EC5011

DIGITAL SIGNAL PROCESSING THEORY AND APPLICATION

BANDARA H.G.T.D.

2022/E/048

GROUP CG04

SEMESTER 5

01 MAY 2025

PART1: SAMPLING, TIME DOMAIN & FREQUENCY DOMAIN REPRESENTATION

01.

part01_01.m | part02.m | part03.m

```
%BANDARA H.G.T.D. 2022e048
%Part 01

Fs = 4000;                  % Sampling frequency
t = 0:1/Fs:1-1/Fs;          % Time vector (1 sec)
x = cos(2*pi*100*t) + cos(2*pi*500*t) + ...
    cos(2*pi*2000*t) + cos(2*pi*2750*t);
y = 0.2 + x;                % y[n] = 0.2 + x[n]

N = length(y);
Y = fft(y);
f = (0:N-1)*(Fs/N);         % (a) Frequency in Hz
w = 2*pi*f/Fs;              % (b) In radians
f_norm = f / (Fs/2);        % (c) Normalized

% Plot y[n] and frequency domain in 4 subplots
subplot(4,1,1); plot(t, y); title('Time-Domain Signal y[n]'); xlabel('Time (s)'); ylabel('Amplitude');

subplot(4,1,2); plot(f, abs(Y)); title('Spectrum - Hz'); xlim([0 Fs/2]); xlabel('Frequency (Hz)');
subplot(4,1,3); plot(w, abs(Y)); title('Spectrum - Radians'); xlim([0 2*pi]); xlabel('Frequency (rad/sample)');
subplot(4,1,4); plot(f_norm, abs(Y)); title('Spectrum - Normalized'); xlim([0 1]); xlabel('Normalized Frequency');

%02 finding dc value
dc_value = mean(y);
disp(['DC Value = ', num2str(dc_value)]);
```

Figure 01:Matlab Code for SAMPLING, TIME DOMAIN & FREQUENCY DOMAIN REPRESENTATION

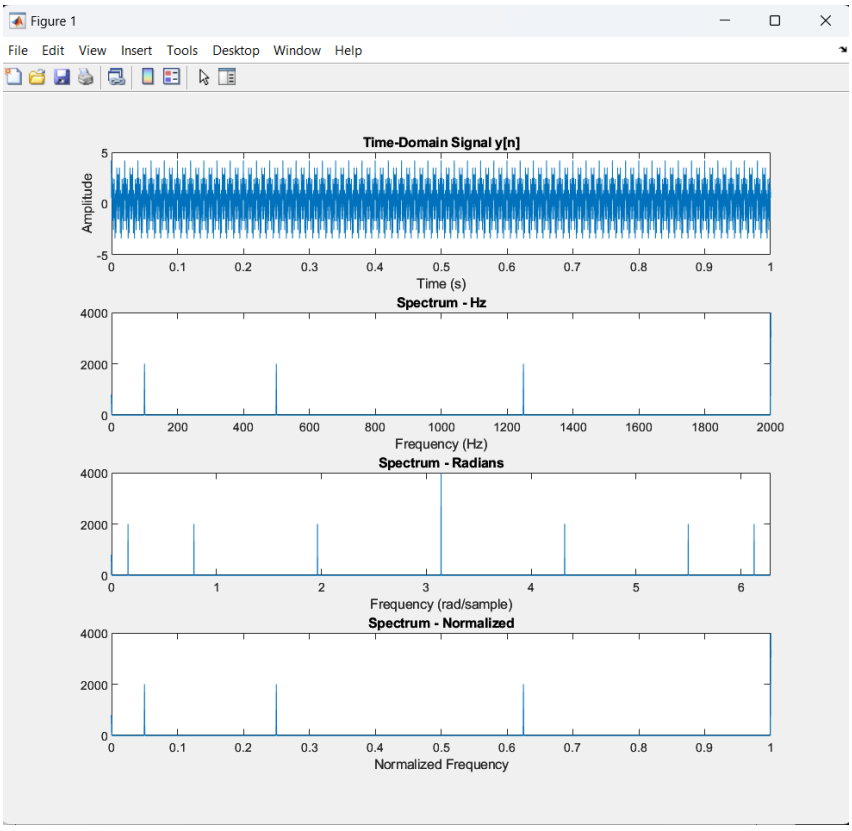

Figure 02:Output

2)

DC value of y[n]=0.2+x[n] is:

DC=0.2+mean of x[n]

```
>> part01_01
DC Value = 0.2
>>
```

3) Aliased frequency components = 2750 Hz

4)

part05.m | part06.m | part01_01.m | part01_after_5.m | part01_04.m | parameters.m

```
%BANDARA H.G.T.D. 2022e048
%Part 01_04

% Parameters
fs1 = 3000;       % Case 1: Low sampling rate (aliasing occurs)
fs2 = 8000;       % Case 2: Higher sampling rate (2000 Hz visible)
f_signal = 2000;  % Signal frequency in Hz
duration = 0.01;  % Signal duration in seconds

% Time vectors
t1 = 0:1/fs1:duration;
t2 = 0:1/fs2:duration;

% Generate signal y[n] = 0.2 + sin(2*pi*2000*t)
x1 = sin(2*pi*f_signal*t1);
y1 = 0.2 + x1;

x2 = sin(2*pi*f_signal*t2);
y2 = 0.2 + x2;

% FFT of y1 (low fs, aliasing)
N1 = length(y1);
Y1 = fft(y1, 1024); % Use zero-padding for better frequency resolution
f1 = fs1*(0:1023)/1024;

% FFT of y2 (high fs, 2000 Hz visible)
N2 = length(y2);
Y2 = fft(y2, 1024);
f2 = fs2*(0:1023)/1024;

% Plot full spectrum up to fs (not just fs/2)
figure;

subplot(2,1,1);
plot(f1, abs(Y1));
title('Spectrum with fs = 3000 Hz (Aliasing Case)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
grid on;
xlim([0 fs1]);

subplot(2,1,2);
plot(f2, abs(Y2));
title('Spectrum with fs = 8000 Hz (2000 Hz visible)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
grid on;
xlim([0 fs2]);
```

