Development of MATLAB plug-ins for time and frequency domain analysis

Abstract: A great deal of emphasis has been paid to the features of the speech audio signal in this work. A set of algorithms for recording, sampling, time and frequency domain analysis was created. Sampling algorithm allows to downsample an audio signal. Different time domain characteristics, such as peak values, root mean square (RMS) values, zero crossing rate (ZCR), and autocorrelation, are provided by time domain algorithms. The frequency spectrum and power density spectrum (PSD) were estimated using the direct FFT method and Welch's approach, respectively, to obtain the results. Researchers will be able to have a better understanding of an audio signal and conduct further analysis as a result of these findings.

0 INTRODUCTION

Audio signal has been one of the most valuable and convenient means of communication throughout the human history. Speech communication is not only dependent on face-to-face interaction but also between individuals at anywhere via any technological media. These days, numerous numbers of audio signals have to be passed to a notable distance though devices. So, the equality of the audio signal and effectiveness of communication must be maintained. To uphold these quality and effectiveness, innumerable DSP algorithms have been thrived so far. Understanding the characteristics of the signals being handled is crucial. This work focuses on the creation and testing of two sets of plug-ins that provide signal properties in the temporal and frequency domains. MATLAB was used to examine the RMS value, peak, and Zero Crossing Rate (ZCR) in the time domain, as well as the power spectrum in the frequency domain FFT spectrum.

There are several voiced and unvoiced zones in speech. For speech processing applications such as speech synthesis, speech augmentation, and speech recognition, the categorization of speech signals into voiced and unvoiced gives a preliminary acoustic segmentation. ZCR provides the information about unvoiced zones of the audio signal and how frequently that happens during a particular speech. In the field of audio and acoustics measurement, the "Fast Fourier Transform" (FFT) is a significant measuring method. It breaks down a signal into its various spectral components and hence offers frequency information. FFTs are employed in machine or system defect analysis, quality control, and condition monitoring.

In section 1. Theories related to this work have been thoroughly explained. Focusing on the development of a suite of plugins which has the features both in time and frequency domains, theories related to RMS value, peak value, ZCR and FFT as well as power spectrum have been discussed. In section 2 and 3, methods and results have been explored respectively.

1. THEORY

Theory, applications, and technologies for implementing signal processing system are always benefited from a close coupling. The representation, transformation, and manipulation of signals and the information they contain are all part of signal processing. The quick development of digital computers and microprocessors, as well as certain crucial theoretical advances such as the fast Fourier transform algorithm (FFT), resulted in a substantial transition to digital technologies, resulting in digital signal processing. From now on we will discuss about some theories that are closely related to this work.

1.1 Digitization and resampling

There are reasons why we want to use digital signal over analog signal. Firstly, a programmable digital signal allows flexibility in reconfiguration of that signal. Secondly, digital signals are easy to store in a disk or tape. A digital signal also provides much control of accuracy requirements. Making a signal from analog to digital, we need a analog to digital converter. That analog to digital converter will have some features like sampler, quantizer and coder. Some basic parts of an analog to digital converter have been portrayed in Fig. Sampling associates with taking samples in terms of taking a continuous portion of the signal into a discrete signal. Quantization prefers to the conversion of a discrete time continuous signal into discrete time, discrete valued signal. Finally, in the coding section, each digital signal is assigned with a b-bit binary value.

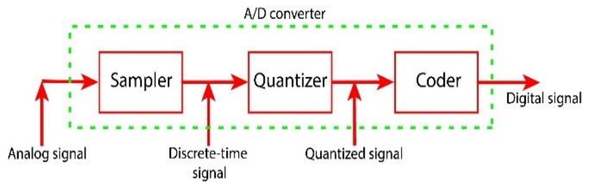


Fig 1: Block diagram of analog to digital signal converter

The Nyquist theorem states that if a sinusoidal function in time or distance is sampled at a frequency higher than or equal to twice every cycle, it may be regenerated with no loss of information [1]. If the waveform is sampled twice as rapidly as its highest frequency component, a bandlimited continuous-time signal can be captured and properly rebuilt from its samples.

