and $N = 50$)

2.2 Fourier Series Coefficients [Graded]

Create the two arrays k and ak with integers in the interval $k = -20, \ldots, 20$ and the Fourier series coefficients of the square wave for each k value in the array, respectively. Use stem() function to plot the Fourier series coefficients against k. Q: Plot normalized Fast Fourier Transform (FFT) coefficients in X_{norm} vs k with stem() function. Use set_xlim() function to limit the x-axis to the interval $[-20, 20]$. [Graded]

```
N = 200
t = np.linspace(0, 1-1/N, N)
x = []
for i in t:
    x.append(square(i))

# Obtaining FFT coefficients
X = fftshift(fft(x))
X_norm = X.real/N
k = np.linspace(-N/2, N/2-1, N)

# plotting fft coefficients
# Your code goes here
<------------
------------->
```

Q: Comment on the observations from the above codes. [Graded]

2.3 Ideal Filters and Actual Filters

In this section, we will observe the filtering operation of ideal filters and actual filters by passing a waveform containing sinusoids with different frequencies through the filters.

Q: Complete the function $x(t)$ to return the function value given in the following equation

$$x(t) = a_1 \sin(\omega_1 t) + a_2 \sin(\omega_2 t) + a_3 \sin(\omega_3 t) \tag{2.4}$$

where $a_1 = 0.75$, $a_2 = 1$, $a_3 = 0.5$, $\omega_1 = 100\pi$, $\omega_2 = 400\pi$, $\omega_3 = 800\pi$ [Graded]

```
# Creating 3 sinusoidal signals
# Your code goes here
w1 = <------>
w2 = <------>
w3 = <------>
a1 = <------>
a2 = <------>
a3 = <------>
fs = 4095
ws = 2*np.pi*fs

def x(t):
    # Your code goes here
    x_t =
    <------------
    ------------->
    return x_t
```

Q: Write a python code to plot the waveform in time domain. Limit the x-axis to the interval $[0, 0.04]$. [Graded]

```
time = np.linspace(0,1,fs+1)
xt = [x(t_) for t_ in time]

# Plotting the input signal in time domain
# our code goes here
<————————————
————————————>
```

Q: Complete the python code for plotting ***absolute*** value of the Fourier transform of $x(t)$, that is X_ω against the angular frequency ω. Execute the cell and sketch the result. [Graded]

```
Xw = fft(xt, 4096)*2*np.pi/fs
Xw = fftshift(Xw)
k = np.arange(1,4097)
w = k/4096*ws - ws/2

# Plotting the input signal in frequency domain
fig, ax = plt.subplots()
# Your code goes here
<————————————
————————————>

ax.set_title('Frequency Response of the Input signal')
ax.set_xlabel('Angular frequency -'+r' $\omega$ (rad/s)')
ax.set_ylabel('Magnitude')
ax.set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,400*np.pi))
ax.set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,400)])
ax.set_xlim(-1000*np.pi, 1000*np.pi)
ax.set_yticks([0,np.pi/2,np.pi])
ax.set_yticklabels([0,r'$\pi$/2',r'$\pi$'])
plt.grid()
```

An ideal filter with following frequency response can be used to obtain the sinusoid with the angular frequency of $\omega_2 = 400\pi$, as the output waveform.

$$H(j\omega) = \begin{cases} 1 & \omega_{c1} < |\omega| < \omega_{c2} \\ 0 & otherwise \end{cases} \tag{2.5}$$

Here, ω_{c1} and ω_{c2} are cutoff frequencies and taken as the mid points of the impulses.

Q: Complete the function, ideal_filter(w) to output the $H(j\omega)$. [Graded]

```
# Ideal filter
wc1  = (w1+w2)/2
wc2  = (w2+w3)/2

def ideal_filter(w):
  # Your code goes here
  gain = 1
  <————————————
  ————————————>
  return gain
```

Q: Use the ideal_filter(w) function to fill the list H_{0w}, with the ideal filter value for each element in w. Complete the following code and sketch the output below. [Graded]

2.3.1 Ideal Filter: Part A

```
k = np.arange(1,4097)
w = k/4096*ws - ws/2
# Your code goes here
H0w  = <-----> 

# Simulation of Filtering
Y0w  = np.multiply(Xw,H0w)

# Obtaining the time domain signal
y0t  = ifft(fftshift(Y0w*fs/(2*np.pi)))

# Ideal filter frequency response (magnitude)
fig, axes  = plt.subplots(3,1, figsize=(18,18))
axes[0].plot(w,H0w)
axes[0].set_title('Frequency Response of the Ideal Filter')
axes[0].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[0].set_ylabel('Magnitude')
axes[0].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[0].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[0].set_xlim(-1000*np.pi, 1000*np.pi)
axes[0].grid()

# Frequency response of the ideal filter output (magnitude)
axes[1].plot(w,abs(Y0w))
axes[1].set_title('Fourier Transform of the Output Signal')
axes[1].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[1].set_ylabel('Magnitude')
axes[1].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[1].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[1].set_xlim(-1000*np.pi, 1000*np.pi)
axes[1].set_yticks([0,np.pi/2,np.pi])
axes[1].set_yticklabels([0,r'$\pi$/2',r'$\pi$'])
axes[1].grid()

# Output signal in time domain
axes[2].plot(time,np.real(y0t))
axes[2].set_title('Output Signal in time domain')
axes[2].set_xlabel('Time (s)')
axes[2].set_ylabel('Amplitude')
axes[2].set_xlim(0, 0.04)
axes[2].grid()
```

