General Sir John Kotelawala Defense University

Electrical, Electronic & Telecommunication Engineering

ET3142 – Digital Signal Processing

Practical II - Semester V

KDU REG NO:D/ENG/20/0079/ET **Intake 37** **Marks**

Noise Removal from an Audio Signal using Fast Fourier Transform (FFT)

(Creating a filtered FFT from scratch)

1) Creating an audio file in Python with the following parameters.

- Frequency = 1000 Hz
- Sampling Rate = 48000.0 bps
- Number of samples = 48000
- Amplitude = 16000
- Number of frames (nframes) = 48000
- comptype="NONE"
- compname="not compressed"
- Number of Channels (nchannels) =1
- Sample Width (sampwidth) = 2 [which indicates 16 bits per sample]

Programme I :

```
import wave
import struct
import matplotlib.pyplot as plt
frequency = 1000
num_samples = 48000
sampling_rate = 48000.0
amplitude = 16000
file = "test.wav"
sine_wave = []
for x in range(num_samples):
    val = np.sin(2 * np.pi * frequency * x/sampling_rate)
    sine_wave.append(val)
```

```
nframes=num_samples
comptype="NONE"
compname="not compressed"
nchannels=1
sampwidth=2
wav_file=wave.open(file, 'w')
wav_file.setparams((nchannels, sampwidth, int(sampling_rate), nframes, comptype, compname))
for s in sine_wave:
  wav_file.writeframes(struct.pack('h', int(s*amplitude)))
```

I. Note down your observation from Programme I [2 marks]

An audio file was created with the following parameters.

Frequency=1000, no. of samples=48000, sampling rate=48000, amplitude=16000

A sinewave tuple is created, and from that number of sample values are created and is appended to the sinewave list. Next a wave file is created to write on, and the parameters were plotted to the file and was written in frames.

2) Convert information in test .wav audio file to frequency domain using FFT.

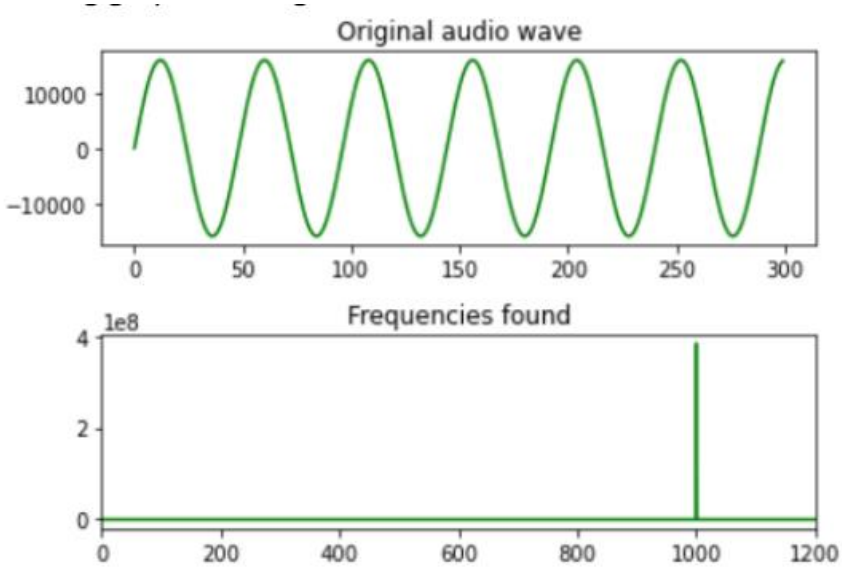
Programme II

```
frame_rate = 48000.0
infile = "test.wav"
num_samples = 48000
wav_file = wave.open(infile, 'r')
data = wav_file.readframes(num_samples)
wav_file.close()
```

```
data = struct.unpack('{n}h'.format(n=num_samples), data)
data = np.array(data)
data_fft = np.fft.fft(data)
frequencies = np.abs(data_fft)
plt.subplot(2,1,1)
plt.plot(data[:300])
plt.title("Original audio wave")
plt.subplot(2,1,2)
plt.plot(frequencies)
plt.title("Frequencies found")
plt.xlim(0,1200)
plt.show()
```

II. Note down your observation from Programme II [2 marks]

For the given number of samples, the wave file is read as frames. After unpacking the data, the original audio wave is hence plotted in time domain. An array of values is created with FFT data, and then the modulus values are taken. Then it is converted to frequency domain and is plotted which identifies the original wave audio as 1000Hz.