The maximum frequency component that can be reliably recorded is the Nyquist limit [1]:

$$f_{max} \leq f_s/2 \qquad (1)$$

"Resampling" is the process of changing the sampling rate by a reasonable factor by combining interpolation and decimation. The most common use of resampling is to connect two systems with differing sampling rates. Decimation or interpolation can be used to adjust the sample rate if the ratio of two systems' rates is an integer; otherwise, interpolation and decimation must be used simultaneously to change the rate. Integer factor decimation: Interpolation is the opposite of K, which involves increasing the sample interval T by an integer factor (or decreasing the sampling frequency).

1.2 Time domain features

We will here discuss about four-time domain features: RMS value, peak value, zero crossing rate and autocorrelation.

1.2.1 RMS value

The RMS value of a signal is calculated as the square root of the signal's average squared value. For a complex valued signal set represented as N discrete sampled values - $[x_o, x_1, \dots x_{N-1}]$, then the mean square x_{RMS} value is given by:

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2}$$

1.2.1 Peak value

Peak value is defined as the highest value achieved by an alternating quantity in a single cycle. Peak amplitude is commonly employed in audio system measurements, telecommunications, and other applications where the measurand is a signal that swings above and below a reference value but is not sinusoidal. If the reference is zero, the peak amplitude is the signal's maximum absolute value; if the reference is a mean value (DC component), the peak amplitude is the difference from that reference's maximum absolute value.

1.2.1 Zero crossing rate

The rate at which a signal transitions from positive to zero to negative or from negative to zero to positive is known as the zero-crossing rate (ZCR). The zero-crossing rate is the number of times the amplitude of voice signals passes through a value of zero in each time interval/frame. Zero crossing rate can be determined as follows [2]:

$$Z_n = \sum_{m=-\infty}^{+\infty} |sgn[x(m)] - sgn[x(m-1)]| \times w(n-m)$$

Where $sgn[x(m)] = 1 \quad ; x(n) > 0$
$\qquad = -1 \quad ; x(n) < 0$

And $w(n)$ is the windowing function with a window size of N samples.

$$\begin{aligned} W &= \frac{1}{2}N \quad ; 0 \leq n \leq N-1 \\ &= 0 \quad ; otherwise \end{aligned}$$

1.2.1 Autocorrelation

Autocorrelation is intended to offer a degree of similarity between a signal and itself at a specified lag. As the crosscorrelation is correlation between two signal and this can be calculated by convolution, one of the sequences is folded, then shifted and multiplied by the other sequence and summing the product. For autocorrelation, the sequence will be convoluted with itself, and it can be given by the following equation [3],

$$\boldsymbol{r_{xx}} = \sum_{\boldsymbol{n}=-\infty}^{\infty} \boldsymbol{x(n)x(n-1)}$$

1.3 Frequency domain features and analysis

Fast Fourier Transform (FFT) is a mathematical procedure used to compute the Discrete Fourier Transform (DFT) of a given pattern. The major difference between FT and FFT is that FT takes a continuous signal as input, whereas FFT takes a discrete signal as input. The discrete time Fourier transform a given sequence is expressed by

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi kn}{N}}$$

On the other hand, FFT works with computational algorithms for the fast execution of DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point, ..., 2r -point, we get the FFT algorithm.
To compute the DFT of an N-point sequence using the equation would take N^2 multiplies and adds. The FFT algorithm computes the DFT using (N log N) multiplies and adds.

Another important feature is estimating the power density spectrum which can be done by Welch's method. To estimate power spectra, Welch's approach (also known as the periodogram method) divides the time signal into sequential blocks, forms the periodogram for each block, and then averages the results. This can be implemented by portioning the sequence the data sequence and for each segment computing the FFT at some frequency, modifying the periodogram value from FFT and averaging the periodogram. [4]

2. METHODS

In this section the methods of using Matlab to develop the plug-ins are described.