2.3.2 Ideal Filter: Part B

Execute the bellow cells and observe the output.

```
# Actual Filter
b, a  = signal.butter(5, [2*wc1/ws, 2*wc2/ws], 'bandpass', analog=False)
ww, h = signal.freqz(b, a, 2047)
ww    = np.append(-np.flipud(ww), ww)*ws/(2*np.pi)
h     = np.append(np.flipud(h), h)

# Filtering
y     = signal.lfilter(b,a,xt)
```

```
# Obtaining the frequency response of the output signal
Y    = fft(y,4096)*2*np.pi/fs
Y    = fftshift(Y)
```

```
# Actual filter frequency response (magnitude)
fig, axes  = plt.subplots(3,1, figsize=(18,18))
axes[0].plot(ww, abs(h) )
axes[0].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[0].set_ylabel('Magnitude')
axes[0].set_title('Frequency Response of the Actual Filter')
axes[0].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[0].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[0].set_xlim(-1000*np.pi, 1000*np.pi)
axes[0].grid()

# Frequency response of the actual filter output (magnitude)
axes[1].plot(w,abs(Y))
axes[1].set_title('Fourier Transform of the Output Signal')
axes[1].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[1].set_ylabel('Magnitude')
axes[1].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[1].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[1].set_xlim(-1000*np.pi, 1000*np.pi)
axes[1].set_yticks([0,np.pi/2,np.pi])
axes[1].set_yticklabels([0,r'$\pi$/2',r'$\pi$'])
axes[1].grid()

# # Output signal in time domain
axes[2].plot(time,np.real(y))
axes[2].set_title('Output Signal in time domain')
axes[2].set_xlabel('Time (s)')
axes[2].set_ylabel('Amplitude')
axes[2].set_xlim(0, 0.04)
axes[2].grid()
```

Q: Comment on your observations in Part - A and Part - B. [Graded]

2.4 Removing Power Line Noise in an ECG Signal

The electrocardiogram (ECG) is a biomedical signal which gives electrical activity of heart. An ECG signal is characterized by six peaks and valleys, which are traditionally labeled P, Q, R, S, T, and U, as shown in Figure 2. ECG has frequency range from 0.5 Hz to 80 Hz and power line interference, mainly coming from electromagnetic interference by power line, introduces 50-Hz frequency component in the ECG signal. This is a major cause of corruption of ECG. In this section, we will design a simple filter to remove the power line interference from an ECG signal. The three main components of an ECG signal are the P wave (depolarization of the atria) the QRS complex (depolarization of the ventricles) and the T wave,(repolarization of the ventricles), as shown in Fig. 2.1

Task 1. *Write a python script to read the data in the file ecg_signal.csv and fill the list ecg with the data.*

```
# Reading the ECG data
ecg = []
# EDIT HERE
```

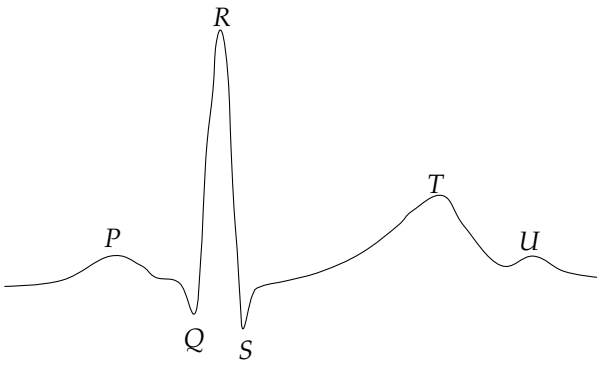
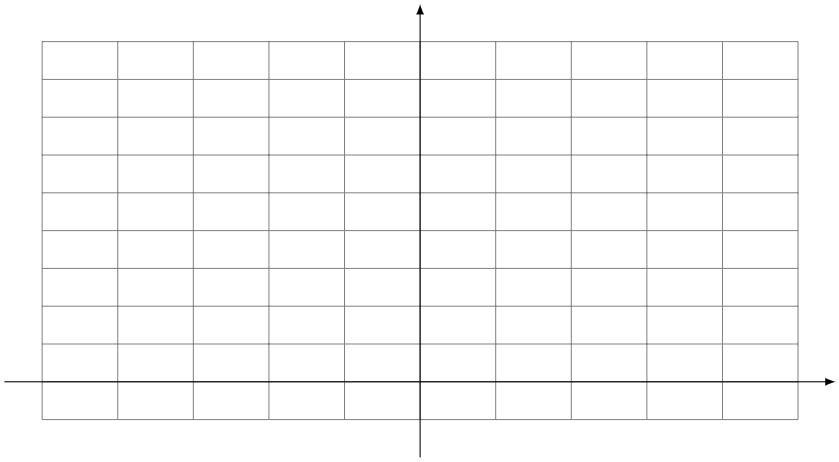


Figure 2.1: PQRS Components of ECG

```
duration = 10 # seconds
T = duration/len(ecg)
Fs = 1/T

# Obtaining the fourier transform
F = fftshift(fft(ecg))
fr = np.linspace(−Fs/2, Fs/2, len(F))
```

