SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Constituent of Symbiosis International (Deemed University), Pune

Department of Electronics and Telecommunication Engineering

Speech and Audio Signal Processing Lab Manual

Department of Electronics and Telecommunication Engineering

Vision. Mission and PEOs of E&TC Department

Department Vision:

To emerge as a leading source for Electronics and Telecommunication engineering, fostering globally proficient engineers to meet the demands of evolving industry and society.

Department Mission:

1. Foster collaboration with industry to facilitate the acquisition of cutting-edge technologies and contribute to the generation of up-to-date knowledge, enhancing employability and sustainability.
2. Encourage innovation, research, and development, creating an environment conducive to higher education, entrepreneurship, and lifelong learning.
3. Cultivate leadership qualities infused with social and ethical values, providing a platform for their development.

Program Educational Objectives (PEOs)	
PEO1	Graduates will possess a strong foundation in science and engineering fundamentals, along with analytical skills to effectively solve real-world problems.
PEO2	Graduates will gain technical proficiency in Electronics and Telecommunication fields and scale new heights in the profession through lifelong learning.
PEO3	Graduates will embrace professionalism, ethical conduct at all levels and constantly evolve in a multidisciplinary approach leading towards sustainability.
PEO4	Graduates will leverage their engineering knowledge, effective communication skills, leadership qualities, and teamwork spirit to serve society and contribute positively to their community.

List of Experiment

Experiment No.	Details	Hours
1	Record speech signal and find Energy and ZCR for different frame rates and comment on the result.	**2**
2	Record different vowels as /a/, /e/, /i/, /o/ etc. and extract the pitch as well as first three formant frequencies. Perform similar analysis for different types of unvoiced sounds and comment on the result.	**2**
3	Write a program to identify voiced, unvoiced and silence regions of the speech signal.	**2**
4	Record a speech signal and perform the spectrographic analysis of the signal using wideband and narrowband spectrogram. Comment on narrowband and wide band spectrogram.	**2**
5	Write a program for extracting pitch period for a voiced part of the speech signal using autocorrelation.	**2**
6	Write a program to design a Mel filter bank and using this filter bank write a program to extract MFCC features.	**2**
7	Write a program to perform the cepstral analysis of speech signal and detect the pitch from the voiced part using cepstrum analysis.	**2**
8	Write a program to find LPC coefficients using Levinson Durbin algorithm.	**2**
9	Write a program to enhance the noisy speech signal using spectral subtraction method.	**2**
10	Write a program to extract frequency domain audio features like SC, SF and Spectral roll off	**2**
11	Minor Project	**10**

EXPERIMENT NO: 1

Experiment Name: Record speech signal and find Energy and ZCR for different frame rates.
.

Objective: To compute short-time energy and zero-crossing rate (ZCR) for a recorded speech signal at different frame sizes.
Theory: Energy represents amplitude strength, and ZCR is the number of zero crossings in a frame—used for voiced/unvoiced classification.
Procedure:

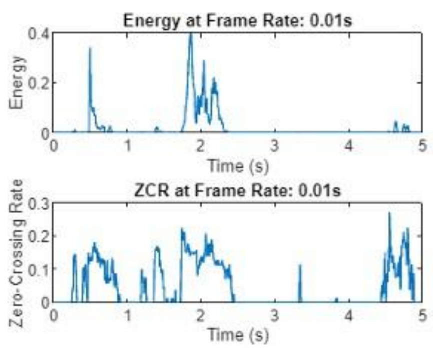
- Record speech
- Divide into frames
- Compute energy and ZCR
- Plot and compare results

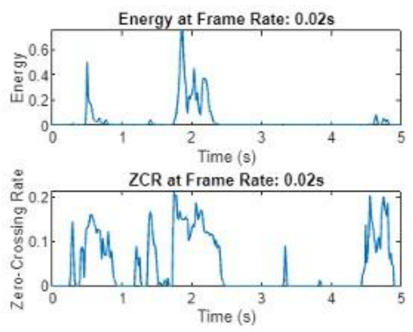
Result: Plots showing variation in energy and ZCR for different frame rates.

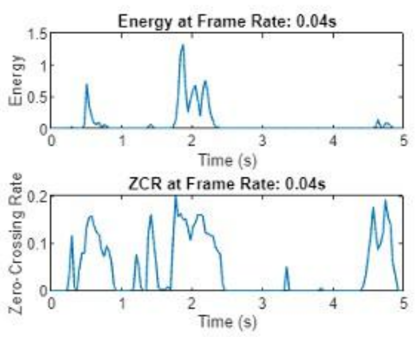
MATLAB Code

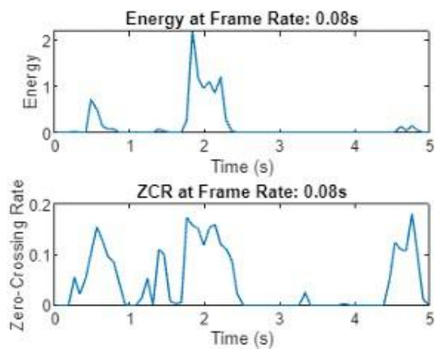
```
% 1. Record Speech Signal
recObj = audiorecorder;
disp('Start speaking...');
recordblocking(recObj, 5); % Record for 5 seconds
disp('End of Recording.');
y = getaudiodata(recObj);
Fs = recObj.SampleRate;

% 2. Define Frame Rates and Parameters
frameRates = [0.01, 0.02, 0.04, 0.08]; % Frame rates in seconds (e.g., 10ms, 20ms, 40ms,
80ms)
windowLength_s = frameRates;
overlapLength_s = 0.005; % 5ms overlap for all frame rates
numFrames = length(frameRates);
energy = cell(1,numFrames);
zcr = cell(1,numFrames);

for i = 1:numFrames
    % 3. Frame the Signal
    windowLength = round(windowLength_s(i) * Fs);
    overlapLength = round(overlapLength_s * Fs);
    hopLength = windowLength - overlapLength;
    [frames, ~] = buffer(y, windowLength, overlapLength, 'nodelay');

    % 4. Calculate Energy and ZCR for each frame
    numFramesProcessed = size(frames, 2);
    energy{i} = zeros(1, numFramesProcessed);
    zcr{i} = zeros(1, numFramesProcessed);

    for j = 1:numFramesProcessed
        frame = frames(:, j);
        energy{i}(j) = sum(frame.^2);
        zcr{i}(j) = zerocrossrate(frame);
    end

    % 5. Plot Results
    t = (0:numFramesProcessed-1) * (windowLength_s(i) - overlapLength_s) +
windowLength_s(i)/2; % Calculate time vector for plotting

    figure;
    subplot(2,1,1);
    plot(t, energy{i});
    title(['Energy at Frame Rate: ', num2str(frameRates(i)), 's']);
    xlabel('Time (s)');
    ylabel('Energy');

    subplot(2,1,2);
    plot(t, zcr{i});
    title(['ZCR at Frame Rate: ', num2str(frameRates(i)), 's']);
    xlabel('Time (s)');
    ylabel('ZCR');
end

% 6. Analyze Results
% Compare the plots for different frame rates.
% You can observe how the energy and ZCR patterns change with varying frame rates.
% For example, shorter frame rates might capture more detail in the signal, while longer
frame rates might smooth out the fluctuations.
```