2.1. Audio file loading and pre-processing

The first step is to be recording an audio sample and saving it for analysis. The sample audio is recorded by specifying sample rate, sample precision in bits, and number of channels, when capturing sound directly in MATLAB through the following command,

```
recorder = audiorecorder (Fs, nbits, ch);
```

This records with a sample rate of 44100, 8 sample precision bit through the monotype channel for 10 seconds. The recorded speech is stored by the following MATLAB command.

```
x = getaudiodata(recorder);

audiowrite('speech.wav',x,Fs);
```

To import the audio and plot the sample audio [data, fs] = audioread('speach2.wav'); command is used and for plotting the signal amplitude vs time where "t = (0:len-1)*T; " as the data stored in MATLAB is an array where T is inverse sampling frequency and "len" is the length of the array of stored data. As there are too many samples, the segmentation of the data is needed for the analysis. Here, 100 sampled data is taken by the following MATLAB command,

```
data_seg = data(50020:50120);

len_seg = length(data_seg);

t_seg = t(50020:50120);
```

The next is to upsampling the data as the factor to upsampling or downsampling must be an integer value. [3] The upsampling has been done by "up_sam=4*resample(data_seg,L,1);" MATLAB command by a factor of 4 to 176.4 kHz and then downsampled to by factor of 8 to 16.03 kHz through a low pass filter to avoid antialiasing [1]. All the analysis is based on this downsampled data.

2.2. Signal visualization

As mentioned in section 2.1 the recorded sound sample is plotted in amplitude vs time. The segmented data is also plotted in normalized amplitude vs time. The segmented data, upsampled and downsampled data is also plotted in continuous and stem plot. The local peak value is indicated through red dots. The FFT spectrum and power spectrum by Welch method have been showed in the result section.

2.3. Time domain feature calculation

In time domain the local peak value is extracted through the MATLAB command;

```
[pks, locs] = findpeaks(low_data);
```

Where low_data is the array of the downsampled data.
The rms value is also measured from the following equation;

$$\boldsymbol{x_{rms} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}|x_n|^2}} \qquad \textbf{[4]}$$

In MATLAB, there is built in rms function to evaluate the rms value of a given array.

2.4. Frequency domain analysis

For the analysis of segmented sampled data in frequency domain Fast Fourier Transform (FFT) have been implemented. As it is the radix-2 decimation-in-time FFT is the simplest implementation of a speed-optimized DFT. The FFT spectrum is computed by following command.
"ft = fft(low_data, nfft);" Here the "low_data" is the column vector of the sample data and "nfft" is next power of 2 which is calculated from "nfft = 2.^nextpow2(L_seg);".

Another power spectrum estimation has been done by using welch method. In FFT the whole signal is fourier transformed but in welch method the signal is segmented in many parts and by averaging the power of each part of FFT, the spectral power is estimated. The segments are overlapped, where the overlaps are usually 50% or 75% large, and the data within a segment are windowed. **[5]** This reduces the effects of noise and gives more accurate results.

3. RESULTS

In this section, the graphical representation of the implemented MATLAB code and the values calculated in our analysis.

3.1 Raw and resampled Audio

The following figure shows the recorded speech spectrum in time domain. From the figure it can be seen that there is too much information. For this reason, later this signal was segmented in useable length.

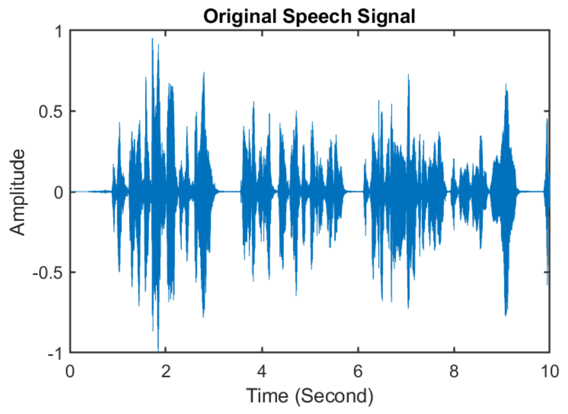


Fig 2: Original speech signal with time

This figure shows a segmented part of the voice signal. It is almost like a sinusoidal signal.

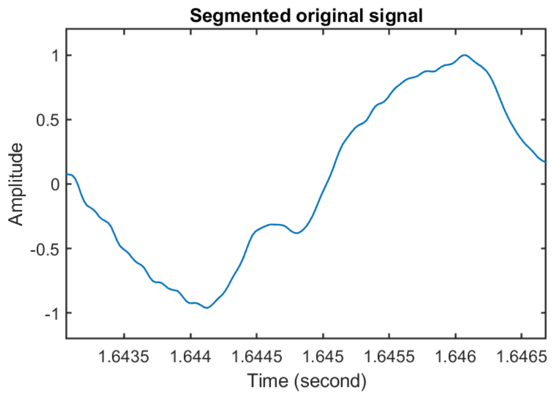


Fig 3: Usual plot of segmented part of original speech signal

Figure 4 shows the sampled signal of the segmented part of voice signal at sample frequency of 44.1 kHz.