FIGURE 03: MATLAB CODE

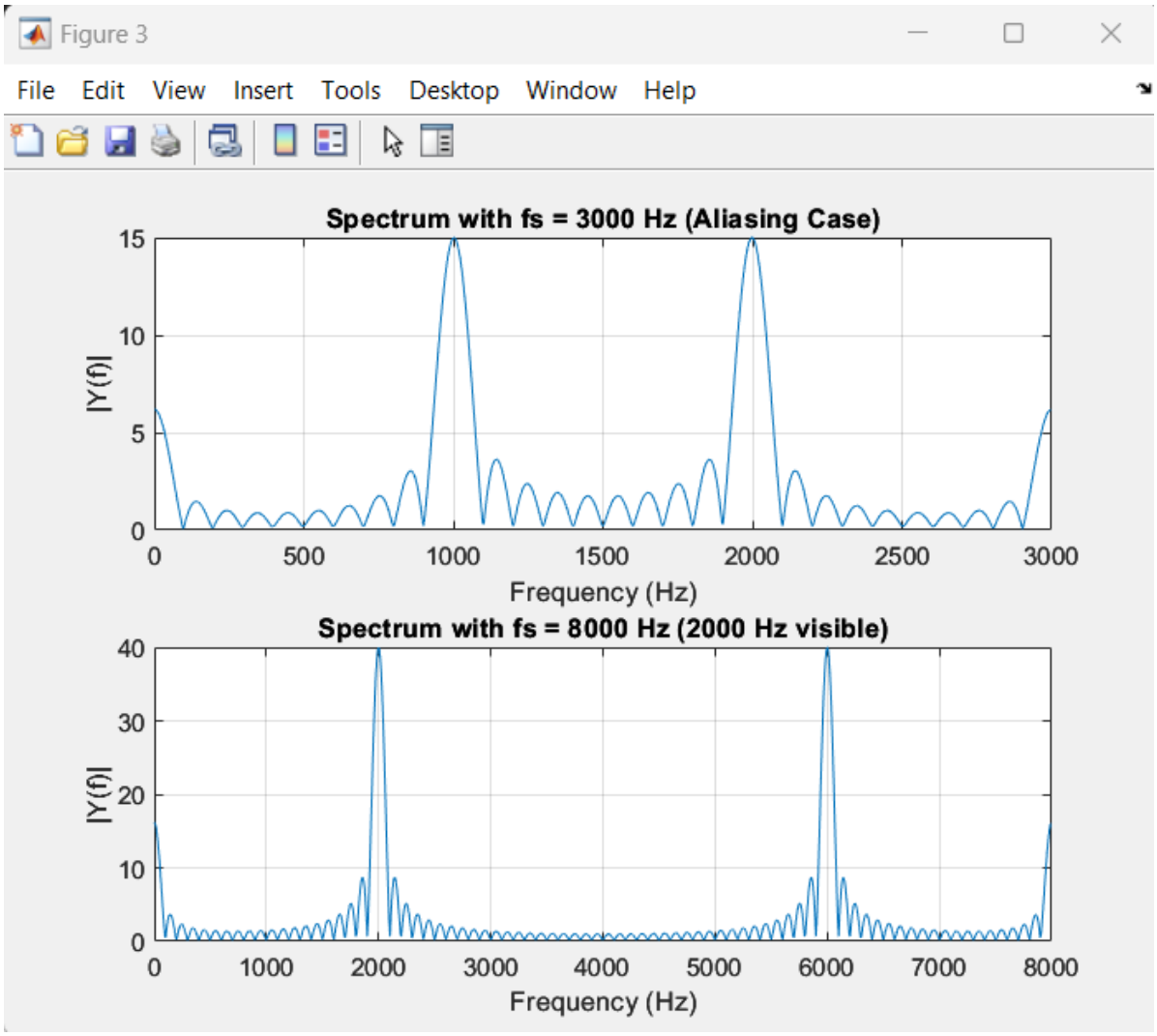

FIGURE04: OUTPUT

The 2000Hz frequency component is not detectable because it aligns with the folding frequency at 2000Hz, causing it to remain unnoticed. Thus, to integrate this frequency into the spectrum, the sampling frequency must surpass 4000Hz.

5)

+4 part03.m part04.m part05.m part06.m part01_01.m part01_after_5.m

```
%BANDARA H.G.T.D. 2022e048
%Part 01_05

% Original signal with Fs = 4000 Hz (hard to see 2000 Hz)
Fs1 = 4000;
t1 = 0:1/Fs1:1-1/Fs1;
y1 = 0.2 + cos(2*pi*100*t1) + cos(2*pi*500*t1) + cos(2*pi*2000*t1);

Y1 = fft(y1);
f1 = (0:length(Y1)-1)*(Fs1/length(Y1));

figure;
subplot(2,1,1);
plot(f1, abs(Y1));
title('Spectrum (Fs = 4000 Hz)'); xlabel('Frequency (Hz)'); xlim([0 Fs1]);

% Improved signal with Fs = 8000 Hz (clearly shows 2000 Hz)
Fs2 = 8000;
t2 = 0:1/Fs2:1-1/Fs2;
y2 = 0.2 + cos(2*pi*100*t2) + cos(2*pi*500*t2) + cos(2*pi*2000*t2);

Y2 = fft(y2);
f2 = (0:length(Y2)-1)*(Fs2/length(Y2));

subplot(2,1,2);