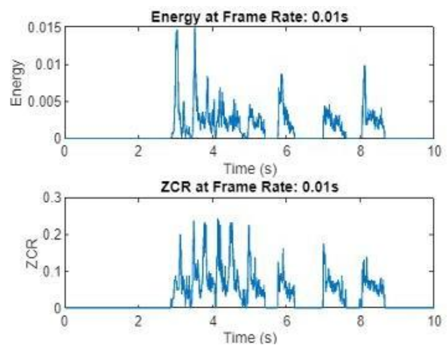
Results

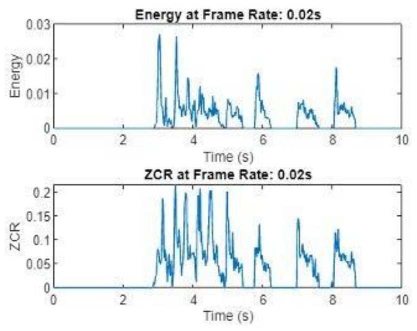


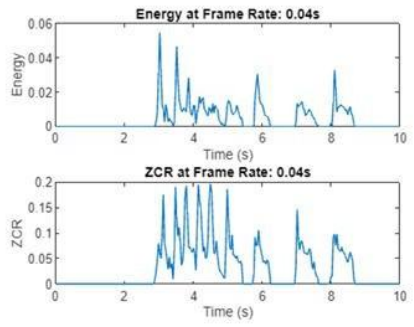


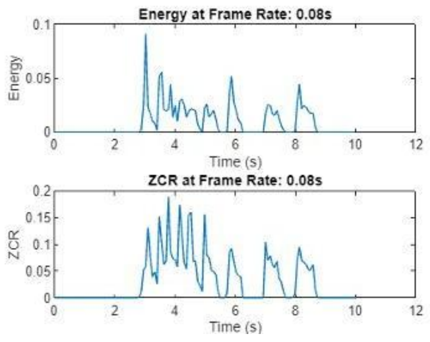












EXPERIMENT NO: 2

Experiment Name: Record vowels and extract pitch and formants. Analyze unvoiced sounds.
Objective: To extract pitch and first three formants for vowels and analyze unvoiced sounds.
Theory: Pitch is the fundamental frequency. Formants are resonant frequencies important for vowel identification.
Procedure:

- Record vowels
- Estimate pitch and formants
- Repeat for unvoiced sounds

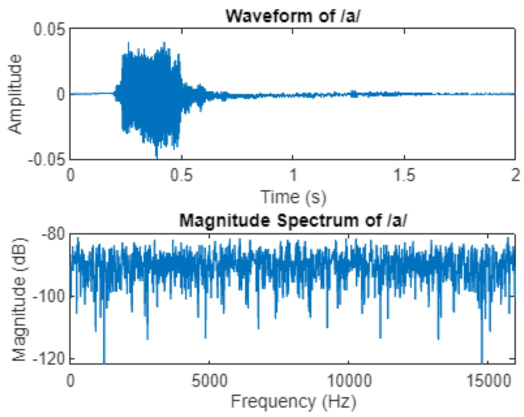
Result: Formant tables and pitch plots for vowels and unvoiced sounds.

MATLAB Code

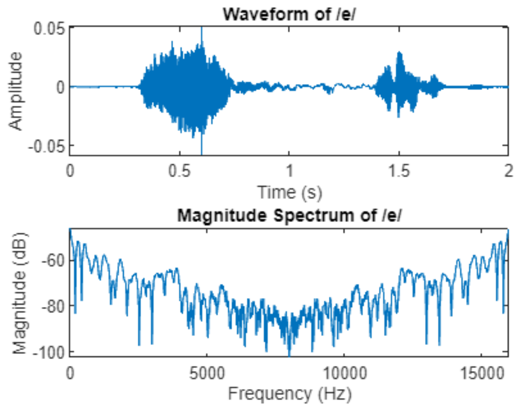
```
clc; clear; close all;

% Define labels and number of sounds
labels = {'a','e','i','o','u','s','sh','f'};  % Vowels and unvoiced consonants
numSounds = length(labels);

% Sampling parameters
Fs = 16000; % 16 kHz
duration = 2; % Record 2 seconds for each sound

% Initialize result storage
pitchValues = zeros(1, numSounds);
formants = zeros(numSounds, 3); % Store F1, F2, F3
```

```
for k = 1:numSounds
    fprintf('Please pronounce /%s/ and wait...\n', labels{k});
    pause(1);

    recObj = audiorecorder(Fs, 16, 1);
    recordblocking(recObj, duration);
    fprintf('Recording for /%s/ done.\n', labels{k});

    y = getaudiodata(recObj);
    y = y .* hamming(length(y)); % Windowing to reduce edge effects

    % Plot waveform
    figure;
    subplot(2,1,1);
    plot((1:length(y))/Fs, y);
    title(['Waveform of /', labels{k}, '/']);
    xlabel('Time (s)');
    ylabel('Amplitude');

    % Estimate pitch using autocorrelation
    pitch = estimatePitch(y, Fs);
    pitchValues(k) = pitch;

    % Estimate formants using LPC
    ncoeff = 12; % Order of LPC
    a = lpc(y, ncoeff);
    rts = roots(a);
    rts = rts(imag(rts) >= 0.01); % Keep positive frequency roots only

    angz = atan2(imag(rts), real(rts));
    formantFreqs = sort(angz * (Fs / (2 * pi)));

    % Store first 3 formants if they exist
    formants(k,1:min(3,length(formantFreqs))) = formantFreqs(1:min(3,end));

    % Plot spectrum
    subplot(2,1,2);
    nfft = 1024;
    Y = abs(fft(y, nfft));
    f = (0:nfft-1) * Fs / nfft;
    plot(f, 20*log10(Y));
    title(['Magnitude Spectrum of /', labels{k}, '/']);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude (dB)');
end

% Display results
fprintf('\nResults:\n');
fprintf('Sound\tPitch(Hz)\tF1(Hz)\t\tF2(Hz)\t\tF3(Hz)\n');
for k = 1:numSounds
    fprintf('/%s/\t%.2f\t\t%.2f\t%.2f\t%.2f\n', ...
        labels{k}, pitchValues(k), formants(k,1), formants(k,2), formants(k,3));
end
function pitch = estimatePitch(signal, Fs)
    signal = signal - mean(signal); % Remove DC
    autocorr = xcorr(signal);
    autocorr = autocorr(length(signal):end); % Take right half
    [~, locs] = findpeaks(autocorr, 'MinPeakDistance', round(Fs/400));
    if length(locs) >= 2
        pitchPeriod = locs(2) - locs(1);
        pitch = Fs / pitchPeriod;
    else
        pitch = 0; % Likely unvoiced
    end
end
```