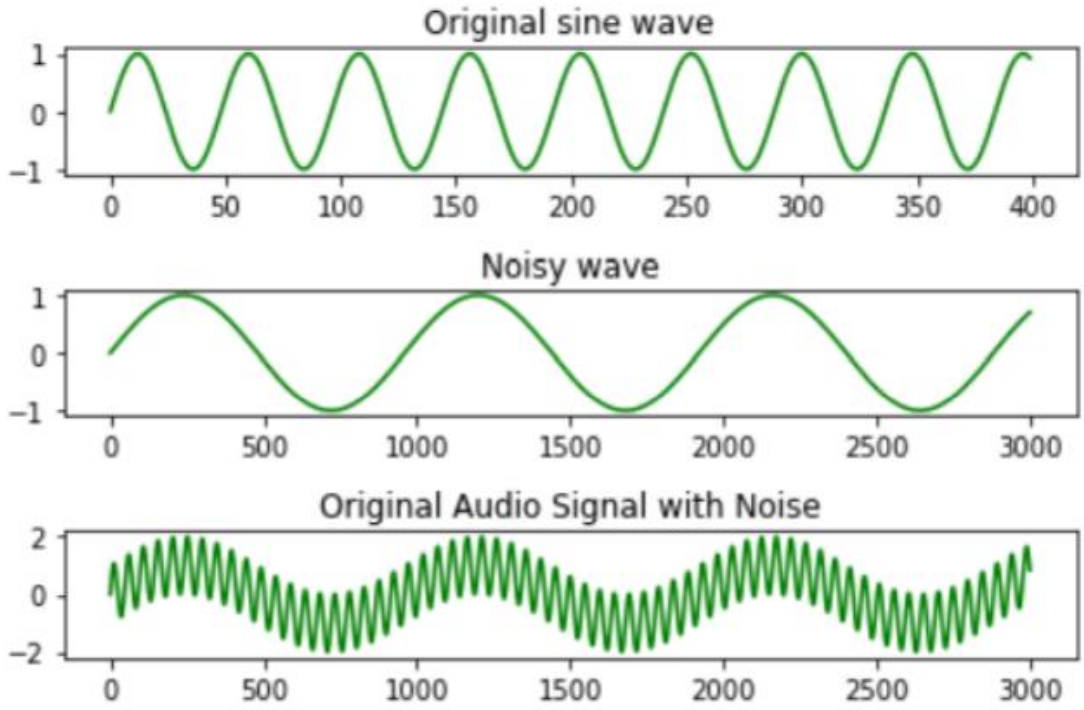
3) Creating a noisy audio file with noise frequency of 50 Hz

Programme III

```
frequency = 1000
noisy_freq = 50
num_samples = 48000
sampling_rate = 48000.0
#Create the sine wave and noise
sine_wave = [np.sin(2 * np.pi * frequency * x1 / sampling_rate) for x1 in range(num_samples)]
sine_noise = [np.sin(2 * np.pi * noisy_freq * x1/  sampling_rate) for x1 in range(num_samples)]
sine_wave = np.array(sine_wave)
sine_noise = np.array(sine_noise)
combined_signal = sine_wave + sine_noise
plt.subplot(3,1,1)
plt.title("Original sine wave")
plt.subplots_adjust(hspace=.5)
plt.plot(sine_wave[:500])
plt.subplot(3,1,2)
plt.title("Noisy wave")
plt.plot(sine_noise[:4000])
plt.subplot(3,1,3)
plt.title("Original Audio Signal with Noise")
plt.plot(combined_signal[:3000])
plt.show()
```