Task 2. *Plot the absolute value of the Fourier transform with respect to frequency. Limit the x-axis to the interval* $[-100, 100]$. *Sketch the output.*



Task 3. *What type of filter that can be used to remove the noise at* 50 Hz*?*

Task 4. *Edit the code below with the correct name of the filter selecting from the table given below. Execute the cell and sketch the frequency response of the filter.*

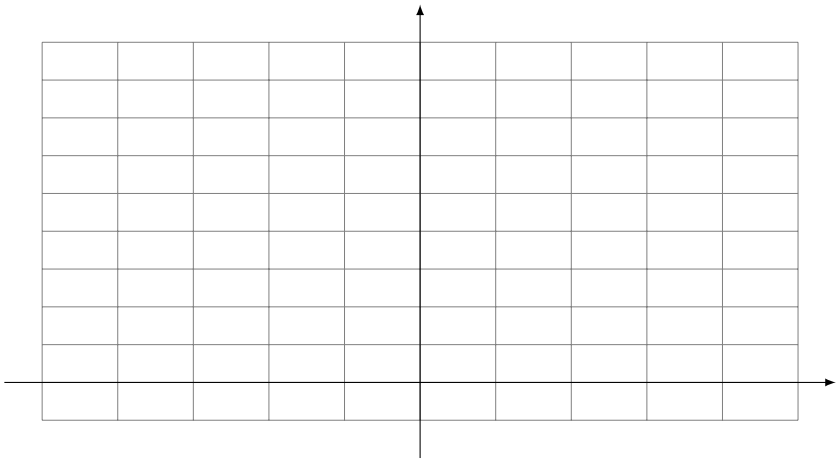
```
# Designing the filter
f1 = 49
f2 = 51
filter_type = '' # EDIT HERE
b, a = signal.butter(2, [2*f1/Fs, 2*f2/Fs], filter_type , analog=False)

# Obtaining the frequency response of the filter
ww, h = signal.freqz(b, a, 2047)
```

```
ww = np.append(−np.flipud(ww), ww)
h = np.append(np.flipud(h), h)

# Plotting the frequency response
fig, ax = plt.subplots(figsize=(10,6))
ax.plot(ww∗Fs/(2∗np.pi), abs(h) )
ax.set_title('Frequency Response of the Actual Filter')
ax.set_xlabel('Frequency [Hz]')
ax.set_ylabel('Magnitude')
ax.set_xlim(−100,100)
ax.grid()
```

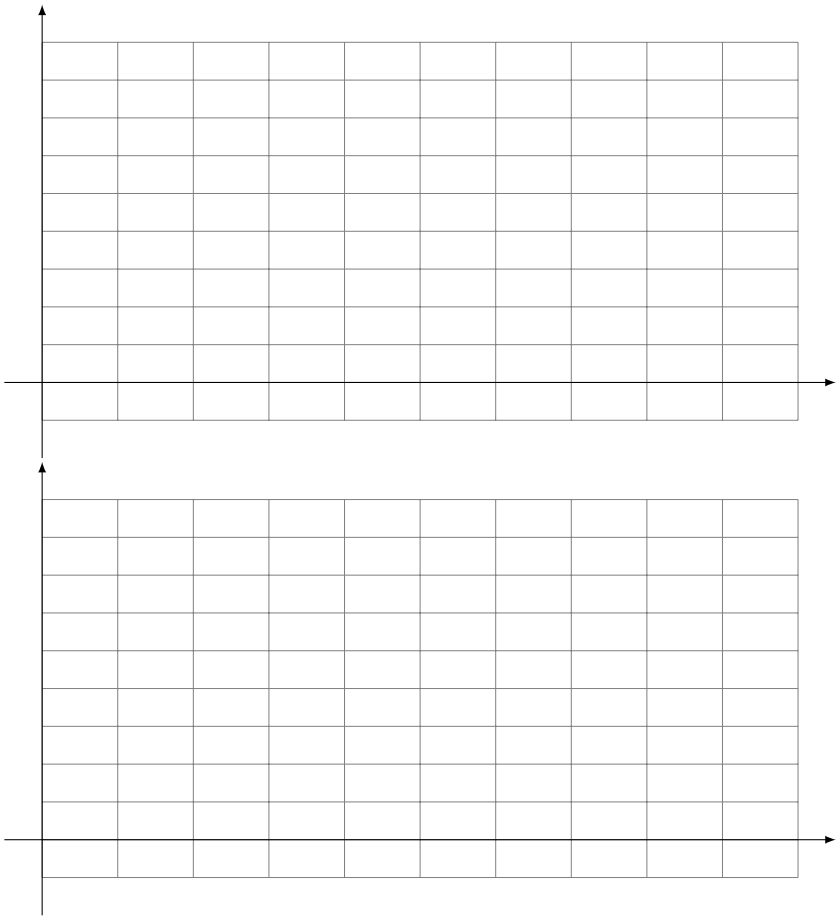
Filter type	Name to be used in code
Low-pass filter	'lowpass'
Band-pass filter	'bandpass'
High-pass filter	'highpass'
Band-stop filter	'bandstop'



Task 5. *Edit the given below to plot the input and the output waveforms vs time. Use the subplots function to plot the graphs in two axes in the same figure. Limit the x-axis to the interval* $[0, 3]$. *Sketch the result.*