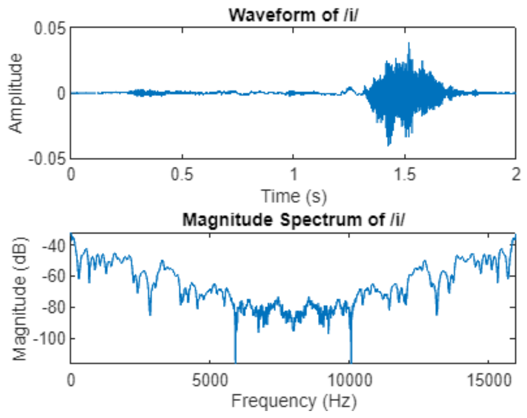
```
Please pronounce /a/ and wait...
Recording for /a/ done.
```



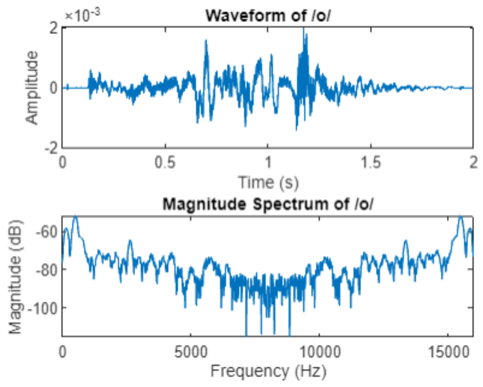
```
Please pronounce /e/ and wait...
Recording for /e/ done.
```



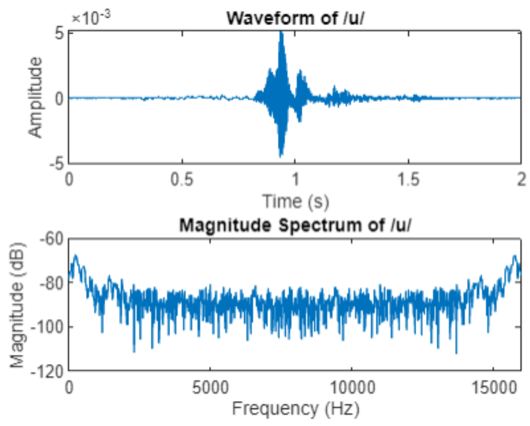
```
Please pronounce /i/ and wait...
Recording for /i/ done.
```



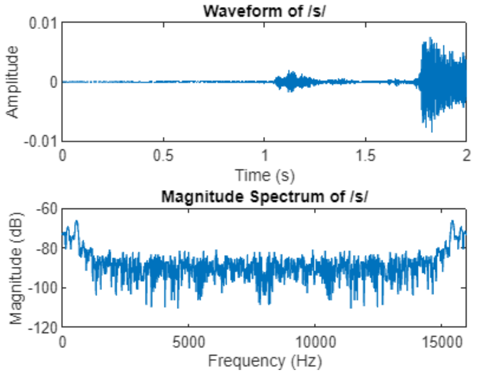
```
Please pronounce /o/ and wait...
Recording for /o/ done.
```



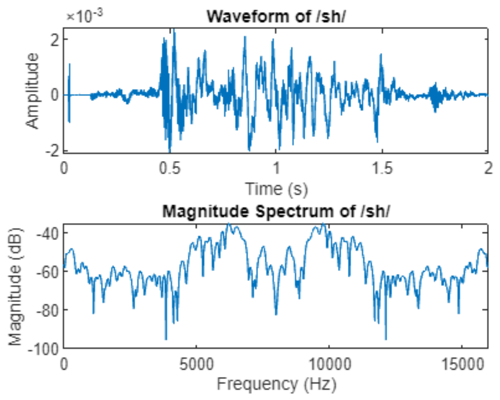
```
Please pronounce /u/ and wait...
Recording for /u/ done.
```



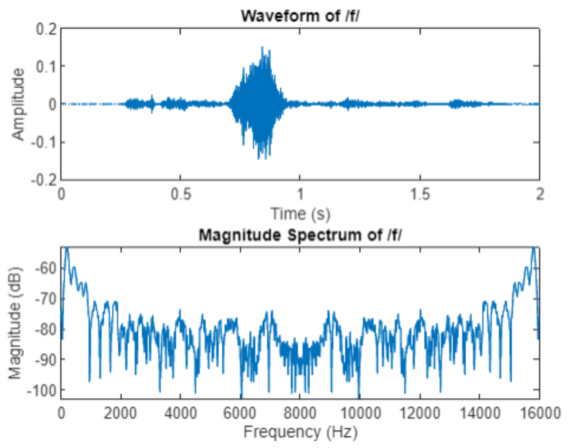
```
Please pronounce /s/ and wait...
Recording for /s/ done.
```



```
Please pronounce /sh/ and wait...
Recording for /sh/ done.
```



```
Please pronounce /f/ and wait...
Recording for /f/ done.
```



```
Results:
Sound   Pitch(Hz)        F1(Hz)           F2(Hz)           F3(Hz)
/a/     231.88           415.30  2947.46 3485.94
/e/     390.24           379.39  3193.24 3759.56
/i/     222.22           436.67  1766.06 3141.76
/o/     326.53           1703.25 3383.23 5229.73
/u/     290.91           293.56  1707.31 3110.50
/s/     372.09           404.33  1836.97 3856.86
/sh/    340.43           696.01  2898.25 4790.34
/f/     313.73           486.30  2943.74 3815.06
```