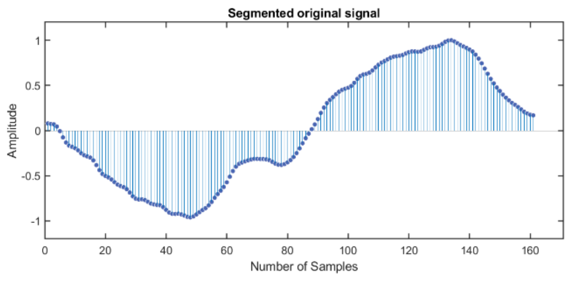
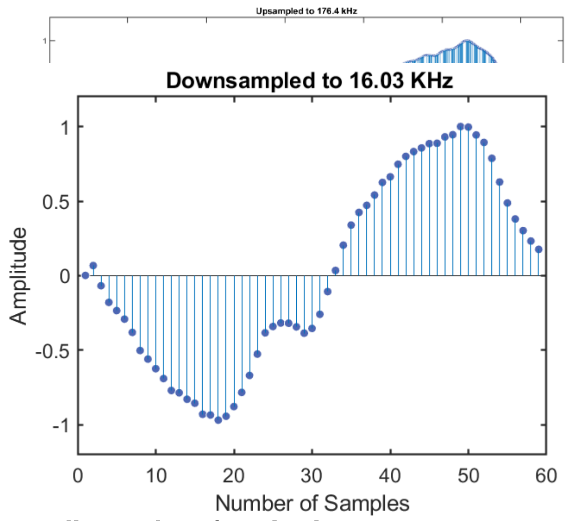


Fig 4: Stem plot of segmented part of the signal

Figure 5 shows the upsampled signal. The time interval between two successive discrete value of time is reduced. Original signal was upsampled by 4 factors. So, sampling frequency of the signal became 176.4 kHz.

Figure 6 shows the downsampled signal at frequency 16.03 kHz. This increases the time interval and thus there



is a smaller number of amplitude point.

Figure 6: Stem plot after downsampling by 11 factors

3.2 Time domain features

The RMS value of the signal was found as 0.0139. Zero crossing rate per second is 551.25. The Table-1 contains the local peak values of the signal which also shows the corresponding times.

Table 1: Local peak values of the signal

Time in sec	Peak value
1.6431	0.0015
1.6446	-0.0069
1.6461	0.0218

Figure 7 shows the downsampled data where local peak values are red marked.

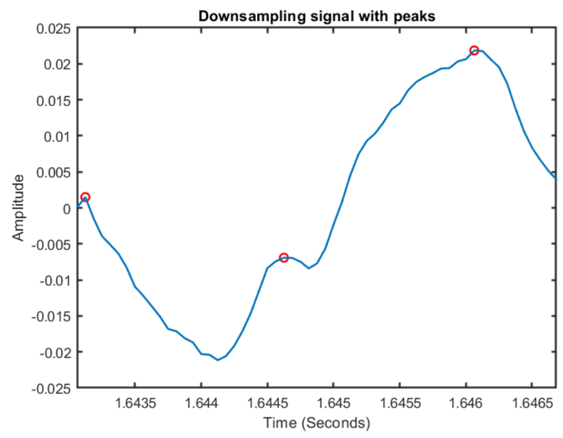


Fig 7: Displaying the local peaks of the signal

Figure 8 shows the autocorrelation of the signal. It has range of indices from -59 to 59.