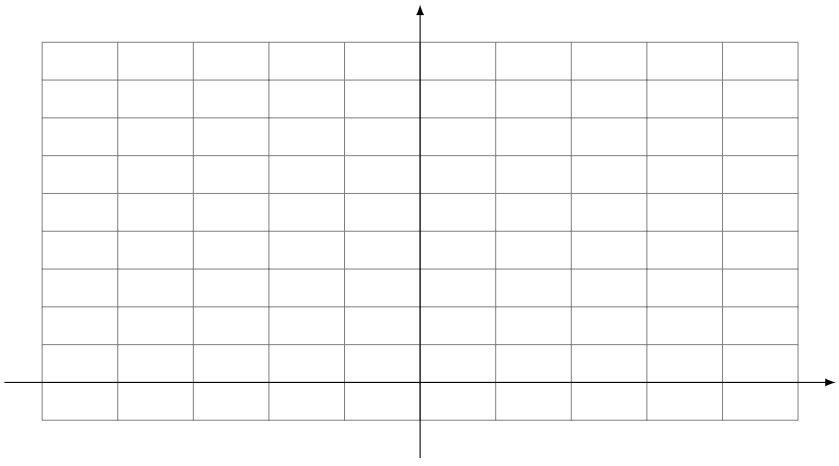
```
time = np.arange(T, duration+T, T)

# Filtering the ECG wavefoem
output = signal.lfilter(b, a, ecg)
```



Task 6. *Complete the code in below to plot the absolute value of Fourier transform of the output waveform with respect to the frequency. Limit the x axis to the interval* $[-100, 100]$. *Execute the cell sketch the output.*

```
F = fftshift(fft(output))
```



Part III

Electronics

Workshop 2: Building and Taking Measurements of Operational Amplifier Circuits

Objective: To design and implement commonly used operational amplifier circuits

Outcome: After successful completion of this session, the student will be able to

1. Design and implement the following operational amplifier application circuits:
   - Inverting voltage amplifier
   - Non-inverting voltage amplifier
   - Voltage comparator
   - Differentiator

Equipment Required:

1. Digital Oscilloscope
2. Signal Generator
3. DC power supply
4. Digital multi-meter
5. Breadboard and wires

Components Required:

1. NE5532 (1 No.)
2. Resistors - 1kΩ (3 Nos.), 10kΩ(3 Nos), 100kΩ (2 Nos.)
3. Capacitors - 1nF (1 No.)
4. 100K potentiometer (1 No.)

2.1 Introduction

Operational amplifiers, often referred to as *"op-amps"*, are high-gain voltage amplifiers with a differential input and , usually a signle-ended output. There are operational amplifiers with differential output as well. Such operational amplifiers are referred to as *"fully differential operational amplifiers"*. In this experiment, we will be focusing on the more commonly used general op-amps. The symbol and the pin-out for a general operational amplifier is shown in figure 2.1, where:

- V_+ is the non-inverting input
- V_- is the inverting input
- V_{s+} is the positive power supply
- V_{s-} is the negative power supply
- V_o is the output

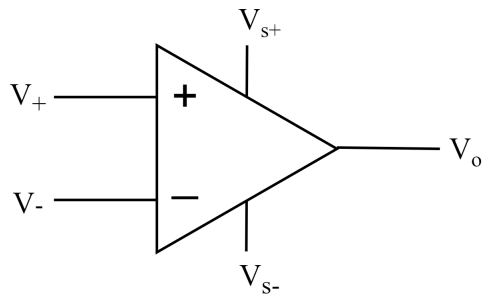


Figure 2.1: Symbol and pin-out of a general operational amplifier

The most noteworthy characteristics of operational amplifiers include very high open-loop voltage gain, very high input impedance and very low output impedance. Due to these special characteristics, the use of operational amplifiers in electronic circuits have become extremely popular nowadays. In this experiment, we will be building and analyzing four basic operational amplifier application circuits.

2.2 Pre-Lab

Task 1. *What is the typical range of values for the open-loop gain of practical operational amplifiers?*

Task 2. *What are the two golden rules which are used to explain the functionality of operational amplifiers? (These two rules explain the governing principles of ideal operational amplifiers. However, these rules are quite useful when analyzing practical operational amplifier circuits.)*

Task 3. *Unlike ideal operational amplifiers, practical operational amplifiers have a finite bandwidth. Explain what is meant by the bandwidth of an operational amplifier.*

Task 4. *What are the undesirable effects or limitations introduced by the finite bandwidth of practical operational amplifiers?*

Task 5. *In operational amplifier terminology, What is meant by "Saturation"?*

In this experiment, you will implement the following commonly used circuits which uses operational amplifiers.