EXPERIMENT NO: 3

Experiment Name: Identify voiced, unvoiced, and silence regions.
Objective: To classify segments of speech into voiced, unvoiced, and silence.
Theory: Voiced: periodic, high energy; Unvoiced: aperiodic, high ZCR; Silence: low energy and ZCR.
Procedure:

- Frame signal
- Compute ZCR and energy
- Classify using thresholds
- Plot result

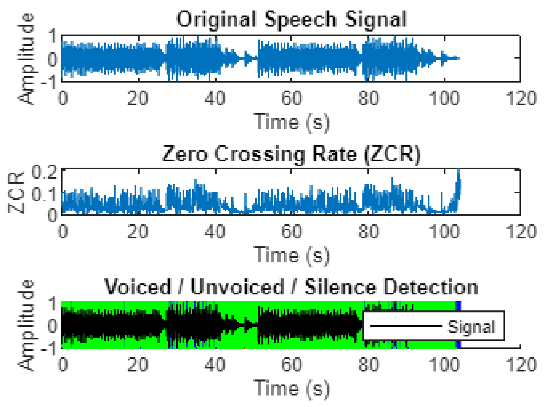
Result: Annotated signal showing segmentation.

MATLAB Code

```
% Read speech signal
[signal, Fs] = audioread('guitar.wav');    % Replace 'speech.wav' with your file
signal = signal(:, 1);                      % Use one channel if stereo
signal = signal - mean(signal);             % DC removal
signal = signal / max(abs(signal));         % Normalization

% Parameters
frameDuration = 0.02;                       % 20 ms frames
frameLength = round(frameDuration * Fs);
overlap = round(0.5 * frameLength);         % 50% overlap
hopLength = frameLength - overlap;
```

```matlab
% Initialize
numFrames = floor((length(signal) - frameLength) / hopLength) + 1;
energy = zeros(1, numFrames);
zcr = zeros(1, numFrames);
labels = strings(1, numFrames);

% Compute short-time energy and ZCR
for i = 1:numFrames
    startIdx = (i - 1) * hopLength + 1;
    frame = signal(startIdx : startIdx + frameLength - 1);

    energy(i) = sum(frame .^ 2);                          % Short-time energy
    zcr(i) = sum(abs(diff(sign(frame)))) / (2*length(frame));  % ZCR
end

% Thresholds (you can tune these for different recordings)
energyThreshold = 0.02;     % Empirical energy threshold
zcrVoiced = 0.1;            % ZCR upper bound for voiced
zcrSilence = 0.02;          % ZCR lower bound for silence

% Classify each frame
for i = 1:numFrames
    if energy(i) < energyThreshold && zcr(i) < zcrSilence
        labels(i) = "Silence";
    elseif zcr(i) < zcrVoiced
        labels(i) = "Voiced";
    else
        labels(i) = "Unvoiced";
    end
end

% Plot results
timeAxis = (0:numFrames-1) * hopLength / Fs;

figure;
subplot(3,1,1);
plot((1:length(signal))/Fs, signal);
title('Original Speech Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot(timeAxis, energy);
title('Short-Time Energy');
xlabel('Time (s)');
ylabel('Energy');

subplot(3,1,3);
plot(timeAxis, zcr);
title('Zero Crossing Rate (ZCR)');
xlabel('Time (s)');
ylabel('ZCR');

% Display detected regions
figure;
hold on;
for i = 1:numFrames
    switch labels(i)
        case "Voiced"
```

```
            color = 'g';
        case "Unvoiced"
            color = 'b';
        case "Silence"
            color = 'r';
    end
    rectangle('Position', [timeAxis(i), -1, frameDuration, 2], 'FaceColor', color,
'EdgeColor', 'none');
end
plot((1:length(signal))/Fs, signal, 'k');
title('Detected Speech Regions');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Voiced (Green)', 'Unvoiced (Blue)', 'Silence (Red)');
hold off;
```

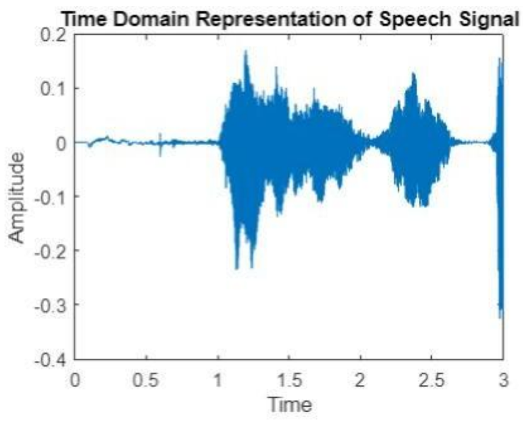


EXPERIMENT NO: 4

Experiment Name: Spectrographic analysis using wideband and narrowband spectrograms.
Objective: To analyze time-frequency properties using spectrograms.
Theory: Wideband: good time resolution; Narrowband: good frequency resolution.
Procedure:

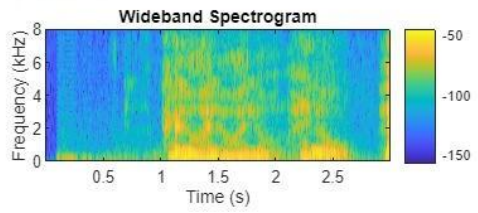
- Record signal
- Compute spectrograms
- Compare plots

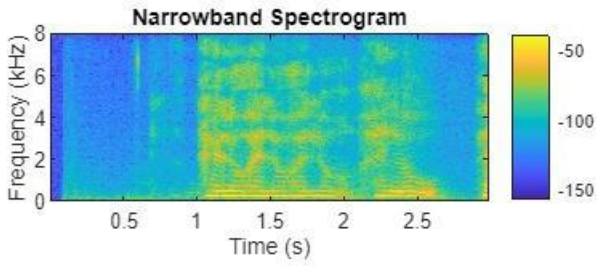
Result: Spectrogram plots with commentary.



```
overlapNB = 360
```

```
overlapWB = 24
```





