BATUHAN TOSUN **2017401141**

EE473 FINAL PROJECT: LPC VOCODER

1) Linear Prediction Model for Speech Production

Linear prediction is one of the most widely used methods in speech related tasks such as speech coding, speech storage, speech production etc. In this project, linear prediction model will be used for speech production task. The main idea behind the linear prediction model is that the current speech sample can be very precisely approximated as the linear combination of its past samples. It can be simply expressed as the equation given below [1]:

$$\boldsymbol{s'(n) = \sum_{i=1}^{p} \alpha_i * s(n-i)}$$

where s'(n) is the estimated speech signal, α_i's are the predictor coefficients (in total there are p coefficients). Therefore, the prediction error signal (or residual signal) e(n) is the difference between the actual speech samples s(n) and the estimated ones s'(n). It can be expressed as [1]:

$$\boldsymbol{e(n) = s(n) - s'(n) = s(n) - \sum_{i=1}^{p} \alpha_i * s(n-i)}$$

There are two common approaches for calculating the predictor coefficients α_i's (LP coefficients). The first method is called "Autocorrelation Method" and the second method is called "Covariance Method". Each method aims minimizing the sum of squared samples of the prediction error e(n) (or in other saying minimizing the energy of the prediction error signal) to calculate the coefficients. Due to its computational efficiency and stability, I preferred the "Autocorrelation Method" in this project. Before going into the details of the autocorrelation method, I should mention that the speech signal is divided into small frames with the technique called overlap-add (OLA). For overlap-add, our speech signal is multiplied by a window, and we get a windowed speech segment which can be denoted by $s_w(n)$. In this project, Hanning window is used. After dividing our speech signal into small frames, we try to find the predictor coefficients that minimize the energy of the prediction error signal for each frame. This brings us to p linear coefficients with p unknown variables. These linear equations can be expressed in the matrix form as given below:

$$\boldsymbol{R_s a = r_s}$$

where $\mathbf{R_s}$ is the short-term autocorrelation matrix, which is also a Toeplitz matrix, **a** is the predictor coefficient vector and $\boldsymbol{r_s}$ is autocorrelation vector. Therefore, the short-time predictor coefficients α_i's of each speech frame can be found by [1]:

$$\boldsymbol{a = R_s^{-1} r_s}$$

After we obtain the predictor coefficients (LP coefficients) for each frame, the next step is to estimate the pitch period/frequency of each speech frame. The details for pitch estimation will be given in the later sections. After accomplishing this task, the next step is to classify each frame as "voiced" or "unvoiced". Again, detailed information for voiced/unvoiced classification will be given in the later sections. There is one more step before the decoding section and it is gain calculation. The gain required for each frame is calculated according to the formula given in the book [2]. After obtaining every necessary variable, the "encoding" part is completed. In the "decoding" part, first, for each speech frame, we generate either an impulse train with period equal to the pitch period of the corresponding frame or a gaussian random noise depending on the voiced/unvoiced label of that frame. The impulse train is generated for "voiced" speech frames and the white gaussian noise is generated for "unvoiced" speech frames. After this, the generated signal (either impulse train or noise) is multiplied with the gain value of the corresponding frame. Then, using previously found predictor (LP) coefficients α_i's , we create our LPC synthesis filter and the resulting signal is passed through this filter. At the end, we get our LPC decoded speech signal.

2) <u>What is Pitch?</u>

Before explaining what pitch is and how it can be calculated we should give a brief information about human speech to understand the concepts better. Human speech can be classified into two different categories which are voiced and unvoiced speech. The main physical factor for a speech (a segment of speech) to be classified as voiced or unvoiced is that the vibration of vocal cords. If the vocal cords vibrate during the creation process of the sound, the resulting sound (speech) is classified as "voiced", if not it is classified as "unvoiced". Those vibrations in voiced sounds can be associated with a certain frequency value. Therefore, "pitch" can be described as the fundamental frequency of vocal cords' vibrations [3].