- Inverting voltage amplifier
- Non-inverting voltage amplifier
- Voltage comparator
- Differentiator

2.3 Inverting Voltage Amplifier

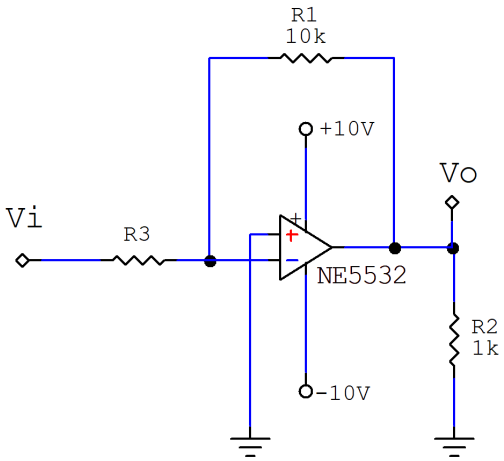


Figure 2.2: Schematic diagram of the inverting amplifier

Task 6. *Derive an equation for the voltage gain of the amplifier in terms of R1 and R3.*

Task 7. *Calculate the value suitable for R3 in order to achieve a gain of (-10).*

Task 8. *Construct the circuit show in figure 2.2. Use the value calculated in task 7 for R3.*

Task 9. *Configure a 10kHz 2Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up inverting amplifier. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 10. *Comment on any differences in the observed output, with respect to the output observed if an ideal operational amplifier was used. (Hint: Calculate the expected output amplitude and measure the observed output amplitude.)*

Task 11. *Suggest any changes that can be made (in the components or the power supply) in order to make the output waveform similar to that of an ideal operational amplifier, without reducing the gain.*

2.4 Non-inverting Voltage Amplifier

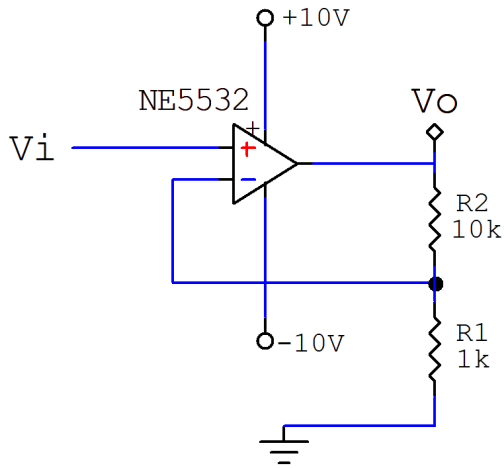


Figure 2.3: Schematic diagram of the non-inverting amplifier

Task 12. *Construct the circuit show in figure 2.3.*

Task 13. *Calculate the theoretical voltage gain.*

Task 14. *Configure a 1kHz 200mVpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up non-inverting amplifier. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 15. *Calculate the practical voltage gain (using the observed input and output waveforms).*

2.5 Voltage Comparator

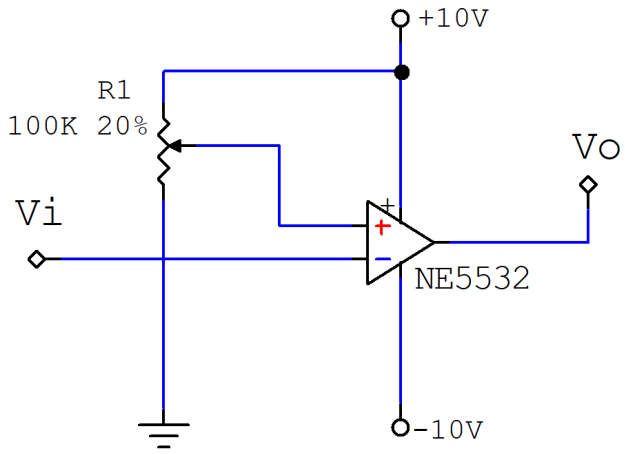


Figure 2.4: Schematic diagram of the comparator

Task 16. *Construct the circuit show in figure 2.4.*

Task 17. *Adjust the potentiometer and set the voltage at the non-inverting terminal to 2V.*

Task 18. *Configure a 1kHz 6Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up comparator. Observe both the input and the output of the amplifier using the oscilloscope.*

Task 19. *Adjust the vertical scales of the oscilloscope channels so that both channels have the **same** vertical scale.*

Task 20. *Enable voltage cursors of one channel and place a cursor at 2V.*

Task 21. *Plot the observed waveforms and the voltage cursor (set at 2V) on top of each other. Make sure you label each waveform and the cursor.*

2.6 Differentiator

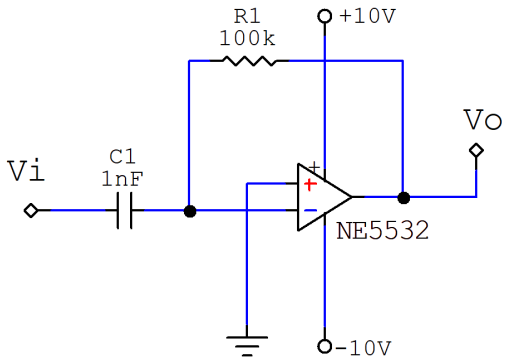


Figure 2.5: Schematic diagram of the differentiator

Task 22. *Construct the circuit show in figure 2.5.*

Task 23. *Configure a 1kHz 2Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up differentiator. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 24. *Configure a 1kHz 2Vpp Square waveform on the signal generator, give that waveform as the input signal to the powered-up differentiator. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 25. *Derive the relationship between V_i and V_o.*

♣ The End ♣

Part IV

Telecommunication

Workshop 2: Baseband Communication

Objective: To simulate and analyze baseband transmission.

Outcome: After successful completion of this session, the student would be able to

1. Identify the basic elements of a baseband communication system
2. Implement a simple baseband communication system using MATLAB
3. Simulate and analyze the bit-error rate of a baseband communication system under different settings
4. Implement a simple error-correction mechanism

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online
3. A headphone set (to be brought by the student)

Components Required: None.