plot(f2, abs(Y2));
title('Spectrum (Fs = 8000 Hz)'); xlabel('Frequency (Hz)'); xlim([0 Fs2]);

%q05 after
Fs = 4000;
n = 0:3999;  % 4000 samples = 1 second

x = cos(2*pi*100*n/Fs) + cos(2*pi*500*n/Fs) + ...
    cos(2*pi*2000*n/Fs) + cos(2*pi*2750*n/Fs);  % original signal
p = cos(2*pi*900*n/Fs) + cos(2*pi*1200*n/Fs);   % new signal


q = zeros(size(n));
q(1:400) = x(1:400);           % 0 ≤ n < 400
q(401:end) = p(401:end);       % 400 ≤ n < 4000


%Step 3: Plot q[n] and its spectrum
figure;

% Time domain
subplot(2,1,1);
plot(n, q);
xlabel('n'); ylabel('q[n]');
title('Signal q[n]');

% Frequency domain
Q = fft(q);
f = (0:length(Q)-1)*(Fs/length(Q));

subplot(2,1,2);
plot(f(1:end/2), abs(Q(1:end/2)));
xlabel('Frequency (Hz)');
ylabel('|Q(f)|');
title('Magnitude Spectrum of q[n]');

%Solution: Use r[n] = x[n] + p[n]
r = x + p;

R = fft(r);
f = (0:length(R)-1)*(Fs/length(R));

figure;
plot(f(1:end/2), abs(R(1:end/2)));
xlabel('Frequency (Hz)');
ylabel('|R(f)|');
title('Spectrum of r[n] = x[n] + p[n]');
```