EXPERIMENT NO: 5

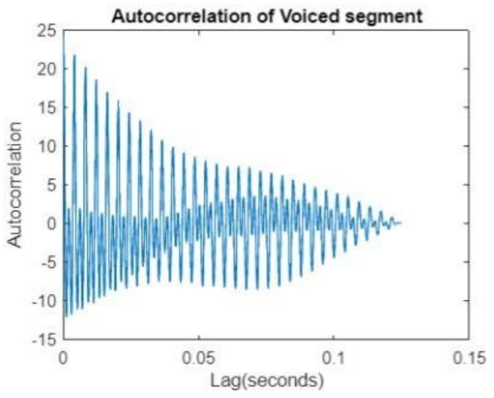
Experiment Name: Pitch extraction using autocorrelation.
Objective: To determine pitch using autocorrelation.
Theory: Autocorrelation emphasizes periodicity in voiced signals.
Procedure:

- Isolate voiced frame
- Compute autocorrelation
- Find peak and estimate pitch

Result: Estimated pitch and autocorrelation plot.

MATLAB Code

```
clc;
clear;

%% Read or Record a Speech Signal
recObj = audiorecorder;

recordblocking(recObj, 3);
signal = getaudiodata(recObj);
Fs = recObj.SampleRate;

%% Preprocess the signal
signal = signal - mean(signal);      % Remove DC component
signal = signal / max(abs(signal));  % Normalize amplitude

%% Select a short voiced segment (optional: manually define for more accuracy)
startSample = 5000;
endSample = 6000;
segment = signal(startSample:endSample);

%% Autocorrelation
autoCorr = xcorr(segment);                  % Full autocorrelation
autoCorr = autoCorr(length(segment):end);  % Keep only non-negative lags

%% Plot autocorrelation
figure;
lag = (0:length(autoCorr)-1) / Fs;
plot(lag, autoCorr);
xlabel('Lag (seconds)');
ylabel('Autocorrelation');
title('Autocorrelation of Voiced Segment');
grid on;

%% Find Pitch Period from Autocorrelation Peaks
minF0 = 70;   % Minimum expected pitch frequency in Hz
maxF0 = 400;  % Maximum expected pitch frequency in Hz
minLag = round(Fs / maxF0);
maxLag = round(Fs / minF0);

[peaks, locs] = findpeaks(autoCorr(minLag:maxLag));
if ~isempty(locs)
    pitchLag = locs(1) + minLag - 1;    % Actual lag corresponding to pitch
    pitchPeriod = pitchLag / Fs;       % In seconds
    pitchFreq = 1 / pitchPeriod;       % In Hz

    fprintf('Estimated Pitch Lag: %d samples\n', pitchLag);
    fprintf('Estimated Pitch Period: %.4f seconds\n', pitchPeriod);
    fprintf('Estimated Pitch Frequency: %.2f Hz\n', pitchFreq);
else
    fprintf('No clear pitch peak found in the given range.\n');
end
```



```
Estimated Pitch Lag: 34 samples
Estimated Pitch Period: 0.0043 samples
Estimated Pitch Frequency: 235.29 samples
```

EXPERIMENT NO: 6

Experiment Name: Design Mel filter bank and extract MFCCs.
Objective: To implement MFCC feature extraction using a Mel filter bank.
Theory: MFCCs mimic human auditory perception and involve filter bank, log, and DCT.
Procedure:

- Pre-emphasize
- Apply FFT and filter bank
- Log and DCT to get MFCCs

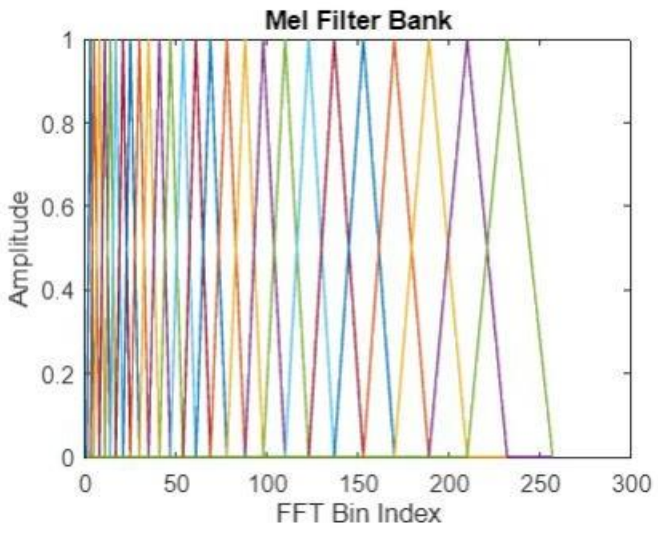
Result: MFCC plot for speech.