3) <u>Pitch: How It Can Be Calculated</u>

There are numerous pitch detection algorithms, and those algorithms can be divided into mainly two depending on the domain in which they are performed (time or frequency). In my project, I have implemented only the ones formulated in the time-domain (in total two algorithms); therefore, in this section the frequency domain approaches will not be included. The pitch period (or frequency) can be calculated using the periodicity property of speech segments in time. My first algorithm to find "pitch period" was using "autocorrelation method". In the previous section, I explained the term "pitch" which is basically the fundamental frequency of the vocal cord vibrations. If there is some "fundamental frequency" that we can talk about, then there must be a corresponding "fundamental period" as well. If the signal in each speech frame has a periodic behavior, its autocorrelation function is also periodic. Moreover, the autocorrelation function has local maximum points (peaks) at values which are the integer multiples of the period ($\tau=n*P$, where $n=\ldots-2,-1,0,1,2,\ldots$) of the speech frame. Therefore, the pitch period of the speech segments can be estimated by taking the autocorrelation and detecting the position where the autocorrelation function has the largest peak. This method is easy to implement but it might be also misleading when one only uses the position information related to the largest peak of the resulting autocorrelation function. The reason for this is that the autocorrelation function has numerous peaks, and these peaks occur not only due to the periodicity of the vocal vibrations but also due to vocal tract response. If we only consider the largest peak for pitch period estimation and that peak is due to vocal tract response but not to periodicity, then this approach will lead us to a wrong result. To overcome this a technique called "center-clipping" is used with autocorrelation method [3]. This is my second algorithm related to pitch estimation problem. This technique is designed to suppress the undesired effects of the vocal tract response and it is basically an operation in which distracting features of the speech signal is eliminated to make periodicity to prominent. In this technique, we determine a clipping threshold (C_L)

value (which is set to the 30% of the maximum absolute value of the speech signal in each frame in my program). The clipping operation is done on the speech signal s(n), and center-clipped signal $s_{clc}(n)$ is obtained according to the expression which is given below [3]:

$$\boldsymbol{s_{clc}(n)} = \boldsymbol{clc}[\boldsymbol{s(n)}] = \begin{cases} \boldsymbol{s(n)} - \boldsymbol{C_L}, & \boldsymbol{s(n)} \geq \boldsymbol{C_L} \\ \boldsymbol{0}, & |\boldsymbol{s(n)}| < \boldsymbol{C_L} \\ \boldsymbol{s(n)} + \boldsymbol{C_L}, & \boldsymbol{s(n)} \leq \boldsymbol{C_L} \end{cases}$$

After obtaining our center-clipped signal, we take the autocorrelation of this signal, and the remaining steps are the same as the previous method. At the end, we get a pitch period/frequency estimate for each frame.

4) <u>Voiced/Unvoiced Decision</u>

One of the most important problems of the project is deciding whether the speech frame is voiced or not. The reason behind its importance is that the precision of this decision will significantly affect the performance of our LPC Vocoder. Voiced-unvoiced speech classification is implemented for each speech frame (10ms-30ms short period of speech segments) of the speech signal. There are several approaches for making this classification. I have implemented the 3 most common approaches which are performed in time domain. In the previous sections, I have mentioned the fact that if the vocal cords vibrate during the creation process of the sound, this sound is classified as “voiced”. Also, I described the term “pitch frequency” (or pitch period) which is basically the fundamental frequency (or period) of vocal cords’ vibrations, during the creation of “voiced” speech. Therefore, we can understand that the input excitation of a voiced speech is periodic (just like impulse train) whereas the input excitation of the unvoiced speech has the characteristic of almost a random noise [4]. This information is very important and will be used in the decoder part after the classification of each speech frame is done but I will come to the details of this topic later. Before explaining the methods, I should mention the fact that voiced speech frame has high energy, less number of zero-crossings and low pitch frequency compared to unvoiced speech frame [4]. The first method for this classification task is based on the pitch frequency of each speech frame. In this method, I set a threshold value for pitch frequency (which is 200Hz in the program), and the pitch frequency of each speech frame (which were calculated at the previous step) is compared with this threshold value. The ones which have a lower pitch frequency is classified as “voiced” and the others are classified as “unvoiced”. The second method for this classification task is based on the energy of each speech frame. In this approach, I set a threshold value for the energy, and the energy of each speech frame is compared with this threshold value. The ones which have higher energy is classified as “voiced” and the others are classified as “unvoiced”. The last method is based on the zero-crossing rate of each speech frame. In this approach, I set a threshold value for the zero-crossing rate (ZCR), and the ZCR of each speech frame is compared with this threshold value. The ones which have lower ZCR is classified as “voiced” and the others are classified as “unvoiced”.