FIGURE 05: MATLAB CODE

PART2: TIME DOMAIN PROCESSING OF AUDIO SIGNALS

01.

part01_01.m | part01_after_5.m | part01_04.m | part02.m

```
%BANDARA H.G.T.D. 2022e048
%Part 02

[y, fs] = audioread('Uoj.wav');
disp(fs); % Shows sampling frequency
```

Figure 06: Matlab code for find the sampling frequency

Command Window

```
>> part02
       22050
```

FIGURE 07: OUTPUT

Sampling frequency = 22050 Hz

02.

part01_01.m | part01_after_5.m | part01_04.m | part02.m

```
%BANDARA H.G.T.D. 2022e048
%Part 02

% Load audio file
[y, Fs] = audioread('Uoj.wav');  % y = signal, Fs = sampling frequency

% Time axis in milliseconds
t = (0:length(y)-1) / Fs * 1000;  % Convert to milliseconds

% Plot time-domain signal
figure;
subplot(4,1,1);
plot(t, y);
xlabel('Time (ms)');
ylabel('Amplitude');
title('Time Domain Signal');

% Compute FFT
N = length(y);
Y = fft(y);
Y = Y(1:N/2+1);  % Take only up to Nyquist frequency
f_hz = Fs * (0:N/2) / N;         % Frequency in Hz
f_rad = 2 * pi * f_hz / Fs;      % Frequency in radians/sample
f_norm = f_hz / Fs;              % Normalized frequency

% Plot (a) Frequency in Hz
subplot(4,1,2);
plot(f_hz, abs(Y));
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
title('Frequency Spectrum (Hz)');

% Plot (b) Frequency in radians/sample
subplot(4,1,3);
plot(f_rad, abs(Y));
xlabel('Frequency (radians/sample)');
ylabel('|Y(\omega)|');
title('Frequency Spectrum (Radians/sample)');

% Plot (c) Normalized Frequency
subplot(4,1,4);
plot(f_norm, abs(Y));
xlabel('Normalized Frequency (f / Fs)');
ylabel('|Y(f)|');
title('Frequency Spectrum (Normalized)');
```