2.1 Baseband Communication

In telecommunications the signals are usually transmitted after converting them to higher frequencies. This process is called modulation and the transmitted signals are known as broadband signals. But the properties of the transmission can be analyzed using a communication system without modulation. Such a system is known as a baseband communication system. Figure 2.1 shows a baseband and a broadband signal in frequency domain side-by-side.

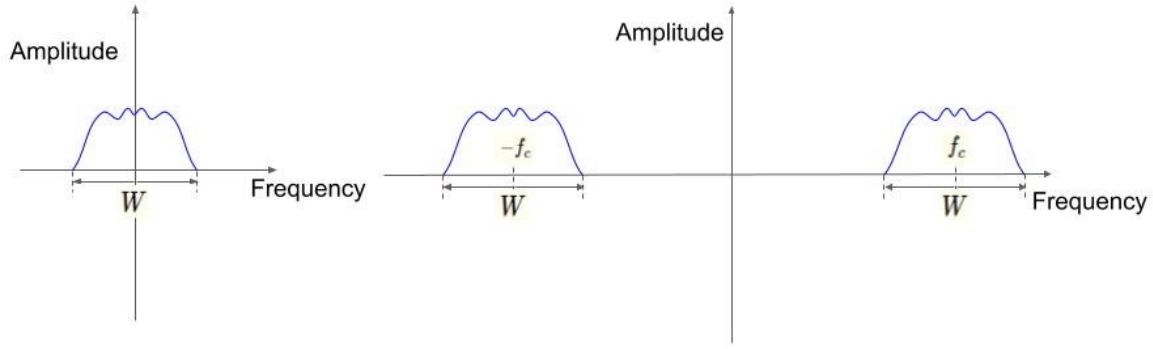


Figure 2.1: Frequency domain representation of a signal in baseband (left) vs broadband (right). Bandwidth of the signal is W and the center frequency of the broadband signal is f_c.

In this practical we will be simulating a baseband communication system using MATLAB and we will be analyzing the bit error rate of the system under different settings. Finally we will study a simple error-correction mechanism.

2.2 Pre-Lab

Prior to the lab we will implement the baseband communication system using MATLAB. You will have to record audio from your headphones. The recorded audio will be tramsitted through the communication system and will be reconstructed at the receiver.

2.2.1 Implementing a Baseband Communication System

Figure 2.2 illustrates the block diagram of a simple baseband communication system.

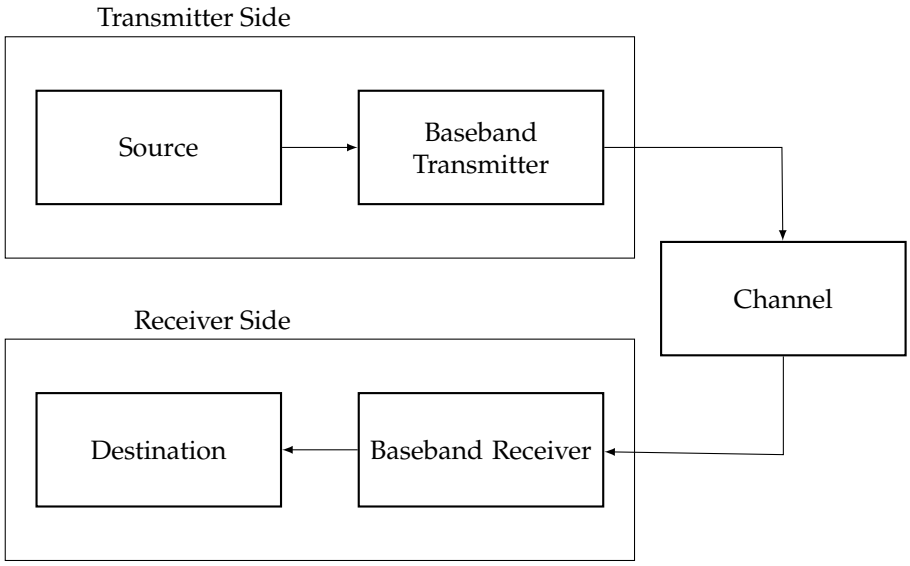


Figure 2.2: The block diagram of a baseband communication system

Next we will understand each block along with the implementation details.

Source

Source is the device/person which generates the signal to be transmitted. In this case you will be the source since you are generating the audio waveform. The analog signal from the source is sent to the transmitter.

Transmitter

Transmitter performs several modifications to the signal received from the source. First, the analog signal is sampled at discrete time intervals (The analog signal is represented as a set of samples). This process is known as **sampling**. Your headphone will perform the sampling.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1);  %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
```

If you have trouble accessing the audio recorders (this can happen in MATLAB online), you can use recorded audio. An audio clip named "Recording.wav" will be uploaded to the Moodle. You can directly

use this audio clip. In case of MATLAB online, upload the file to MATLAB online. Then use the following code instead of the above code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
[audio_samples,Fs] = audioread('Recording.wav'); %Replace filename with the location of your file.
```

If you prefer to record your own audio, first you will have to record a 5 second audio clip from your PC. Then convert it to .wav format using an online converter. You can use https://www.aconvert.com/audio/. Make sure you set the sampling rate to 6000. Rename the file as "Recording.wav" and use it for the workshop.