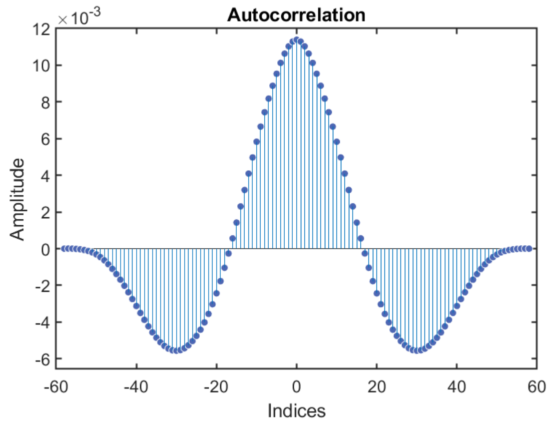


Fig 8: Autocorrelation of the signal

3.3 Frequency domain analysis and representations

Fig. 9 represents the straightforward FFT spectrum. The horizontal axis represents frequency, and the vertical axis is magnitude value. The figure shows that it has a pulse around 9755 Hz which indicates that the segmented input audio has the frequency of 9755 Hz.

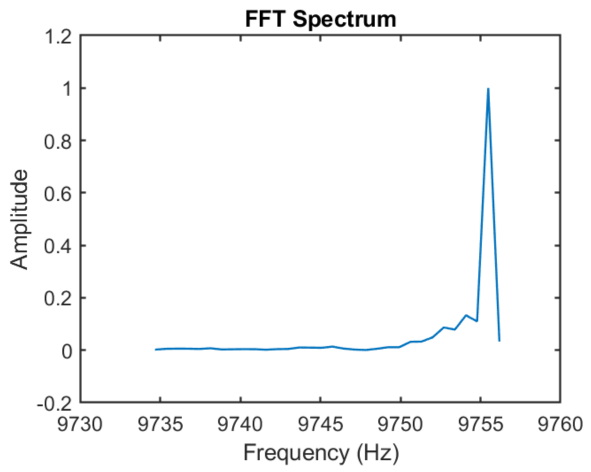


Fig 9: FFT spectrum of the signal

Figure 10 shows the normalized power spectrum density (PSD) using Welch's method. The figure is almost identical to straightforward FFT spectrum but in this case it is smoother.

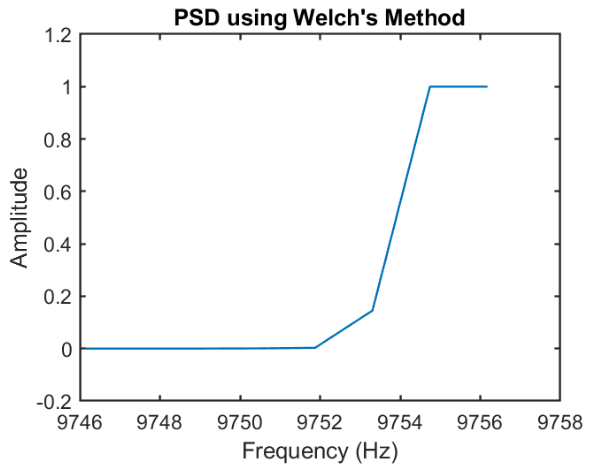


Fig 10: Power spectrum density (PSD) using Welch's method

4. CONCLUSION AND FUTURE WORK

In this work a MATLAB based plug-ins has been developed for time domain and frequency domain analysis. In time domain many important parameters i.e., local peaks, rms can evaluated of an input of voice. Also, upsampling and downsampling of an input can be done easily. In frequency domain the FFT spectra can be extracted, and power spectra density can be estimated more efficiently through this plug-ins.

In future we like to analyze noise and the efficient way to reduce noise and adding more time domain and frequency domain feature in the plug-ins.

References

[1] H. J. Landau, "Sampling, data transmission, and the Nyquist rate," *in Proceedings of the IEEE, vol. 55, no. 10, pp. 1701-1706, Oct. 1967..*

[2] D. S. S. B. P. a. S. P. Shete, " "Zero crossing rate and Energy of the Speech Signal of Devanagari Script."," *IOSR-JVSP 4.1 (2014): 1-5. .*

[3] R. W. S. Alan V Oppenheim, Discrete-Time Signal Processing 3rd Edition.

[4] [Online]. Available: https://nl.mathworks.com/help/signal/ref/rms.html.

[5] P. M. e. a. Djuric, "Spectrum estimation and modeling.", CRC Press, 1999.