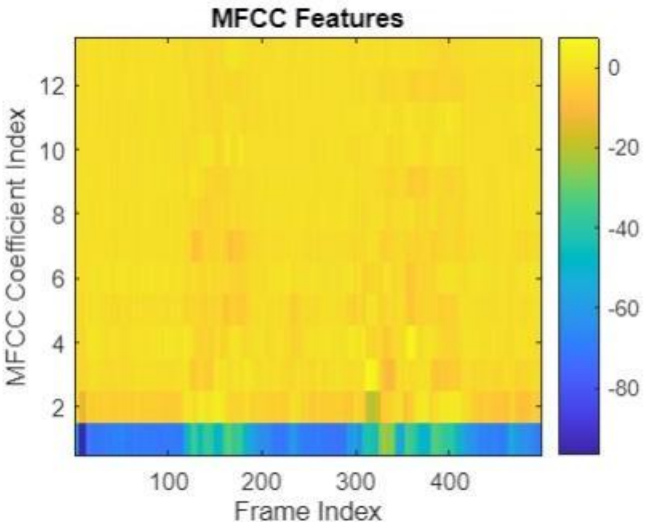
MATLAB Code

```
clc;
clear;
close all;

%% Step 1: Read Audio
[audio, Fs] = audioread('speech.wav');  % Use your own file
audio = audio(:, 1);                     % Mono
audio = audio - mean(audio);             % Remove DC offset
audio = audio / max(abs(audio));         % Normalize

%% Step 2: Pre-emphasis
preEmph = 0.97;
emphasized = filter([1 -preEmph], 1, audio);

%% Step 3: Framing
frameSize = 0.025; % 25 ms
frameStep = 0.010; % 10 ms
```

```
frameLength = round(frameSize * Fs);
frameStepSamples = round(frameStep * Fs);

numFrames = floor((length(emphasized) - frameLength) / frameStepSamples) + 1;
frames = zeros(numFrames, frameLength);

for i = 1:numFrames
    startIdx = (i-1) * frameStepSamples + 1;
    frames(i, :) = emphasized(startIdx : startIdx + frameLength - 1);
end

%% Step 4: Windowing (Hamming)
hammingWin = hamming(frameLength)';
frames = frames .* hammingWin;

%% Step 5: FFT and Power Spectrum
NFFT = 512;
magFrames = abs(fft(frames, NFFT, 2));
powFrames = (1 / NFFT) * (magFrames .^ 2);

%% Step 6: Design Mel Filter Bank
numFilters = 26;  % Number of mel filters
lowFreq = 0;
highFreq = Fs / 2;

% Convert Hz to Mel
hz2mel = @(hz) 2595 * log10(1 + hz / 700);
mel2hz = @(mel) 700 * (10.^(mel / 2595) - 1);

lowMel = hz2mel(lowFreq);
highMel = hz2mel(highFreq);

% Equally spaced points in Mel scale
melPoints = linspace(lowMel, highMel, numFilters + 2);
hzPoints = mel2hz(melPoints);

% Convert Hz to FFT bin numbers
bin = floor((NFFT + 1) * hzPoints / Fs);

% Initialize filter bank
fbank = zeros(numFilters, NFFT/2 + 1);

for m = 2:numFilters+1
    f_m_minus = bin(m - 1); % left
    f_m = bin(m);            % center
    f_m_plus = bin(m + 1);  % right

    for k = f_m_minus:f_m
        fbank(m-1, k+1) = (k - bin(m-1)) / (bin(m) - bin(m-1));
    end
    for k = f_m:f_m_plus
        fbank(m-1, k+1) = (bin(m+1) - k) / (bin(m+1) - bin(m));
    end
end

%% Step 7: Apply Filter Bank to Power Spectrum
filterBankEnergies = powFrames(:, 1:NFFT/2+1) * fbank';
```

```matlab
% Avoid log(0)
filterBankEnergies(filterBankEnergies == 0) = eps;

%% Step 8: Log Energies
logEnergies = log(filterBankEnergies);

%% Step 9: Discrete Cosine Transform (DCT) for MFCC
numCoeffs = 13;
mfccs = dct(logEnergies, [], 2);
mfccs = mfccs(:, 1:numCoeffs); % Keep first 13 coefficients

%% Step 10: Plot Filter Bank
figure;
plot(fbank');
title('Mel Filter Bank');
xlabel('FFT Bin');
ylabel('Gain');

%% Step 11: Plot MFCC Features
figure;
imagesc(mfccs');
axis xy;
xlabel('Frame Index');
ylabel('MFCC Coefficient Index');
title('MFCC Features');
colorbar;
```


Mel Filter Bank
Amplitude
FFT Bin Index
0
0.2
0.4
0.6
0.8
1
0
50
100
150
200
250
300


MFCC Features
MFCC Coefficient Index
Frame Index
2
4
6
8
10
12
100
200
300
400
0
-20
-40
-60
-80

EXPERIMENT NO: 7

Experiment Name: Cepstral analysis and pitch detection using cepstrum.

Objective: To estimate pitch from cepstrum.

Theory: Cepstrum is the IFFT of the log spectrum; pitch appears as a peak.

Procedure:

- Compute FFT and log
- Compute IFFT
- Locate pitch peak

Result: Cepstral plot and pitch.

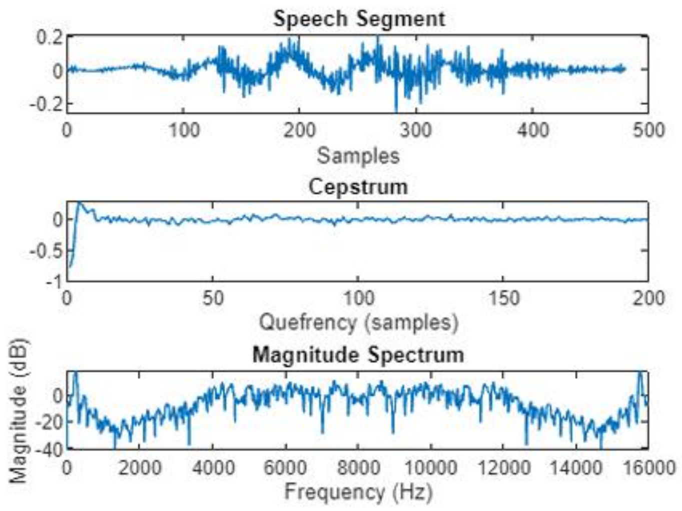
MATLAB Code

```
clc;
clear;
close all;

%% Step 1: Load Speech Signal
[audio, Fs] = audioread('guitar.wav');    % Load your speech file
audio = audio(:,1);                       % Use single channel if stereo
audio = audio - mean(audio);              % DC removal
audio = audio / max(abs(audio));          % Normalization

%% Step 2: Choose a voiced segment (manually or by frame)
% Take a short frame (e.g., 30 ms) from middle of the signal
frameLength = round(0.03 * Fs);   % 30 ms frame
startIdx = round(length(audio)/2); % Take from middle for demo
segment = audio(startIdx : startIdx + frameLength - 1);

%% Step 3: Apply window (Hamming)
segment = segment .* hamming(length(segment));

%% Step 4: Cepstrum Calculation
NFFT = 2^nextpow2(2*length(segment));
spectrum = fft(segment, NFFT);
logSpectrum = log(abs(spectrum) + eps);   % log magnitude spectrum
cepstrum = real(ifft(logSpectrum));       % real cepstrum

%% Step 5: Detect Pitch from Cepstrum
% Pitch typically lies between 50 Hz – 400 Hz
minPitch = 50;
maxPitch = 400;

minQuef = floor(Fs/maxPitch);
maxQuef = ceil(Fs/minPitch);

% Search for peak in this region
[~, qIndex] = max(cepstrum(minQuef:maxQuef));
pitchPeriod = qIndex + minQuef - 1;      % quefrency (samples)
pitchFreq = Fs / pitchPeriod;            % Fundamental frequency (Hz)

%% Step 6: Display Results
fprintf('Estimated Pitch Frequency: %.2f Hz\n', pitchFreq);

% Plot signals
```

```
figure;
subplot(3,1,1);
plot(segment); title('Speech Segment'); xlabel('Samples');

subplot(3,1,2);
plot(cepstrum);
xlim([0 200]); % show first 200 samples of cepstrum
title('Cepstrum');
xlabel('Quefrency (samples)');

subplot(3,1,3);
plot((0:length(spectrum)-1)*Fs/length(spectrum), 20*log10(abs(spectrum)));
title('Magnitude Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
```