Next, we have to represent the set of samples using a bit stream in order to transmit as a digital signal. For that, we **quantize** each sample, where we approximate each sample with a discrete level. In this case we represent each sample with one of 256 discrete levels.

```
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                            %of the recorded audio

for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
   num = 2*num;
   if(num>=1)
    bit_stream(1,(i-1)*qbits+j) =  1;
    num = num - 1;
   else
    bit_stream(1,(i-1)*qbits+j) =  0;
   end
 end
end
```

Now we have the binary stream of 1's and 0's which we aim to transmit. Although we transmit bits we still have to represent (**encode**) them using analog signals. We will have to represent bit zero and bit one using two distinct finite duration signals. For simplicity, we use two sinusoidal signals with different phases (phase_0 and phase_1). For this part we will use $\text{phase}_1 = 0$ and $\text{phase}_1 = \pi$ (This formation is called as antipodal signalling since a bit 0 is represented by the negative of the signal used to represent bit1).

$$\begin{aligned} \text{bit0} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_0) \quad 0 \leq t \leq T, \\ \text{bit1} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_1) \quad 0 \leq t \leq T, \end{aligned} \tag{2.1}$$

where T is the bit period.

Although in a practical implementation these signals are implemented in analog domain, in MATLAB we approximate the actual analog signal with a set of samples (We cannot create analog signals in MATLAB).

```
samples_per_bit = 100;  %Number of samples used to approximate the
                %sinosoid for a single bit (We approximate
                % sin(Wc*t+phase0) or sin(Wc*t+phase1) with 100
                % bits in this case
```

```
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal
X = zeros([1,number_of_samples]); %X(t) - This array is used to store the transmitted analog signal


%This section does the encoding which is described previously

power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit1;
  else
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit0;
  end
end
```

At this point the transmitted signal is stored in the array X and is ready to be transmitted (X is a discrete time approximation of the actual transmitted signal $X(t)$).

Channel

The signals are transmitted through channels (cables for wired transmissions and free space for wireless transmissions). The channel induces several effects to the transmitted signal. Below we model the induced effects.

First, the power of the signal is reduced when it is propagating through the channel. This is known as **attenuation**. Next, channels add **noise** to the transmitted signal. In addition, channels also filter out certain frequencies of the signal (This effect is known as channel **bandwidth limitation**).

The signal after attenuation and noise is

$$\tilde{X}(t) = AX(t) + N(t). \tag{2.2}$$

Here A is the attenuation factor (The proportion of the power of the transmitted signal received at the receiver. $N(t)$ is the noise which is a random signal containing random values. The signal $\tilde{X}(t)$ is filtered (bandlimited) to produce the signal $Y(t)$. We use a low-pass butterworth filter of order 10 to model the filtering effect.

```
att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
     % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,samples_per_bit*number_of_bits); %N(t) - noise signal
X_hat = att*X + N; %signal after noise

%Filtering Effect
fc_butter = fc*25;
```

```
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
```

Receiver

The receiver receives $Y(t)$. Since $Y(t)$ and $X(t)$ are not the same, we have to figure out a method to obtain (**decode**) the transmitted bit stream.

In this case we use phase estimation to perform decoding. Observe that if the transmitted bit was a zero, the phase of the encoding sinosoid was phase_0, and if the transmitted bit was a one, the phase of the encoding sinosoid was phase_1. But due to noise addition and bandwidth limitation the phase may get changed. If the estimated phase is close to phase_0 than phase_1, we interpret the bit as a zero and vice versa. Observe that depending on the strength of the noise signal, bit errors may occur at the receiver.

```
decoded_bit_stream = zeros([1,number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
   if(f(1) >= 0)
      decoded_phase = pi/2;
   else
      decoded_phase = -pi/2;
   end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
       decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
       decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
     decoded_bit_stream(i) = 0;
  else
     decoded_bit_stream(i) = 1;
  end
end
```

Next, we **reconstruct** the audio samples from the bit-stream.

```
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
   st = 0.5;
   for j=2:qbits
     decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream((i-1)*qbits+j);
     st = st/2;
   end
   if(decoded_bit_stream((i-1)*qbits+1) == 1)
      decoded_audio_samples(i) = -decoded_audio_samples(i);
   end
```

```
end
```

Destination

Destination is the device/person to which the signal transmission is intended. In this case you are the destination. Now we will listen to the reconstructed signal.

```
pause(10);
sound(5*decoded_audio_samples, Fs);
```

Change the amplitude of the transmitted signal and observe the impact of amplitude on the quality of the reconstructed signal.

