Siham Elmali **Johns Hopkins University**

November 17th, 2016

520.445/645

Audio Signal Processing

Fall 2016

Project 2: LPC Vocoder

Abstract

For this project, we were tasked with designing a speech codec using the LPC compression scheme. We were given 10 speech samples and the option of using provided estimates of voiced/unvoiced decision and pitch frequencies at 100 frames per second. Completing the speech codec required implementing both an LPC encoder and a speech synthesizer to recreate the sound using as little data as possible.

Overview of Speech Generation

Speech is achieved by compression of the lung volume, causing air flow that may be made audible if set into vibration by the activity of the larynx. This sound can then be made into speech by various modifications of the subpharyngeal vocal tract.

- Lungs provide the energy source - **Respiration**
- Vocal folds convert the energy into audible sound - **Phonation**
- Articulators transform the sound into intelligible speech - **Articulation**

Methods used in the project

Linear predictive coding (LPC) model

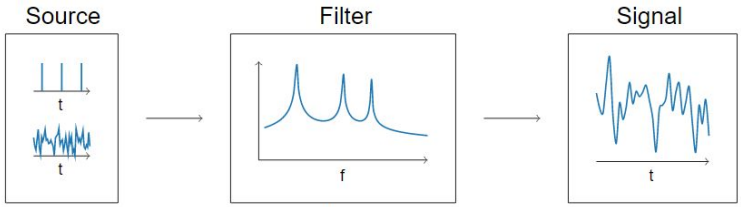


Figure 1: LPC model

Linear predictive coding (LPC) is a method for signal source modelling in speech signal processing. It is often used by linguists as a formant extraction tool. It has wide application in other areas. LPC analysis is usually most appropriate for modeling vowels which are periodic, except nasalized vowels. The LPC is based on the source-filter model of speech signal. There are two possible signals for the source: an impulse train or random white noise. These signals model pitched sounds and plosive/fricatives respectively. The common characteristic for both impulse train and white noise is that they are spectrally flat; all spectral information is modeled in the filter.

Modified Autocorrelation

The modified autocorrelation pitch detector MACF differs from the common autocorrelation method by using center-clipping technique in a pre-processing stage. The relation between the input signal x(n), and the center-clipped signal y(n) is:

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

where C_L is the clipping threshold. Generally, C_L is about 50% of the maximum absolute signal value within the signal frame. For better results, the clipping threshold was set to 55%. Non-linear operations on the speech signal such as center-clipping tend to flatten the spectrum of the signal passed to the candidate generator. This results in the increase of the distinctiveness of the true period peaks in the autocorrelation function. The figure below presents the example of voiced frame, its center clipped version, and the difference between the autocorrelation function calculated from original signal frame and center-clipped signal frame.

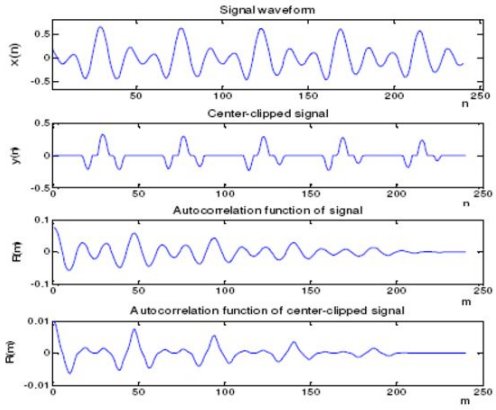


Figure 2: Comparison of autocorrelation function
calculated from voiced frame and its center-clipped version

The process of applying the modified autocorrelation algorithm begins by segmenting the signal into overlapping frames. Speech recordings with a 16kHz sampling frequency were used for this project, resulting in approximately 320 samples in a 20 millisecond frame. Frames were overlapped with as few as 40 samples or as many as 160. This method requires low-pass filtering (LPF) with a cutoff of around 800 Hz. The low pass filter designed for this implementation is a butterworth IIR with a corner frequency of 750 Hz and 80 dB attenuation by 1 kHz. Then, the autocorrelation function for the frame is computed over the range of lags above some offset (default 20; can be changed in the main script). Additionally, the autocorrelation function at 0 delay is computed for appropriate normalization purposes. The normalized autocorrelation function is then searched for its maximum peak. The value of the peak and its position are defined. If the value of peak exceeds the configured threshold (0.30 is a reasonable value), the frame is classified as voiced. Otherwise, the section is classified as unvoiced. The value of fundamental frequency can be computed from the pitch period given by the peak position.

Code overview

Part 1

Main script

The main script loads the audio file to get the audio data. The main script then passes the speech data and parameters to the encoder function, and receives vectors of coefficients, pitches, and gains for each frame in return. It then passes these vectors directly onto the decoding function, which returns the re-synthesized speech. Note that various parameters (such as thresholds and the filter order) can be set by modifying the globals defined at the stop of **lpc_main.m**.