Results

```
Estimated Pitch Frequency: 222.22 Hz
```



EXPERIMENT NO: 8

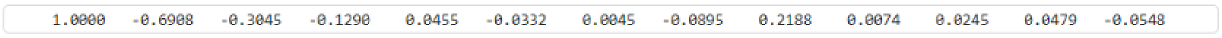
Experiment Name: Find LPC coefficients using Levinson-Durbin algorithm.
Objective: To compute LPC coefficients.
Theory: LPC models the vocal tract, Levinson-Durbin computes coefficients efficiently.
Procedure:

- Frame signal
- Autocorrelation
- Apply Levinson-Durbin

Result: LPC coefficients.

MATLAB Code

```
clc;
clear;

%% Step 1: Load Speech Signal
[audio, Fs] = audioread('speech.wav');    % Load speech file
audio = audio(:,1);                       % Use mono channel if stereo
audio = audio - mean(audio);              % DC removal
audio = audio / max(abs(audio));          % Normalization

%% Step 2: Parameters
p = 12;   % LPC order (typical values 8–16 for speech)

%% Step 3: Compute Autocorrelation
R = xcorr(audio, p, 'biased');   % Autocorrelation sequence
R = R(p+1:end);                  % Keep only lags 0...p

%% Step 4: Apply Levinson-Durbin Algorithm
[a, E, refl] = levinson(R, p);

% 'a'   -> LPC coefficients (length p+1, with a(1) = 1)
% 'E'   -> Final prediction error
% 'refl'-> Reflection (PARCOR) coefficients

%% Step 5: Display Results
disp('LPC Coefficients (using Levinson-Durbin):');
disp(a);
disp(['Final Prediction Error: ', num2str(E)]);

disp('Reflection Coefficients:');
disp(refl);
```



```
    1.0000   -0.6908   -0.3045   -0.1290    0.0455   -0.0332    0.0045   -0.0895    0.2188    0.0074    0.0245    0.0479   -0.0548
```

EXPERIMENT NO: 9

Experiment Name: Enhance noisy speech using spectral subtraction.
Objective: To reduce noise using spectral subtraction.
Theory: Estimate noise spectrum and subtract it from noisy signal.
Procedure:

- Record noisy speech
- Estimate noise
- Subtract and reconstruct

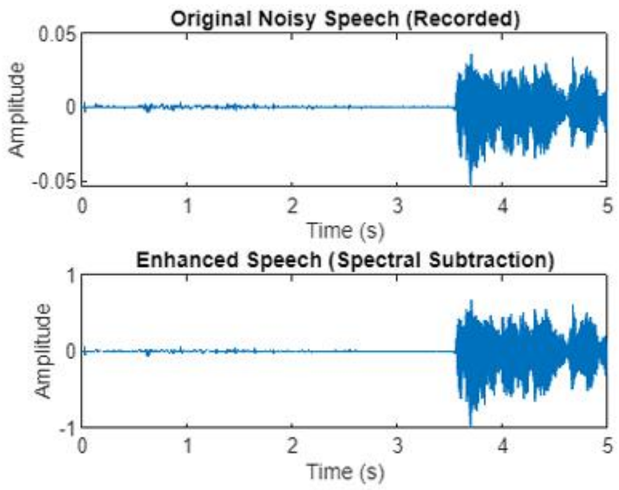
Result: Cleaned waveform and spectrogram.

MATLAB Code

```
% Spectral Subtraction for Speech Enhancement
clear; close all; clc;

% === Load Noisy Speech Signal ===
[noisy_speech, fs] = audioread('noisy_speech.wav');
noisy_speech = noisy_speech(:,1);  % Use mono if stereo

% === Parameters ===
frame_len = 256;
overlap = 128;
window = hamming(frame_len);

% === Voice Activity Detection (VAD) to estimate noise ===
% Assume the first 0.25 sec is noise
noise_duration = 0.25;
num_noise_samples = round(noise_duration * fs);
noise_signal = noisy_speech(1:num_noise_samples);

% === Estimate Noise Spectrum ===
[noise_frames, ~] = buffer(noise_signal, frame_len, overlap, 'nodelay');
noise_spectrum = zeros(frame_len, 1);

for i = 1:size(noise_frames, 2)
    frame = noise_frames(:,i) .* window;
    spec = abs(fft(frame));
    noise_spectrum = noise_spectrum + spec;
end

noise_spectrum = noise_spectrum / size(noise_frames, 2);  % Average

% === Frame the Noisy Speech ===
frames = buffer(noisy_speech, frame_len, overlap, 'nodelay');
num_frames = size(frames, 2);
enhanced_speech = zeros(length(noisy_speech), 1);
output_index = 1;
```

```
% === Process Each Frame ===
for i = 1:num_frames
    frame = frames(:, i) .* window;
    spectrum = fft(frame);

    mag = abs(spectrum);
    phase = angle(spectrum);

    % Spectral subtraction
    sub_mag = mag - noise_spectrum;
    sub_mag = max(sub_mag, 0);  % Avoid negative values

    % Reconstruct spectrum
    enhanced_spec = sub_mag .* exp(1j * phase);
    enhanced_frame = real(ifft(enhanced_spec));

    % Overlap-add
    if i == 1
        enhanced_speech(1:frame_len) = enhanced_frame;
    else
        enhanced_speech(output_index:output_index+frame_len-1) = ...
            enhanced_speech(output_index:output_index+frame_len-1) + enhanced_frame;
    end
    output_index = output_index + (frame_len - overlap);
end

% === Normalize Output ===
enhanced_speech = enhanced_speech / max(abs(enhanced_speech));

% === Save and Plot ===
audiowrite('enhanced_speech.wav', enhanced_speech, fs);

% === Plotting ===
t = (0:length(noisy_speech)-1)/fs;
figure;
subplot(2,1,1);
plot(t, noisy_speech);
title('Original Noisy Speech');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, enhanced_speech(1:length(t)));
title('Enhanced Speech using Spectral Subtraction');
xlabel('Time (s)');
ylabel('Amplitude');
```