5) Block Diagram of LPC Encoder and Decoder

The block diagram of LPC Encoder and Decoder is given below.

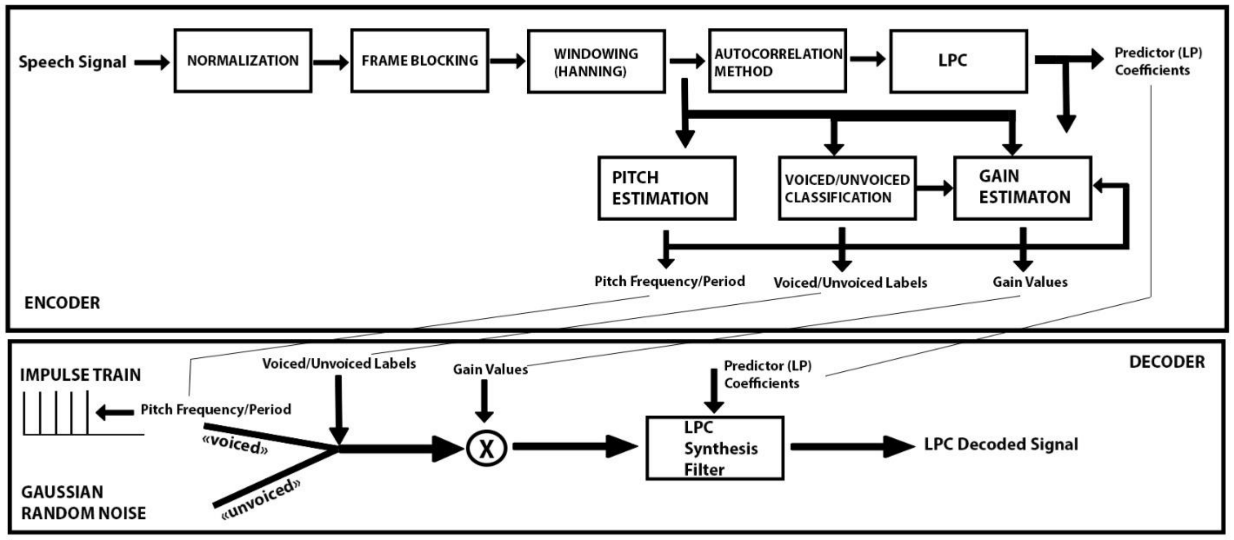

Fig1: Block Diagram of LPC Encoder and Decoder

6) The Details of the Algorithm

At the beginning of my program, the speech input is read from the file first. Then, overlap-add (OLA) method is implemented with "Hanning" window for dividing the speech signal into small frames of 30ms (with a frame/window shift of 15ms). The resulting frames are stored in the rows of "X_m" matrix. The next step is to find the short-time predictor coefficients of each frame using "Autocorrelation Method". Here, the values of autocorrelation function of each frame are used in the creation of the "Autocorrelation (Toeplitz) Matrix" and they are also stored in the rows of "R_m" matrix for later stages. Using the resulting Toeplitz matrix, the short-time predictor coefficients α_i's for each frame are stored in the columns of matrix "A".