III. Note down your observation from Programme III [2 marks]

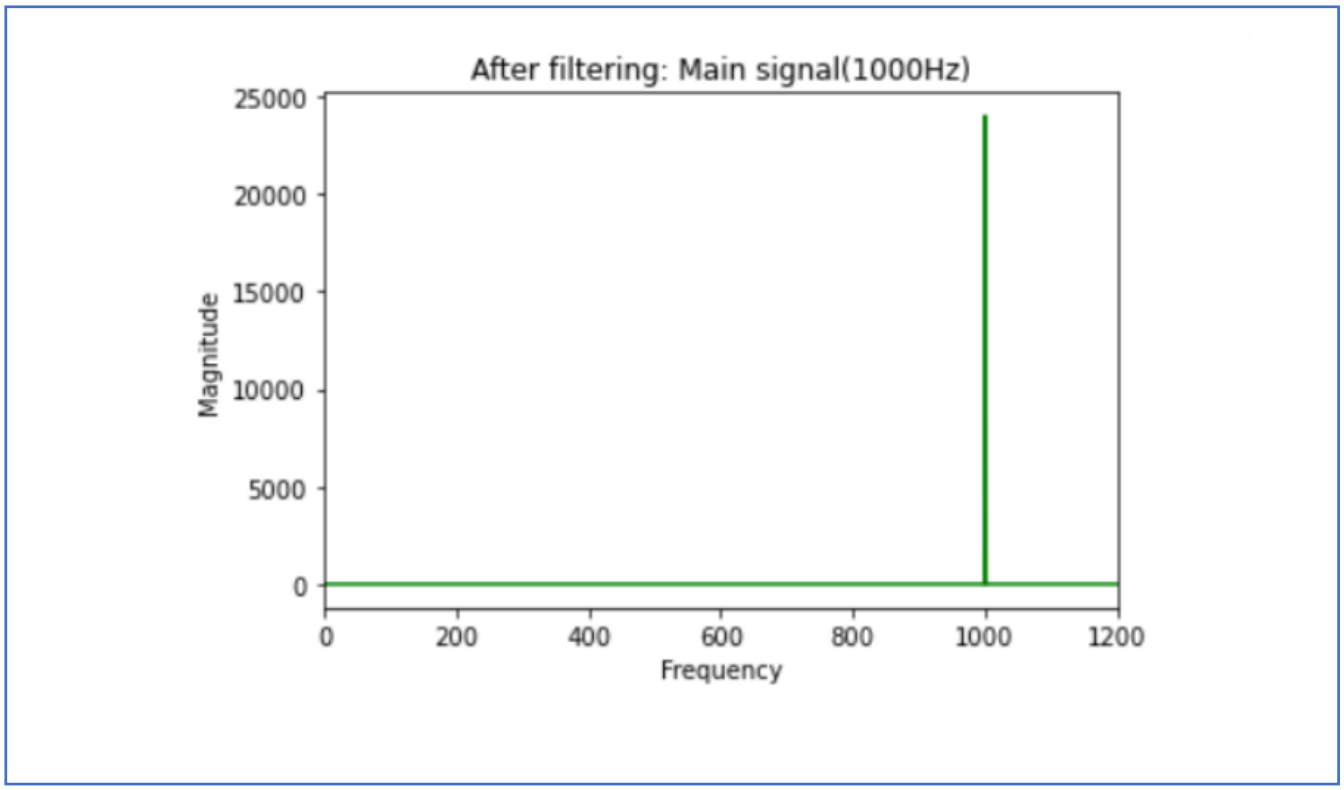
A noisy wave of 50Hz with the original sine wave is created, and both were added to create an audio signal with noise.



4) Represent the combined_signal in frequency domain.

Hint : use the absolute values from FFT. [np.abs()]

IV. Plot the graph which represents the combined_signal in frequency domain. Clearly Indicate the y and x axis. [2 marks]



V. Note down your observation for the plot defined in section IV. [2 marks]

Noisy Audio signal is converted to frequency domain using fft, and both plots are separately shown.

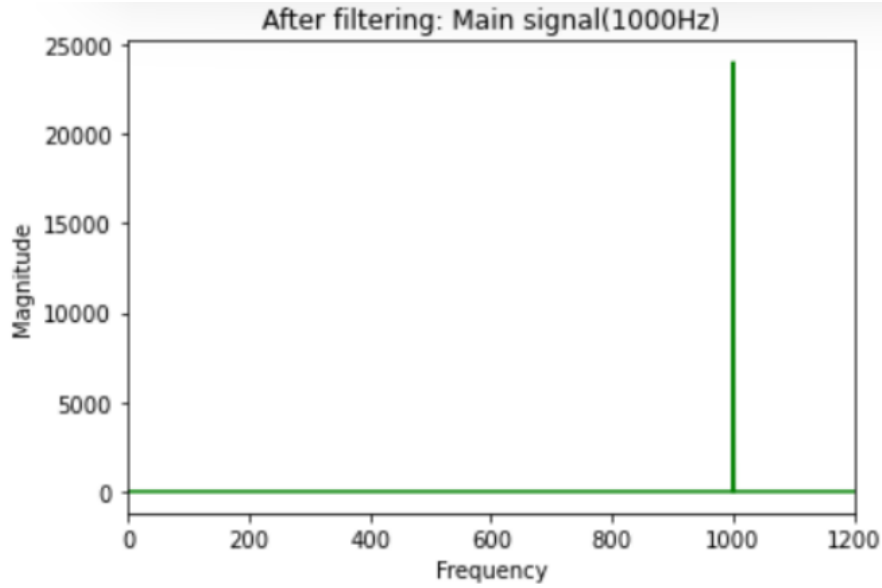
5) Create a filter to filter the original signal which has the frequency of 1000Hz