Results


Original Noisy Speech (Recorded)
Amplitude
0.05
0
-0.05
0
1
2
3
4
5
Time (s)
Enhanced Speech (Spectral Subtraction)
Amplitude
1
0
-1
0
1
2
3
4
5
Time (s)

EXPERIMENT NO: 10

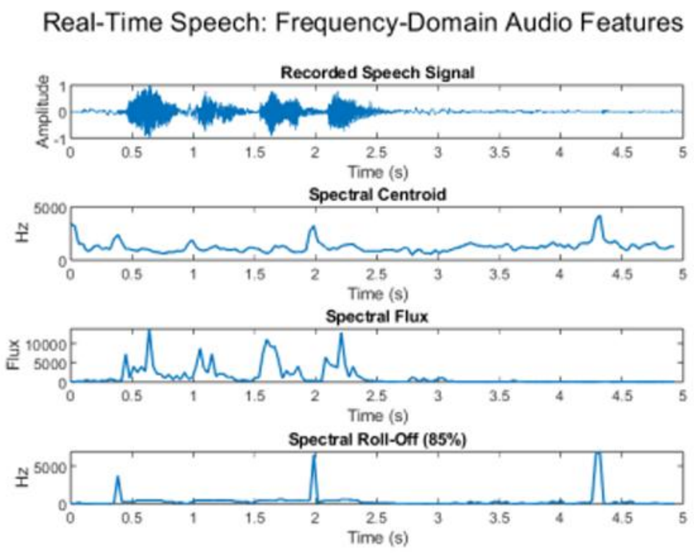
Experiment Name: Extract spectral features – SC, SF, spectral roll-off.
Objective: To compute spectral features from audio.
Theory: SC = brightness; SF = spectrum change; Roll-off = energy threshold.
Procedure:

- Frame and FFT
- Compute SC, SF, roll-off
- Plot results

Result: Graphs of SC, SF, and roll-off.

MATLAB Code

```
clc;
clear;
close all;

%% Step 1: Load Audio Signal
[audio, Fs] = audioread('audio_sample.wav');   % Replace with your file
audio = mean(audio, 2);                        % Convert to mono if stereo
audio = audio / max(abs(audio));               % Normalize

%% Step 2: Frame Settings
frameSize = 1024;
hopSize = 512;
numFrames = floor((length(audio) - frameSize) / hopSize) + 1;

%% Step 3: Predefine feature vectors
spectralCentroid = zeros(numFrames, 1);
spectralFlux = zeros(numFrames, 1);
spectralRollOff = zeros(numFrames, 1);

%% Step 4: Parameters for spectral rolloff
rolloffPercent = 0.85; % 85% energy threshold

%% Step 5: Feature Extraction Loop
previousSpectrum = zeros(frameSize/2 + 1, 1);

for i = 1:numFrames
    % Frame indexing
    startIdx = (i-1)*hopSize + 1;
    endIdx = startIdx + frameSize - 1;
    frame = audio(startIdx:endIdx);

    % Apply Hamming window
    windowedFrame = frame .* hamming(frameSize);

    % Compute magnitude spectrum
    magSpectrum = abs(fft(windowedFrame));
    magSpectrum = magSpectrum(1:frameSize/2+1);
    freqAxis = (0:frameSize/2)*(Fs/frameSize);

    % --- Spectral Centroid (SC) ---
    spectralCentroid(i) = sum(freqAxis'.*magSpectrum) / sum(magSpectrum);

    % --- Spectral Flux (SF) ---
    diffSpec = magSpectrum - previousSpectrum;
    spectralFlux(i) = sum(diffSpec.^2);
    previousSpectrum = magSpectrum;

    % --- Spectral Roll-off (SR) ---
    totalEnergy = sum(magSpectrum.^2);
    threshold = rolloffPercent * totalEnergy;
    cumulativeEnergy = cumsum(magSpectrum.^2);
    rollOffIndex = find(cumulativeEnergy >= threshold, 1);
    spectralRollOff(i) = freqAxis(rollOffIndex);
end

%% Step 6: Plot the Features
timeVec = (0:numFrames-1)*(hopSize/Fs);

figure;
subplot(4,1,1);
plot((1:length(audio))/Fs, audio);
title('Audio Signal');
xlabel('Time (s)'); ylabel('Amplitude');

subplot(4,1,2);
plot(timeVec, spectralCentroid, 'LineWidth', 1.2);
title('Spectral Centroid');
xlabel('Time (s)'); ylabel('Hz');

subplot(4,1,3);
plot(timeVec, spectralFlux, 'LineWidth', 1.2);
title('Spectral Flux');
xlabel('Time (s)'); ylabel('Flux');

subplot(4,1,4);
plot(timeVec, spectralRollOff, 'LineWidth', 1.2);
title('Spectral Roll-off (85%)');
xlabel('Time (s)'); ylabel('Hz');

sgtitle('Frequency-Domain Audio Features');
```



EXPERIMENT NO: 11

Experiment Name: Minor Project
Objective: To apply learned techniques to a project.
Theory: Implement a speech/audio application such as speaker ID, emotion detection.
Procedure:

- Choose topic
- Develop system
- Submit code/report

Result: Working demo and report.

Recommended Books:

	Book title	**Authors**	**Publishers**
No.	**Text books**		
1	Discrete Time Processing of Speech Signals	Deller J. R. Proakis J. G. and Hanson J. H.,	Wiley Interscience
2	Speech and audio signal processing	Ben Gold and Nelson Morgan	Wiley
	Reference books		
1	Digital processing of speech signals	L. R. Rabiner and S.W. Schafer	Pearson Education
2	Discrete-Time Speech Signal Processing: Principles and Practice	Thomas F. Quateri,	Pearson
3	Speech and audio processing	Dr. Shaila Apte	Wiley India Publication

Pedagogy:

1. Experiments using Matlab and SCILAB
2. Use of Open Source Software such as PRAAT and Audacity