The next step is to calculate the pitch period/frequency of each frame. To achieve this task, I have created a function called "pitch_estimation_func". Using this function, the user can obtain the pitch periods/frequencies of every speech frame which are calculated via one of the methods, previously mentioned in the section "Pitch: How It Can Be Calculated". The method can be chosen by setting "method" string input to either "autocorrelation" or "center-clipping". Also, the user must adjust the threshold value (which is used for center-clipping method) to get good estimations. After giving the proper inputs to the function, it returns pitch period and frequency value of each frame, which are stored in "pitch_periods" and "pitch_frequencies" arrays. Period estimation is done by first taking the autocorrelation function of the current speech frame using MATLAB's "xcorr" function. Since the result is symmetric only the right half of the resulting autocorrelation vector is used. Then, using MATLAB's "findpeaks" function, the locations of the local maximum points (peaks) are found and the mean of the distances between each two closest peak is calculated. To find the pitch period of the corresponding frame the result is divided by the sampling rate (fs). When "center-clipping" method is used, the only difference is that the center-clipped version of the speech frame is created first, the rest is the same.

Next step is to classify each frame as voiced or unvoiced. To achieve this task, I have created a function called "voiced_unvoiced_detector". This function implements one of the methods, previously mentioned in the section "Voiced/Unvoiced Decision". The method can be chosen by setting "method" string input to "zero-crossing", "pitch_frequency" or "energy". The user must also adjust the threshold value properly for the selected method. After giving the proper inputs to the function, it returns "voiced_unvoiced" array in which the voiced/unvoiced labeling of each frame is stored. In the "zero-crossing" method, the number of zero-crossings in each frame is calculated by counting the number of sign changes and then the resulting number is compared with the given threshold value. If the number of zero-crossings of that frame is smaller than the threshold the frame is classified as "voiced" (label 1), otherwise "unvoiced" (label 0). In the "pitch_frequency" method, the previously found pitch frequency value of each frame is compared with the given threshold. If it is smaller than the threshold, the frame is classified as "voiced", otherwise "unvoiced". In the "energy" method, the square of the elements of the speech frame is taken and then elements of the resulting array are summed up. If the result is bigger than the given threshold the speech frame is classified as "voiced", otherwise "unvoiced".

After these two important steps, it is time for the gain value calculation for each frame. To achieve this task, I have created a function called "gain_calculator". This function calculates the gain of each frame using a different expression with respect to its voiced/unvoiced labeling. The expressions are taken from the book and since there is no algorithmic complexity in this part I will move onto the next step: "Decoding".

Now, since we have calculated all the required variables to generate our speech, it is time for implementing the "decoder" of our LPC Vocoder. We generate our speech again frame-by-frame basis at the first. So, for each frame, we first look whether it is classified as "voiced" or "unvoiced". If the label of the current speech frame is "voiced", then an impulse train with the period of the pitch period of the corresponding frame is generated. If the label of the speech frame is "unvoiced", then a gaussian random noise is generated. Then the resulting signal (impulse train or noise) is multiplied with its corresponding gain value and then it is passed through the filter whose coefficients are the predictor (LP) coefficients of the corresponding frame. The resulting signals (for every frame this operation is done) of the filter are then stored in rows of "Decoded_speech" matrix. By reverse ordering the OLA operation, we concatenate those frames (rows) and generate "dec_speech" array, which holds the samples of our decoded speech signal. Normalization is done on this signal as well and finally we obtain the LPC decoded speech signal.

7) <u>Input Waveform and Output Waveform</u>

The number of predictor coefficients, which is denoted by p, is set to 12. For pitch estimation "Center-Clipped Autocorrelation" method is used with clipping threshold 0.3. For voiced/unvoiced classification "Pitch Frequency" approach is used with frequency threshold value of 200. The plots of the original speech signal and the resulting LPC decoded signal can be observed in the figure given below.