APPENDIX

Voice_record.m

```
% this is record an audio signal of
44.1kHz sampling freq.

Fs = 44.1e3;
ch = 1;
datatype = 'unit8';
nbits = 16;
Nseconds = 10;

recorder = audiore-
corder(Fs,nbits,ch);
disp('Start speaking');

recordblocking(recorder,Nseconds);
disp('End of recording');

x = getaudiodata(recorder);
audiowrite('speach.wav',x,Fs);
```

resampling.m

```
function [low_data,new_fs] =
resampling(filename,n1,n2)
% This function will be used to con-
vert the 44.1kHz sampled audio sig-
nal
% to 16kHz.
% filename = path of audio file
% n1 = start point of segment
% n2 = end point of segment
% low_data = data after downsampling
% new_fs = downsampled sampling fre-
quency

%% importing the speech audio
[data, fs] = audioread(filename);
% import the speech audio file
len = length(data);
% length of the data
T = 1/fs;
t = (0:len-1)*T;
% creating time matrix

%% plotting original speech signal
figure('Color','w')
% making figure background color
white
plt1 = plot(t, data/max(abs(data)));
% normalize data plot against time
plt11 = get(plt1,'parent');
set(plt11,'linewidth',1.4,'font-
size',14);%increasing axis linewidth
xlabel('Time (Second)');
ylabel('Amplitude');
title('Original Speech Signal');

%% Segmentation
data_seg = data(n1:n2);     % crop-
ping a segment from original data
len_seg = length(data_seg); % length
of the segment
t_seg = t(n1:n2);           % crop-
ing same time segment for data
%t_seg = t(72400:70560);

% normal plotting
figure('color', 'w');
plt2 = plot(t_seg,
data_seg/max(abs(data_seg)),'Lin-
eWidth',1.4); % plot of the segment
plt22 = get(plt2,'parent');
set(plt22,'linewidth',1.4,'font-
size',14);
title('Segmented original signal');
xlim([min(t_seg) max(t_seg)]);  %
setting x axis limit
ylim([-1.2 1.2])                %
setting y axis limit
xlabel('Time (second)');
ylabel('Amplitude');

% stem plotting
figure('color','w');
plt3 =
stem(data_seg/max(abs(data_seg)),'fi
lled', ...
'MarkerEdgeColor','white','Marker-
FaceColor',[0.3 0.4 0.7]); % normal-
ize plot
plt33 = get(plt3,'parent');
```

```
set(plt33,'linewidth',1.4,'font-
size',14)
xlim([0 len_seg+10]);
ylim([ -1.2 1.2]);
title('Segmented original signal')
xlabel('Number of Samples');
ylabel('Amplitude');

%% Upsampling
L = 4;
up_sam = resample(data_seg,L,1);  %
upsampling by L factor

figure('color','w');
plt4 =
stem(up_sam/max(abs(up_sam)),'filled
', ...
    'MarkerEdgeColor','white','Mark-
erFaceColor',[0.3 0.4 0.7]);
plt44 = get(plt4,'parent');
set(plt44,'linewidth',1.4,'font-
size',14)
xlim([ 1 length(up_sam)+10]);
ylim([-1.2 1.2]);                  %
setting y axis limit
title('Upsampled to 176.4 kHz');
xlabel('Number of Samples');
ylabel('Amplitude');

%% Downsampling with decimator
M = 11;
low_filt = lowFilt(up_sam,M,1,5);
% passing to a low pass filter
low_data =
low_filt([1:M:length(low_filt)]); %
downsampling by M factor
new_fs = fs*L/M;
% new sampling frequency

figure('color','w');
plt5 =
stem(low_data/max(abs(low_data)),'fi
lled', ...
    'MarkerEdgeColor','white','Mark-
erFaceColor',[0.3 0.4 0.7]);
plt55 = get(plt5,'parent');
set(plt55,'linewidth',1.4,'font-
size',14)
title('Downsampled to 16.03 KHz ')
ylim([-1.2 1.2])
xlabel('Number of Samples');
ylabel('Amplitude');

%% Save as .mat file
save('low_data.mat','low_data','t_se
g');

%% function for low pass filter
function [out] = lowFilt(input,fac-
tor,gain,order)
    h = designfilt('lowpass-
fir','FilterOrder',order,'Cut-
offFrequency',1/factor);
    out = gain*filter(h,input);
end
end
```