FIGURE 08:MATLAB CODE

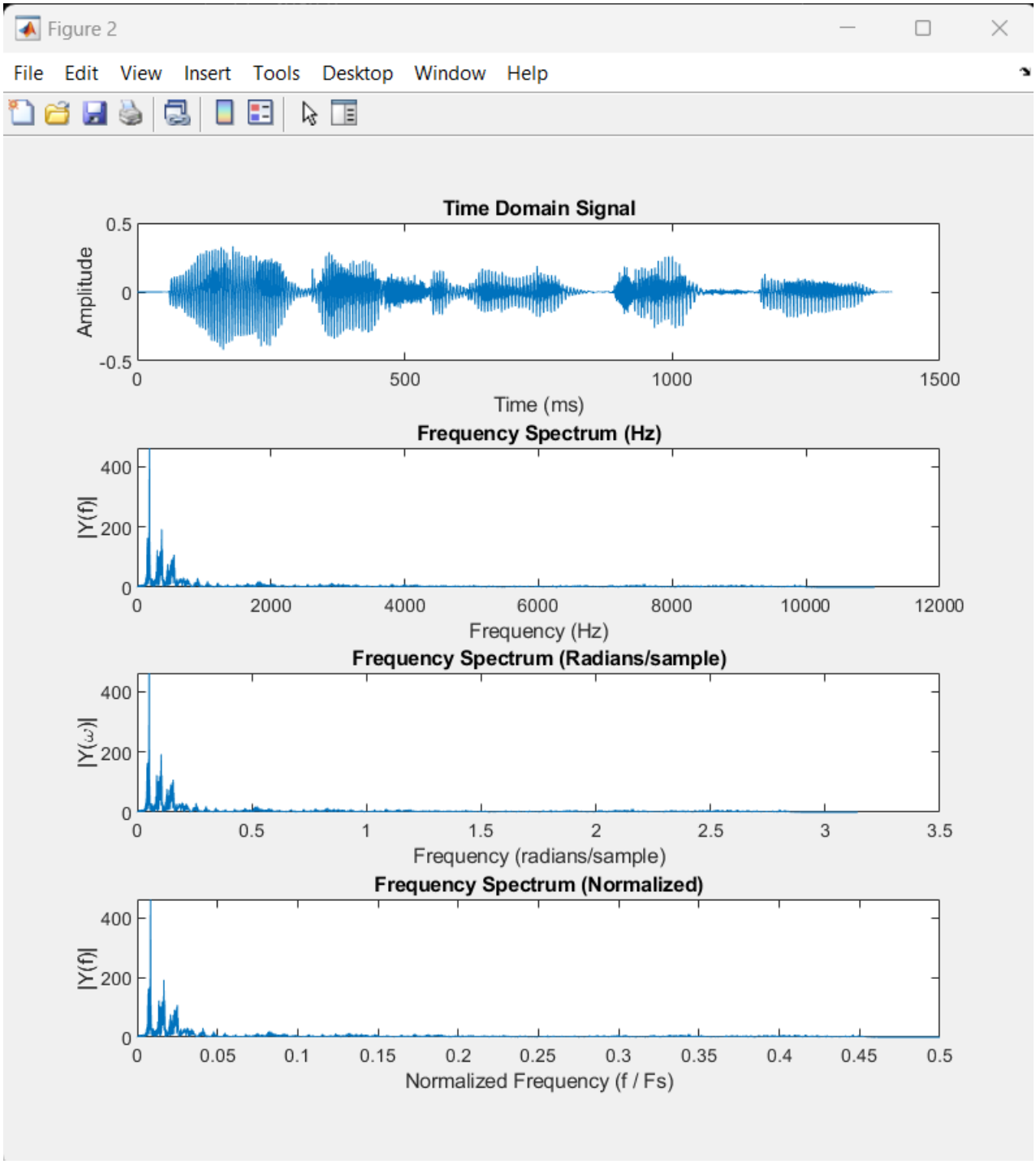

FIGURE 09:OUTPUT

3)

part01_01.m | part01_after_5.m | part01_04.m | part02.m | part02_02.m

```
%BANDARA H.G.T.D. 2022e048
%Part 02_02

% Load the full audio
[y, Fs] = audioread('Uoj.wav');

% Time vector for full signal (optional for reference)
t_full = (0:length(y)-1)/Fs;

% --- Estimated sample ranges (you must adjust these based on your audio pl
university_range = 00001:16000;
of_range         = 16000:19500;
jaffna_range     = 20000:30000;

% --- Extract words ---
university = y(university_range);
of_word    = y(of_range);
jaffna     = y(jaffna_range);

% --- Time vectors for each word ---
t_uni = (0:length(university)-1)/Fs * 1000;  % ms
t_of  = (0:length(of_word)-1)/Fs * 1000;     % ms
t_jaf = (0:length(jaffna)-1)/Fs * 1000;      % ms

% --- Plotting ---
figure;

subplot(3,1,1);
plot(t_uni, university);
xlabel('Time (ms)');
ylabel('Amplitude');
title('Word: University');

subplot(3,1,2);
plot(t_of, of_word);
xlabel('Time (ms)');
ylabel('Amplitude');
title('Word: of');

subplot(3,1,3);
plot(t_jaf, jaffna);
xlabel('Time (ms)');
ylabel('Amplitude');
title('Word: Jaffna');

```

FIGURE 10: MATLAB CODE

part01_01.m | part01_after_5.m | part01_04.m | part02.m | part02_02.m | part02_02_all_words.m

```
% Load the audio
[y, Fs] = audioread('Uoj.wav');

% Define sample ranges (update these based on your waveform!)
university_range = 00001:16000;
of_range         = 16000:19500;
jaffna_range     = 20000:30000;

% Extract word signals
university = y(university_range);
of_word    = y(of_range);
jaffna     = y(jaffna_range);

% Create time axes in milliseconds
t_uni = (0:length(university)-1)/Fs * 1000;
t_of  = (length(university):(length(university)+length(of_word)-1))/Fs * 1000;
t_jaf = (length(university)+length(of_word):(length(university)+length(of_word)+length(jaffna)-1))/Fs * 1000;

% Concatenate time and amplitude for continuity in x-axis
figure;
hold on;
plot(t_uni, university, 'b', 'DisplayName', 'University');
plot(t_of, of_word, 'r', 'DisplayName', 'of');
plot(t_jaf, jaffna, 'g', 'DisplayName', 'Jaffna');
xlabel('Time (ms)');
ylabel('Amplitude');
title('Words: University, of, Jaffna');
legend;
grid on;
```