2.2.2 Comparing the Original and the Reconstructed Signals

Now we will plot the original recorded analog audio signal and the signal reconstructed at the receiver, side by side.

```
figure(1)
subplot(1,2,1)
plot(audio_samples)
title('Original Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
subplot(1,2,2)
plot(decoded_audio_samples);
title('Decoded Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
```

Task 1. *Compare the two signals and comment on the observed differences.*

2.2.3 Comparing the Original and the Transmitted Signals in Frequency Domain

Use the following code to plot the frequency domain representations of the original recorded analog audio signal and the transmitted signal, side by side.

```
L_1 = size_audio_samples;
Audio_freq = fft(audio_samples);
Audio_freq_norm = abs(Audio_freq/L_1);
Audio_freq_norm_one = Audio_freq_norm(1:L_1/2+1);
Audio_freq_norm_one(2:end-1) = 2*Audio_freq_norm_one(2:end-1);

L_2 = samples_per_bit*number_of_bits;
X_freq = fft(X);
X_freq_norm = abs(X_freq/L_2);
X_freq_norm_one = X_freq_norm(1:L_2/2+1);
X_freq_norm_one(2:end-1) = 2*X_freq_norm_one(2:end-1);

f_1 = Fs*(0:(L_1/2))/L_1;
f_2 = sampling_rate*(0:(L_2/2))/L_2;

figure(2)
```

```
subplot(1,2,1)
plot(f_1,Audio_freq_norm_one);
title('Single-Sided Amplitude Spectrum of the Original Audio Stream');
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(1,2,2)
plot(f_2,X_freq_norm_one);
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

Task 2. *Compare the two frequency spectra and comment on the observed differences*

2.3 Analyzing the Bit-Error Rate (BER) Using a Baseband Communication System

Next we will calculate the bit-error rate of the system.

Task 3. *Write a code snippet to calculate the bit-error rate (BER) for the transmission discussed in section 2.2.1. Include your code at the bottom of the "Pre_Lab.m" file given in the Moodle, and submit the updated MATLAB file with the BER calculation to Moodle. In your code, assign the calculated BER to a variable named "BER". (Hint: The original bit stream is the array "bit_stream" and the decoded bit stream is the array decoded_bit_stream". Your code should calculate the error rate using the difference between these two arrays.)*

Task 4. *What is the calculated bit-error rate?*

Now we will analyze the bit-error rate of a baseband communication system. You will have to vary the signal amplitude and observe it's effect on the bit-error rate. Note that since we keep the noise variance constant, the Signal-to-Noise ratio is proportional to the signal amplitude.

Task 5. *Repeat section 2.2.1 with integer amplitudes ranging from -10 to 10 to 10 dB (Hint: You can use a for loop on the Amplitude variable to achieve this). Plot the Amplitude (dB) vs BER graph in the Task Sheet.*

Task 6. *Repeat the same procedure of task 5 with* $phase_1 = 0$ *and* $phase_1 = \pi/2$ *and plot the graph in the Task Sheet.*

Task 7. *Comment on the differences of the two graphs and the reasons for the difference.*

2.4 Implementing a Simple Error Correction Mechanism

We observed that bit-errors occur due to the channel noise and bandwidth limitation. We can perform error correction at the receiver in order to correct some of the errors occurred. In this section we will be studying a simple error-correction mechanism (Known as an error correcting code).

In this mechanism we will transmit a single bit three times. At the receiver we will observe the three bits and consider the most frequent bit out of the three to be the correct bit.

Replace the code we used for 4.2 with the following code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1); %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
```

```
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                                %of the recorded audio
for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
    num = 2*num;
    if(num>=1)
      bit_stream(1,(i-1)*qbits+j) =  1;
      num = num - 1;
    else
      bit_stream(1,(i-1)*qbits+j) =  0;
    end
 end
end
samples_per_bit = 100;
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                  %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal

X = zeros([1,3*number_of_samples]); %X(t) - This array is used to store the transmitted analog signal
power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit1;
  else
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit0;
  end
end

att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
      % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,3*number_of_samples); %N(t) - noise signal
```

```
X_hat = att*X + N; %signal after noise
fc_butter = fc*25;
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
decoded_bit_stream = zeros([1,3*number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:3*number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
   if(f(1) >= 0)
     decoded_phase = pi/2;
   else
     decoded_phase = -pi/2;
   end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
      decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
      decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
    decoded_bit_stream(i) = 0;
  else
    decoded_bit_stream(i) = 1;
  end
end
decoded_bit_stream_error_corrected = zeros([1,number_of_bits]); %The array to store the decoded bit
                                     %stream after error-correction
for i = 1:number_of_bits %This loop does the task of correcting erros by looking at the most frequent
                  %bit from the three bits
  for j = 1:3
    decoded_bit_stream_error_corrected(i) = decoded_bit_stream_error_corrected(i)+decoded_bit_stream(3*(i-1)+j);
  end
  decoded_bit_stream_error_corrected(i)= round(decoded_bit_stream_error_corrected(i)/3);
end
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
  st = 0.5;
  for j=2:qbits
    decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream_error_corrected((i-1)*qbits+j);
    st = st/2;
  end
  if(decoded_bit_stream_error_corrected((i-1)*qbits+1) == 1)
    decoded_audio_samples(i) = -decoded_audio_samples(i);
  end
end
```

Go through the above code and understand how the code for section 2.2.1 is changed to perform error-correction.

Task 8. *Similar to task 3 implement the code to calculate the bit-error rate (Hint: the arrays "bit_stream" and*

"decoded_bit_stream_error_corrected" contain the transmit and received bit streams after error correction). What is the calculated BER?

Task 9. *Repeat task 5 and task 6 with error correction and include the graphs in the Task Sheet.*

Task 10. *Comment on the observations and the effect of using the error-correction mechanism on the bit-error rate.*

Task 11. *What are the disadvantages of using this error-correction mechanism*

♣ The End ♣