execute_resamplig.m

```
clc;clear all;
n1 = 72460; % start point of segment
n2 = 72620; % end point of segment
[low_data,new_fs]=resampling('speach
.wav',n1,n2);
```

timeDomain.m

```
function [pks,locs,RMS,zcr,autocorr]
= timeDomain(low_data,t_seg)
% This function will show the time
domain features like Peak values,
%RMS, ZRC values of the input data
%   Detailed explanation goes here
% low_data = downsampled data
% t_seg = time segment of the data
% pks = local peak values
% locs = location of local peak val-
ues
% RMS = rms value of the signal
%% Finding local peaks
indices = 1:length(low_data);     %
creating index matrix
[pks, locs] = findpeaks(low_data);%
finding local peaks and locations
low_dt = lin-
space(min(t_seg),max(t_seg),length(l
ow_data)); % creating time matrix
pks_locs = [];                    %
assigning locations of the peaks ma-
trix
% creating locations of the peaks
matrix
for i = 1: length(locs)
    loc = locs(i);
    pks_loc = low_dt(loc);
% getting the location
    pks_locs = [pks_locs pks_loc ];
% storing in pks_locs matrix
end
% showing plot with peaks
figure()
plot(pks_locs,pks,'or','Lin-
eWidth',1.4);        % peaks plot
hold on;
plt8 = plot(low_dt,low_data,'Lin-
eWidth',1.4);   % low_data plot
```

```
xlim([min(low_dt) max(low_dt)]);
plt88 = get(plt8,'parent');
set(plt88,'linewidth',1.4);
xlabel('Time (Seconds)');
ylabel('Amplitude');
title('Downsampling signal with
peaks');
% displaying the result
disp('Times of local peaks');
disp(pks_locs);
disp('Values of local peaks');
disp(pks);

%% Finding RMS of the signal
RMS = rms(low_data); % rms value
X = sprintf('RMS value %0.4f',RMS);
% taking upto 4 point float
disp(X);

%% ZCR (zero crossing rate)
%zcr = mean(abs(diff(sign(Signal)));
zero_crossing =
sum(abs(diff(low_data>0)));
T = max(t_seg)-min(t_seg);
zcr = zero_crossing/T;
Y = sprintf('Zero crossing rate
%0.4f',zcr);
disp(Y);

%% Autocorrelation Calculation
[row,col] =size(low_data);  % get-
ting no of row & col from low_data
matrix
if ( row>col)
    x = transpose(low_data);% con-
verting into row matrix
else
    x = low_data;
end

h = fliplr(x);      % creating 2nd
matrix for autocorrelation
n1 = 0:length(x)-1; % generating in-
dices for x data
n2 = -fliplr(n1);   % generating in-
dices for h data
z=[]; % assigning
for i=1:length(x)
    g=h.*x(i);      % multiplication
with each h component
    z=[z;g];        % storing data
end

[r, c]=size(z);     %getting no of
row & col from z matrix
k = r+c;
t = 2;
autocorr = [];
cd = 0;
% diagonal additon
while(t<=k)
    for i=1:r
        for j=1:c
            if((i+j)==t)     % get-
ting diagonal elements
                cd=cd+z(i,j); % diag-
onal addition
            end
        end
    end
    t=t+1;
    autocorr=[autocorr cd];  % stor-
ing autocorrelation data in each
step
    cd=0;                    % re-
turn to zero
end

nl=min(n1)+min(n2);          % low
point on autocorrelation indices
nh=max(n1)+max(n2);          % high
point on autocorrelation indices
idx=nl:1:nh;                 % gen-
erating indices
%plotting
figure('color','w');
p = stem(idx,autocorr,'filled', ...
'MarkerEdgeColor','white','Marker-
FaceColor',[0.3 0.4 0.7]);
g = get(p,'Parent');
set(g,'LineWidth',1.4,'Font-
Size',14);
ylim([min(autocorr)-1e-3 12e-3]);
xlabel('Indices');
ylabel('Amplitude');
title('Autocorrelation');

end
```

execute_timeDomain.m

```
clc;clear all;
low = load('low_data.mat');
low_data = low.low_data;
t_seg = low.t_seg;
[pks,locs,RMS,zcr,autocorr]=timeDo-
main(low_data,t_seg);
```