FIGURE 11:MATLAB CODE FOR GET ALL WORDS IN ONE PLOT

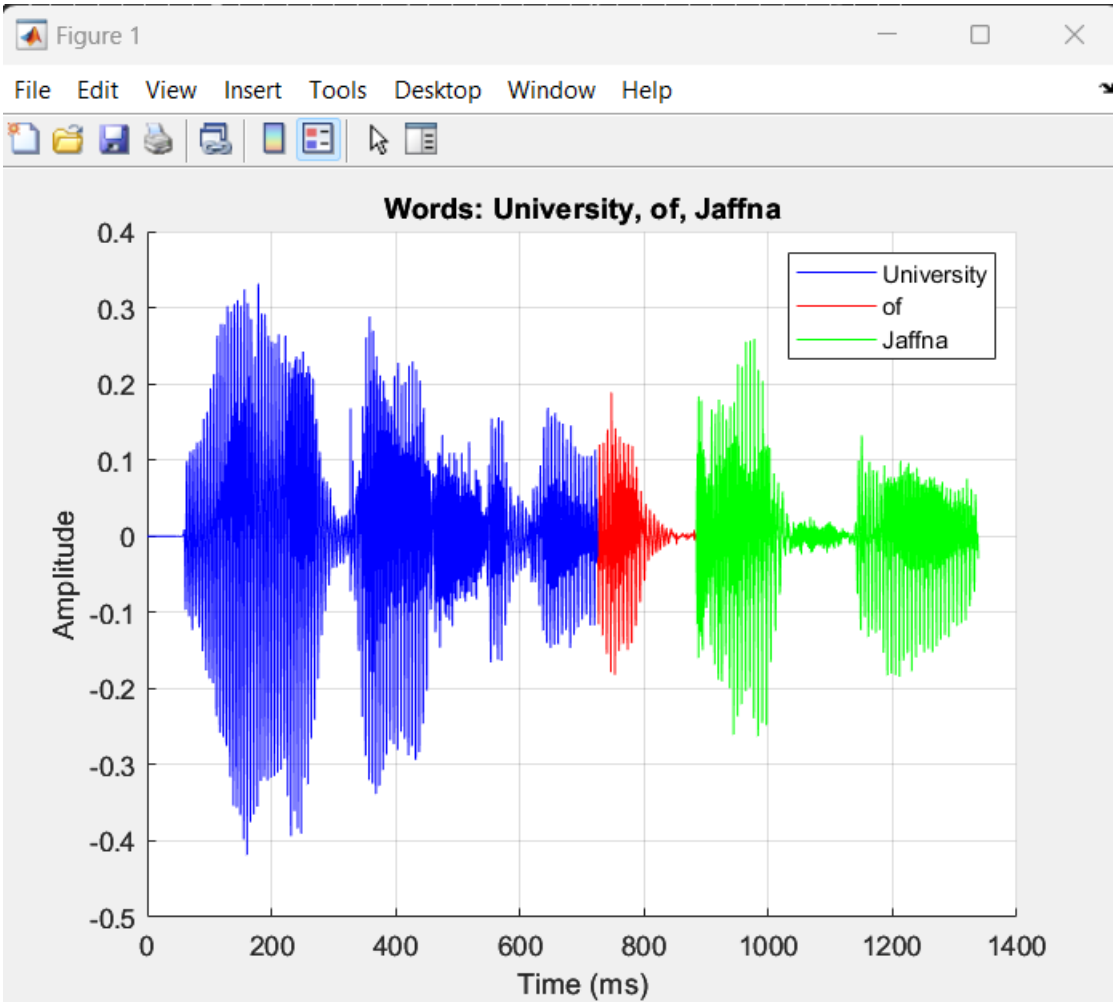

FIGURE 12:OUTPUT

4) 5)

```
+2  part01_04.m  part02.m  part02_02.m  part02_02_all_words.m  Part02_03_playthreewords
1   %BANDARA H.G.T.D. 2022e048
2   %Part 02_02_Play three words
3
4   % Load the full audio
5   [y, Fs] = audioread('Uoj.wav');
6
7   % === Step 1: Define sample ranges for each word ===
8   university_range = 00001:16000;
9   of_range         = 16000:19500;
10  jaffna_range     = 20000:30000;
11
12  % === Step 2: Extract each word ===
13  university = y(university_range);
14  of_word    = y(of_range);
15  jaffna     = y(jaffna_range);
16
17  % === Step 3: Play each word with a pause in between ===
18  disp('Playing "University"');
19  sound(university, Fs);
20  pause(length(university)/Fs + 0.5);  % wait for playback + short gap
21
22  disp('Playing "of"');
23  sound(of_word, Fs);
24  pause(length(of_word)/Fs + 0.5);
25
26  disp('Playing "Jaffna"');
27  sound(jaffna, Fs);
28  pause(length(jaffna)/Fs + 0.5);
29  %5) Save them
30  audiowrite('university.wav', university, Fs);
31  audiowrite('of.wav', of_word, Fs);
32  audiowrite('jaffna.wav', jaffna, Fs);
33
```

FIGURE 13: MATLAB CODE FOR PLAY ONE BY ONE SEPARATELY

PART3: CONVOLUTION AND FILTERING IN LTI SYSTEM

01.

+3 | part02.m | part02_02.m | part02_02_all_words.m | Part02_03_playthreewords.m | part03.m

```
%BANDARA H.G.T.D. 2022e048
%Part 03

x = [1 0 4 0 3];              % Input
h = [0.6 0.2 -0.4];           % Impulse response

% 1. Using convolution
y_conv = conv(x, h);

% 2. Using filter
b = h;                        % Numerator coefficients
a = 1;                        % Denominator (FIR)
y_filter = filter(b, a, x);

% 3. Using initial condition
zi = [1 2];                   % Initial condition (for longer systems)
[y_ic, zf] = filter(b, a, x, zi);

% 4. Plot all
subplot(4,1,1); stem(x); title('Input x[n]');
subplot(4,1,2); stem(y_conv); title('Using conv()');
subplot(4,1,3); stem(y_filter); title('Using filter()');
subplot(4,1,4); stem(y_ic); title('Using filter() with initial condition');


% 7. manual convolution
x = [1 2 3];
h = [1 -1];
N = length(x) + length(h) - 1;
y = zeros(1, N);

for n = 1:N
    for k = 1:length(x)
        if n-k+1 > 0 && n-k+1 <= length(h)
            y(n) = y(n) + x(k)*h(n-k+1);
        end
    end
end
stem(y);