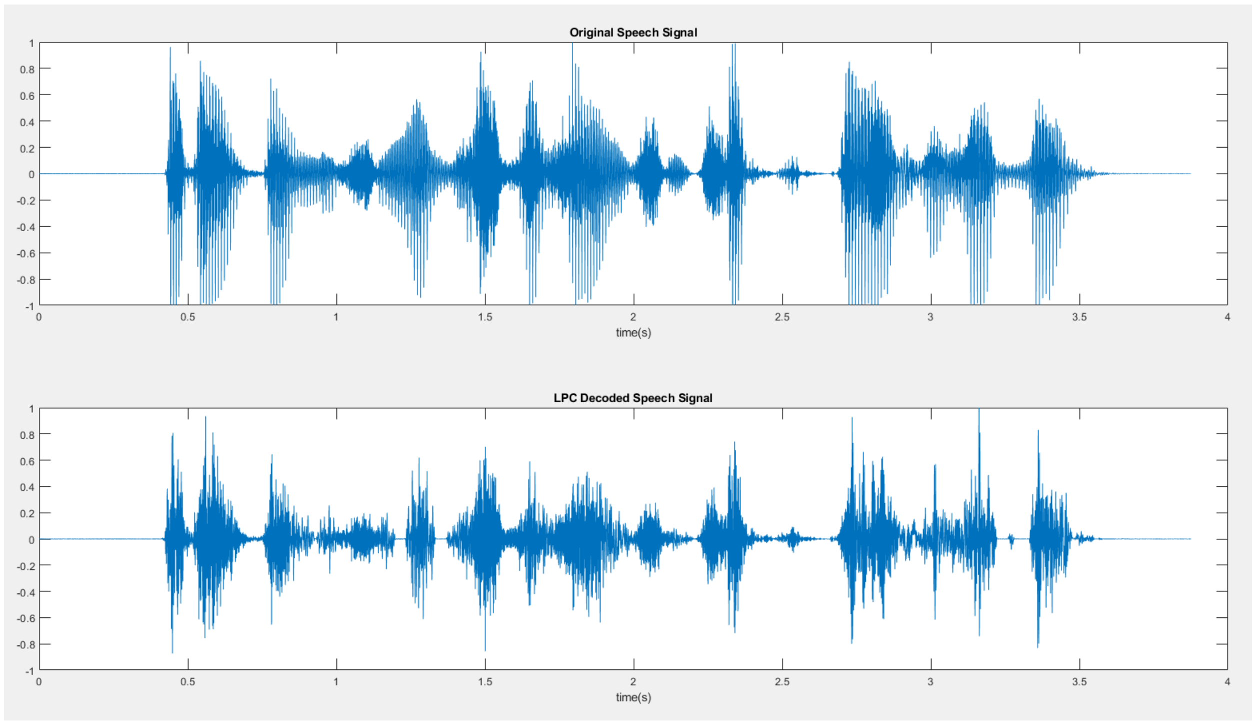

Fig2: Plots for Original Speech Signal and LPC Decoded Speech Signal

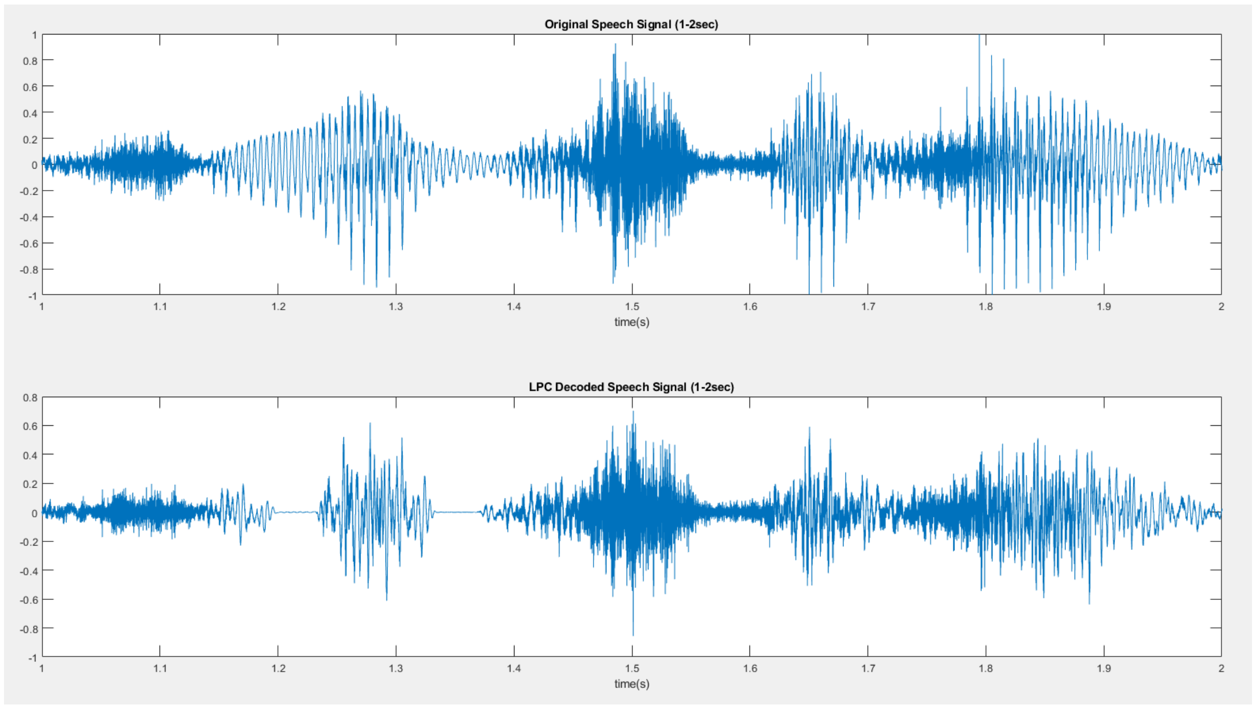

Fig3: Plots for Original Speech Signal and LPC Decoded Speech Signal in the time interval [1 sec,2 sec]

REFERENCES

[1] Harrington J., Cassidy S. (1999) Linear Prediction of Speech. In: Techniques in Speech Acoustics. Text, Speech and Language Technology, vol 8. Springer, Dordrecht.

[2] Deller, Proakis, and Hansen, Discrete-Time Processing of Speech Signals

[3]_S. S. Upadhya, "Pitch detection in time and frequency domain," *2012 International Conference on Communication, Information & Computing Technology (ICCICT)*, 2012, pp. 1-5,

[4] S.nandhini and A.shenbagavalli. Article: Voiced/Unvoiced Detection using Short Term Processing. IJCA Proceedings on International Conference on Innovations in Information, Embedded and Communication Systems ICIIECS (2):39-43, November 2014

APPENDIX : The MATLAB Code

1) pitch_estimation_func.m

```
function [pitch_periods,pitch_frequencies] = pitch_estimation_func(X_m,fs, method,
percentage_threshold)

% this function calculates the pitch period/frequency of each speech frame
% (chunk) using one of the two methods "autocorrelation" or "center clipping"
n_overlap = size(X_m,1);

pitch_periods = zeros(1,n_overlap);
pitch_frequencies = zeros(1,n_overlap);

for i = 1:n_overlap % for each speech frame

    if method == "autocorrelation"

        % we diractly take the autocorrelation function of the current
        % speech frame

        [R_xx, ~] = xcorr(X_m(i,:), X_m(i,:));
        % only the one side is enough