VI. Write a filter to filter 1000Hz original frequency spectrum. State the code below. [2 marks]

```
import wave
import struct
import matplotlib.pyplot as plt
import numpy as np
frequency = 1000
noisy_freq = 50
num_samples = 48000
sampling_rate = 48000.0
sine_wave = [np.sin(2 * np.pi * frequency * x1 / sampling_rate) for x1 in range(num_samples)]
sine_noise = [np.sin(2 * np.pi * noisy_freq * x1/ sampling_rate) for x1 in range(num_samples)]
sine_wave = np.array(sine_wave)
sine_noise = np.array(sine_noise)
combined_signal = sine_wave + sine_noise
data_fft = np.fft.fft(combined_signal)
```

```
freq = (np.abs(data_fft[:len(data_fft)]))

filtered_freq = [] index = 0 filtered_freq = [f if (950 < index < 1050 and f > 1) else 0 for index, f in enumerate(freq)]

plt.title("After filtering: Main signal(1000Hz)")

plt.xlim(0,1200)

plt.xlabel("Frequency")

plt.ylabel("Magnitude")

plt.plot(filtered_freq,'g')

plt.show()

plt.close()
```

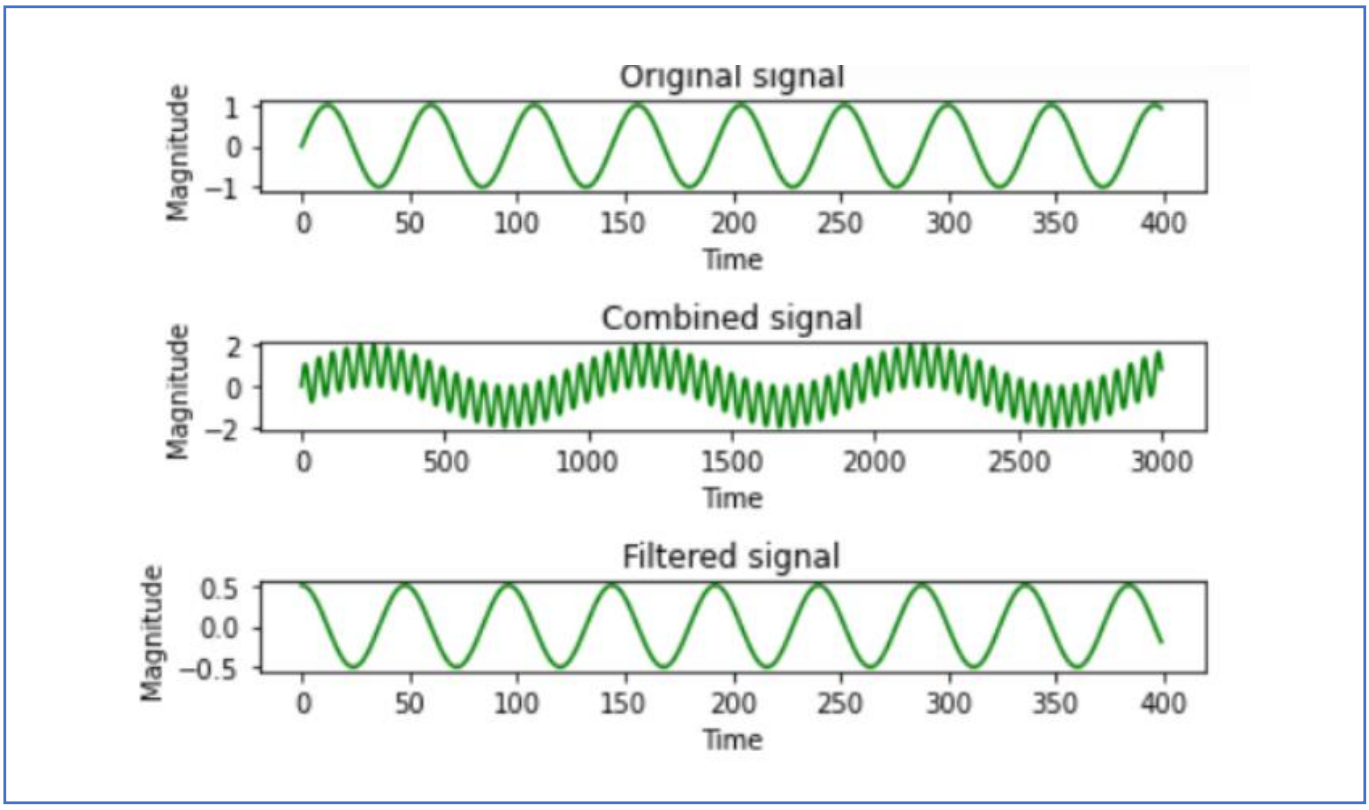
VII. Plot the filtered output plot [4 marks]



6) Filtered frequency domain signal convert to time domain.

Hint : recovered_signal = np.fft.ifft(*FilteredSignal*)

VIII. Plot the Original Signal, Combined Signal and Filtered signal in three subplots. [2 marks]



IX. Discuss your observation. [4 marks]

The frequency component is filtered, and the IFFT to retrieve original sine wave in time domain.

After the filtering process it is seen that the amplitude of the filtered signal is less than the amplitude of the original signal.