fftSpectrum.m

```
function [ft2,xfft2] = fftSpec-
trum(low_data,t_seg,fs)
% This function will show the direct
FFT spectrum
%   Detailed explanation goes here
%% FFT Implementation
L_seg = length(low_data);   % length
of the input data
```

```
nfft = 2.^nextpow2(L_seg);  % con-
verting to 2th power
down_fs = fs;               % sam-
pling freq
ft = fft(low_data, nfft);   % nfft
point fft
ft2 = ft(1:nfft/2)/nfft;    % con-
verting to one-sided fft

%% Frequency Matrix calculation
dt = lin-
space(min(t_seg),max(t_seg),nfft/2);
% creating time matrix
xfft = down_fs*1./dt;       % con-
verting to frequency matrix
xfft2 = transpose(xfft);    % trans-
posing the matrix

%% FFT spectrum plot
figure('color','w'); % white figure
background
plt =
plot(xfft2,abs(ft2)/max(abs(ft2)),'L
ineWidth',1.4); % normalize plot
plt2 = get(plt,'parent');
set(plt2,'linewidth',1.4,'font-
size',14);
ylim([-0.2 1.2]);
title('FFT Spectrum');
ylabel('Amplitude');
xlabel('Frequency (Hz)');

end
```

execute_fftSpectrum.m

```
clc;clear all;
low = load('low_data.mat');
low_data = low.low_data;
t_seg = low.t_seg;
fs = 16.03e3;
[ft,freq]=fftSpec-
trum(low_data,t_seg,fs);
```

WelchPSD.m

```
function [Px,freq] =
WelchPSD(low_data,t_seg,fs)
% This function will generete power
spectrum density (PSD) using
% Welch's method
% Px = power spectrum density
%freq = frequency matrix
% low_data = input data
% t_seg = time segment data
% fs = sampling frequency
%% Defining parameter
L = length(low_data);   % length of
the input data
overlap = 0.5;          % 50% over-
lap between windows
L_seq = 10;             % window
length
n1 = 1;                 % start
point of the window
n2 = L_seq;             % end point
of the window
n0 = (1-overlap)*L_seq; % increment
in step
nsubseq = 1+floor((L-L_seq)/n0); %
number of subsection window will run
Px = 0;                 % assigning
PSD variable

%% Running a window over total input
data
for i = 1:nsubseq
   Px = Px + modified_periodo-
gram(low_data,n1,n2)/nsubseq; % tak-
ing psd for each step
   n1 = n1+n0; % increasing start
point
   n2 = n2+n0; % increasing end
point
end

%% One-sided PSD calculation
len_Px = length(Px);
Px_one_sided = Px(1:len_Px/2);
%cropping the one side
%% Frequency matrix calculation
dt = lin-
space(min(t_seg),max(t_seg),len_Px);
% creating time matrix
dt = transpose(dt);
%transposing
freq = fs*1./dt;
% converting to frequency matrix
freq = freq(1:len_Px/2);
% compatable for one-sided fft

%% PSD plot
figure('color','w');
plt =
plot(freq,Px_one_sided/max(Px_one_si
ded),'linewidth',1.4); %normaliza-
tion
plt2 = get(plt,'parent');
set(plt2,'linewidth',1.4,'font-
size',14);
xlabel('Frequency (Hz)');
ylabel('Amplitude');
title("PSD using Welch's Method");
ylim([-0.2 1.2]);

%% PSD function
function Px = modified_periodo-
gram(x,n1,n2)
```

```
N = n2-n1+1;        % length of the
window
w = hanning(N);     % hanning window
for tapering the input window data
U = norm(w)^2/N;    % normalization
of hanning window
xw = x(n1:n2).*w;   % tapering with
hanning window
nfft = 2.^nextpow2(length(xw));
Px = abs(fft(xw,nfft)).^2/(N*U); %
PSD calculation
Px(1) = Px(2);
end
end
```

execute_WelchPSD.m

```
clc;clear all;
low = load('low_data.mat');
low_data = low.low_data;
t_seg = low.t_seg;
fs = 16.03e3;
[Px,freq]=WelchPSD(low_data,t_seg,fs
);
```