        R_xx_one_side = R_xx((length(R_xx)-1)/2+1:end);
        % then we find locations of the peaks (local maximum)
        [pks_max, locs_max] = findpeaks(R_xx_one_side);
        %the mean of the differences between local maximums is divided by
        %the sampling frequency and the result is our pitch period
        period_in_samples = mean(diff(locs_max));
        pitch_periods(1,i) = period_in_samples/fs;
        pitch_frequencies(1,i) = fs/period_in_samples;

    elseif method == "center_clipping"

        % first clipped version of the current frame is generated according
        % to the formula
        y_m = zeros(1,size(X_m,2));
        max_abs_val = max(abs(X_m(i,:)));
        clipping_threshold = percentage_threshold*max_abs_val;
        y_m = y_m+(X_m(i,:)-clipping_threshold).*(X_m(i,:)>=clipping_threshold);
        y_m = y_m + (X_m(i,:)+clipping_threshold).*(X_m(i,:)<=-clipping_threshold);

        % then the autocorrelation function of the center-clipped speech
        % frame is calculated

        [R_yy, ~] = xcorr(y_m, y_m);
        R_yy_one_side = R_yy((length(R_yy)-1)/2+1:end);
        %only the one side is enough
        % then we find locations of the peaks (local maximum)
        [pks_max, locs_max] = findpeaks(R_yy_one_side);
        %the mean of the differences between local maximums is divided by
        %the sampling frequency and the result is our pitch period
        period_in_samples = locs_max(1);
        %period_in_samples = mean(diff(locs_max));
        pitch_periods(1,i) = period_in_samples/fs;
        pitch_frequencies(1,i) = fs/period_in_samples;

    end

end
```

```
end
```