```

FIGURE 14:FUL MATLAB CODE FOR ALL SUBPARTS IN PART 03

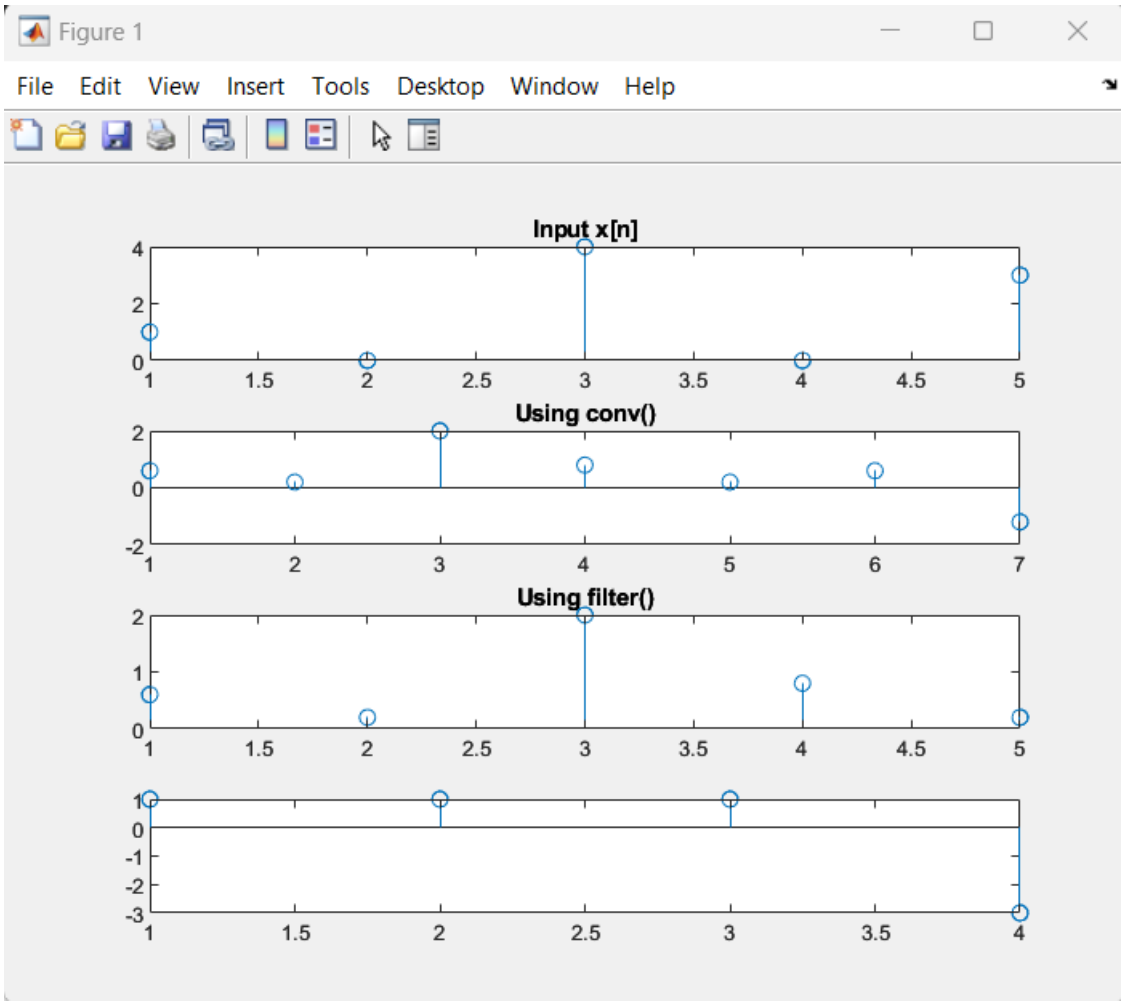

FIGURE 15: OUTPUT

03. **What it does**: Implements convolution assuming causal system starting at n=0n=0n=0.

Length of result: Same as x.

filter is based on the difference equation of the system:

$y[n]=h[0]x[n]+h[1]x[n-1]+h[2]x[n-2]+...y[n] = h[0]x[n] + h[1]x[n-1] + h[2]x[n-2] + ...y[n]=h[0]x[n]+h[1]x[n-1]+h[2]x[n-2]+...$

04) **Initial Condition (zi)**

- Represents the **memory of the filter** before the current input is applied.
- Useful when We're processing signals in **blocks** — it ensures **continuity** between blocks.

Final Condition (zf)

- Captures the **final internal state** of the filter **after** processing a block.
- We can use zf as the zi for the **next block** to simulate a continuous filter state.

06)

- **conv(x, h):**

- **Full convolution**, includes all output samples from n=0n = 0n=0 to n=N+M−2n = N + M - 2n=N+M−2
- **Used for theoretical analysis** or non-causal systems.

- **filter(h, 1, x):**

- Used in **real-time or causal systems**.
- Output length = length of input x
- Computation starts with assuming **zero past values**, unless given via zi.

- **Final Condition (zf):**

- Stores the **last internal filter state**.
- Important for **processing signals in blocks** — allows continuity between blocks.

CONCLUSION

In this lab, we explored the foundational concepts and practical applications of digital signal processing through three major sections: sampling and frequency analysis, time-domain processing of audio signals, and convolution/filtering in LTI systems.

- **Part 1: Sampling and Frequency Domain Analysis**

We began by generating and analyzing sampled signals using both time and frequency representations. By applying the FFT, we visualized how aliasing occurs when the signal frequency exceeds half the sampling rate (Nyquist limit). We learned that increasing the sampling frequency allows us to accurately capture higher-frequency components (e.g., 2000 Hz), and we demonstrated this effect with clear MATLAB plots. This part reinforced our understanding of signal reconstruction, aliasing, and frequency scaling in Hz, radians/sample, and normalized units.

- **Part 2: Audio Signal Time-Domain Processing**

We processed an actual audio file by plotting its waveform and identifying individual words (“University”, “of”, “Jaffna”). Using sample indices determined from the waveform, we segmented, played, and saved each word separately. This hands-on experience emphasized the importance of sampling frequency, audio segmentation, and MATLAB’s audio functions for practical signal manipulation.

- **Part 3: Convolution and Filtering in LTI Systems**

We implemented linear time-invariant (LTI) system responses using the conv and filter commands. By comparing these methods and manually coding convolution with loops, we gained deep insights into how filtering operations affect signals. We also explored the significance of initial and final conditions in the filter command, which are essential for block-wise or real-time processing to preserve system memory across signal segments.