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

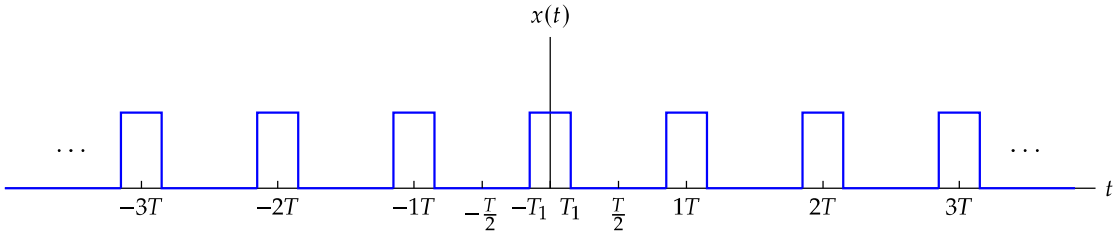
Components Required: None.

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
```

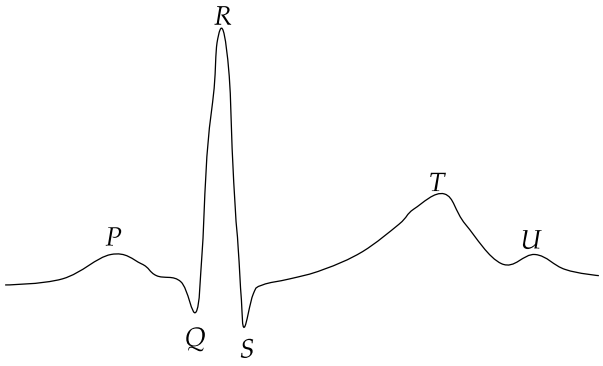
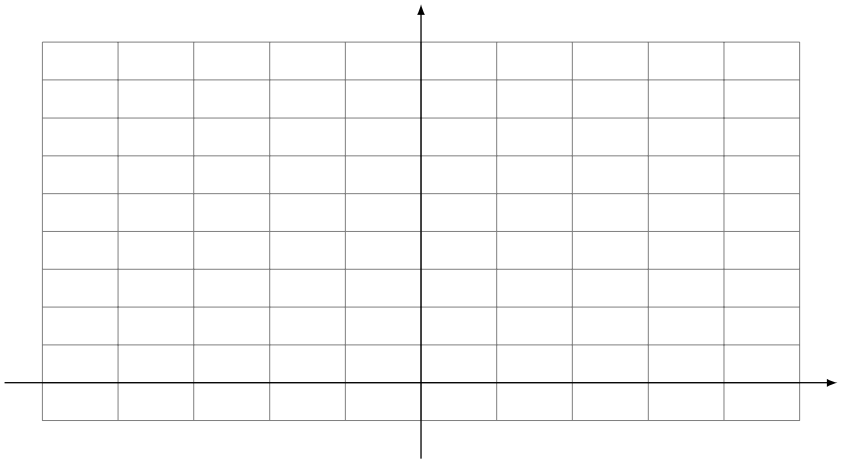


Figure 2.1: PQRS Components of ECG

```
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.*

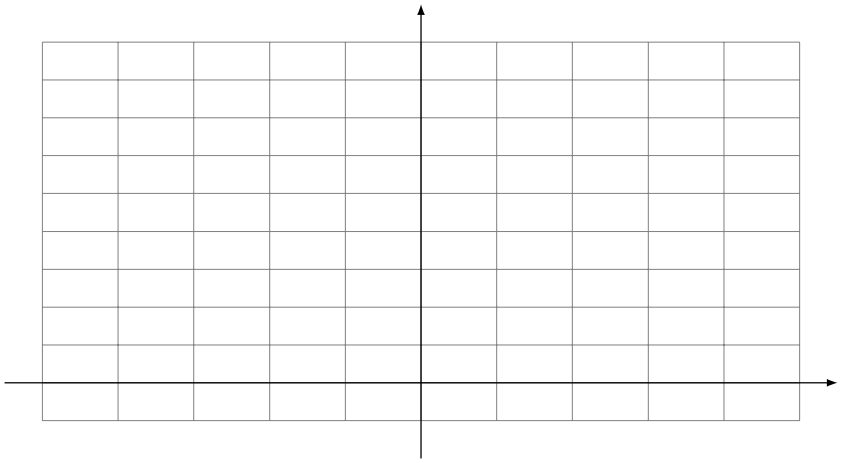
```
# Designing the filter
f1 = 49
f2 = 51
filter_type = '' # EDIT HERE
b, a = signal.butter(2, [2*f1/Fs, 2*f2/Fs], filter_type , analog=False)

# 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()
```

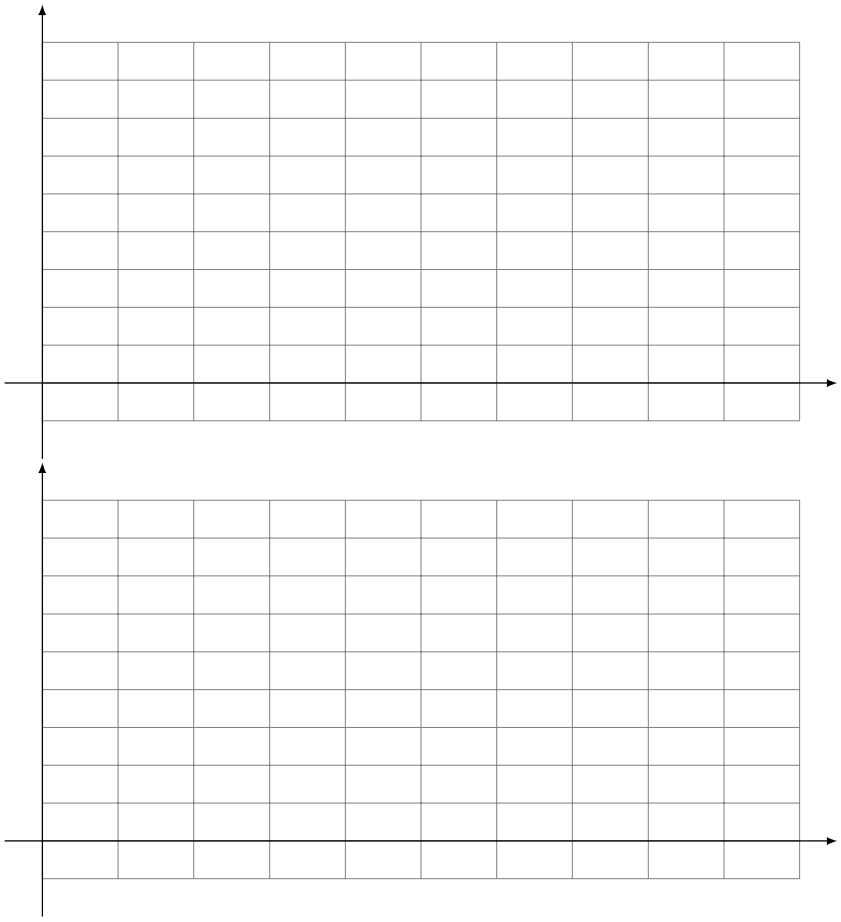
Filter type	Name to be used in code
Low-pass filter	'lowpass'
Band-pass filter	'bandpass'
High-pass filter	'highpass'
Band-stop filter	'bandstop'



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))
```