2) voiced_unvoiced_detector.m

```
function [voiced_unvoiced] = voiced_unvoiced_detector(X_m, method,
threshold_val, pitch_frequencies)

% this function classifies each speech frame as voiced or unvoiced
% (chunk) using one of the three methods: "zero-crossing",
% "pitch_frequency", "energy"

n_overlap =  size(X_m,1);
voiced_unvoiced = zeros(1,n_overlap);

for i = 1:n_overlap

    if method == "zero-crossing"

        signed_frame = sign(X_m(i,:));
        num_zero_cross = sum(abs(diff(signed_frame)));
        % if the total number of zero-crossing of the current speech frame is
smaller than the
        % threshold value it is classified as "voiced", otherwise "unvoiced"
        voiced_unvoiced(1,i)= num_zero_cross<threshold_val;

    elseif method == "pitch_frequency"

        % if the estimated pitch frequency of the current speech frame is
lower than the
        % threshold value it is classified as "voiced", otherwise "unvoiced"
        voiced_unvoiced(1,i) = pitch_frequencies(1,i)<threshold_val;

    elseif method == "energy"
        % if the energy of the current speech frame is higher than the
        % threshold value it is classified as "voiced", otherwise "unvoiced"
        energy_frame = sum(abs(X_m(i,:)).^2);
        voiced_unvoiced(1,i)= energy_frame>threshold_val;
    end

end

end
```

3) gain_calculator.m

```
function [gain] = gain_calculator(R_m, A_m, pitch_periods,voiced_unvoiced)

% this function calculates the required filter gain for each speech frame
% according to the gain formula given in the lecture book
n_overlap =  size(R_m,1);
gain = zeros(1,n_overlap);

for i = 1:n_overlap % for each frame
```

```
    inner = R_m(i,1) - sum(A_m(:,i)'.*R_m(i,2:end));

    if voiced_unvoiced(1,i) == 1 % the formula changes wrt speech frame being 
voiced/unvoiced
        gain(1,i) = sqrt(pitch_periods(1,i)*inner);
    else
        gain(1,i) = sqrt(inner);
    end

end

end
```

4) lpc_vocoder_main.m

```
%% EE473 Final Project : LPC Vocoder
% Batuhan Tosun 2017401141

%% Overlap & Add Method
% speech signal is divided into 30ms frames

[speech, fs] = audioread("input.wav");

speech = speech(:,1)';

% normalize speech
speech = speech/(max(abs(speech)));

% total number of samples for 30ms frame
N = floor(30*fs/1000);

% total number of samples per window shift 15ms
R = floor(0.5*N);

% total number of frames created is equal to n_overlap
n_overlap = floor((length(speech)-N)/R)+1;

% speech frames will be stored in X_m
X_m = zeros(n_overlap, N);

% Hanning window is used
w = hann(N, 'periodic');

for i=1:n_overlap

    X_m(i,:) = w'.*speech((1:N)+(i-1)*R);

end

%% Autocorrelation Method
```

```
% the filter coefficients for each frame are calculated using the
autocorrelation method

% total number of predictor coefficients (LP coefficients)
p = 12;

% matrix for holding the values of autocorrelation function of each frame
R_m = zeros(n_overlap,p+1);

% matrix for holding the short-time predictor coefficients for each frame
A = zeros(p,n_overlap);

% matrix for holding the error signal at the end of each frame
E = zeros(n_overlap,N);

for k = 1:n_overlap % for each frame

    s_frame = X_m(k,:);

    for j=1:p+1
        R_m(k,j) = s_frame(1:N-j+1)*s_frame(1+j-1:N)';
    end

    r1 = R_m(k,2:end);
    r2 = R_m(k,1:end-1);
    R_toep = toeplitz(r2);

    % alfa_k's are found
    A(:,k)=R_toep\r1';

    subConv = conv([0 X_m(k,:)], A(:,k)');
    E(k,:) = X_m(k,:)-subConv(1:N);

end


%% finding pitch periods (Center-clipping)

method1 = "center_clipping";
percentage_threshold=0.3;
[pitch_periods,pitch_frequencies] = pitch_estimation_func(X_m, fs, method1,
percentage_threshold);

%% finding voiced & unvoiced (pitch-freq)

method = "pitch_frequency";
threshold_val = 200;
voiced_unvoiced = voiced_unvoiced_detector(X_m,method,threshold_val,
pitch_frequencies);

%% gain calculation
gain = gain_calculator(R_m, A, pitch_periods,voiced_unvoiced);
```

```
%% Decoding section

Decoded_speech = zeros(n_overlap,N);

pulse_train = zeros(1,N);
offset = 0;

for i=1:n_overlap % for each speech frame

     % if the current speech frame is voiced we generate impulse train with a
period equal to corresponding pitch period
    if(voiced_unvoiced(1,i)==1)

        % creating impulse train
        src1 = zeros(N,1);
        step = round(pitch_periods(1,i)*fs);
        pts = (offset+1):step:N;

        if ~isempty(pts)
            offset = step + pts(end) - N;
            src1(pts) = 1; % impulse train, compensate power
        end

    % the resulting impulse train is multiplied by the corresponding gain
    % factor and then filtered out using the corresponding short-time
    % predictor coefficients

        src1 = src1';
        pitch_periods(1,i)
        sum(src1)
        x_hat = filter(1,[1; -A(:,i)], src1*gain(1,i));

    % if the current speech frame is unvoiced we generate gaussian random
    % noise
    else
        src2 = randn(1,N);
        % the resulting gaussian is multiplied by the corresponding gain
        % factor and then filtered out using the corresponding short-time
        % predictor coefficients
        x_hat = filter(1,[1; -A(:,i)],src2*gain(1,i));
    end
    % each decoded frame is stored in the rows of "Decoded_speech" matrix
    Decoded_speech(i,:)=x_hat;
end

%% decoding our speech

% here the decoded speech frames are concatenated in the reverse order of OLA
and the decoded speech signal is created
dec_speech = zeros(1,length(speech));
w = hann(N, 'periodic');

for j=1:n_overlap
```

```
    dec_speech = dec_speech + [zeros(1,(j-1)*R) w'.*Decoded_speech(j,:)
zeros(1,length(speech)-(j-1)*R-N)];

end

% normalization
dec_speech = dec_speech/(max(abs(dec_speech)));

%% plotting

figure
subplot(2,1,1)
plot([0:length(speech)-1]/fs,speech)
title("Original Speech Signal")
xlabel("time(s)")
subplot(2,1,2)
plot([0:length(speech)-1]/fs,dec_speech)
title("LPC Decoded Speech Signal")
xlabel("time(s)")

figure
subplot(2,1,1)
plot([0:length(speech)-1]/fs,speech)
title("Original Speech Signal (1-2sec)")
xlabel("time(s)")
xlim([1,2])
subplot(2,1,2)
plot([0:length(speech)-1]/fs,dec_speech)
title("LPC Decoded Speech Signal (1-2sec)")
xlabel("time(s)")
xlim([1,2])

%%
disp("Original Sound Playing...")
sound(speech,fs)

pause(round(length(speech)/fs)+1);

disp("LPC-decoded Sound Playing...")
sound(dec_speech,fs)
```