All the parameters used were declared as global variables, and can be changed in the script. The right choice of these parameters can improve the quality of the resynthesized signal drastically. The main script displays plots comparing the original signal and resynthesized signal as well as the resynthesized signal and the all poles filter for analysis. The script also audio writes all of received signals in the format specified in the project prompt.

Encoder function

The encoder function in **lpc_encoder.m** encodes the provided speech segment using LPC. This function decomposes the input signal into overlapping frames and generates parameters required for

re-synthesis, including LPC coefficients, gain, voicing, and pitch. This data is returned as a single struct. Voicing and pitch are determined using the previously-mentioned modified autocorrelation algorithm.

The process begins by computing the frame size in samples for each audio segment. We then zero padded each speech segment for even frame division and overlap space. A pre-emphasis high pass filter was used to help with the LPC parameters extraction. The main goal of the pre-emphasis filter is to boost the higher frequencies in order to flatten the spectrum. This improvement leads to a better result for the calculation of the coefficients using LPC. The main section of the encoder function is application of the modified autocorrelation method, LPC method then storing all required parameters in a structure.

The process begins as follow:

- We start by iterating over all of the frames
- We get the overlapped frames (the size of the overlap is set as a global variable)
- We multiply the overlapped frames by Hanning window of the overlapped size
- We then apply a low pass filter with passband of 750 Hz
- We get the frame energy
- We apply a noise gate threshold that can changed from the main script
- We then apply a center clipping with a threshold of 60% (can be modified in main script)
- We apply the autocorrelation method using MATLAB autocorr function
- We then compute the peak and the period from output of the autocorrelation
- We threshold the fundamental amplitude to determine if voiced
    - If the fundamental is too weak to consider voiced, we store a zero
    - We then look at the fundamental frequency if it is too high to be realistic, we store a zero as well
    - If the fundamental frequency is reasonable, we go ahead by storing the pitch.
    - If the frame is silent we store a zero for the pitch
    - The voicing decisions for each frame were as follow: voicing decisions for each frame with 0 indicating silence, +1 indicating voiced frame, and -1 indicating unvoiced frame.
- The final step is Applying the LPC function to get the LPC coefficients, and gains and format the data in a single struct to return.

The gain for the frame is extracted as the square root of the power provided by the LPC function.

Decoder function

The decoder function decodes and re-synthesizes speech using the provided data. It generates either glottal pulses at the appropriate frequency for voiced frames or noise for unvoiced frames, and then filters each frame according to the provided LPC coefficients and gain.

If the frame is voiced, a train of impulses spaced by the given pitch period is convolved with a glottal pulse wave generated using the **Rosenberg Pulse model** via *lpc_rosenberg*.m (the function publicly available over the internet). This results in an appropriately-spaced train of glottal pulses. I had to ensure that the pulses are appropriately spaced around the transition between two adjacent voiced frames. If the previous frame was voiced, the code looks for the end of the previous pulse and introduces an appropriate gap before beginning the next frame rather than simply placing the first pulse starting at the first sample in the frame. If the frame is silent, the code returns the frames as zeros. If the frame is unvoiced, a noise is generated using white Gaussian noise (provided by the function *wgn*) with a configurable noise power. The result is then filtered using the LPC coefficients and scaled by the gains. The order of the filter for voiced frames is equal to the number of LPC coefficients. The normalized audio vector is returned to the caller as a cell array that contains all of the received speech segment.

Performance

To assess the number of bits per seconds that is required to reconstruct the signal, we need to determine how much data is required to transmit a single frame, and how many frames occur in a second. Assuming our frame size is 20ms, we're transmitting 50 frames per second. The worst-case scenario is having to transmit all of our LPC coefficients and other data (i.e. no silent frames or unvoiced frames, which allow for sending reduced information). Given a filter order of 16, we have 17 coefficients. We'll will need 17x4 = **68 bytes,** assuming that a single precision floating point format does not introduce too much quantization noise. The gain could possibly be quantized into a single byte (providing integer levels 0 through 255), but a more conservative estimate of two bytes for an unsigned short (uint16) might be more appropriate. Voicing and pitch data can be combined into a single 16-bit signed integer, by assuming that a positive value indicates pitch, zero indicates silence, and negative one indicates an unvoiced frame.

Altogether, we get the following:

17*4 + 2 + 2 = 72 bytes per frame * 50 frames per seconds = **28.8 kbps**. However, on average we would expect a considerably lower number, since we can send reduced data for unvoiced/silent frames and other data-packing techniques could be applied. A reasonable number might be 20 kbps.

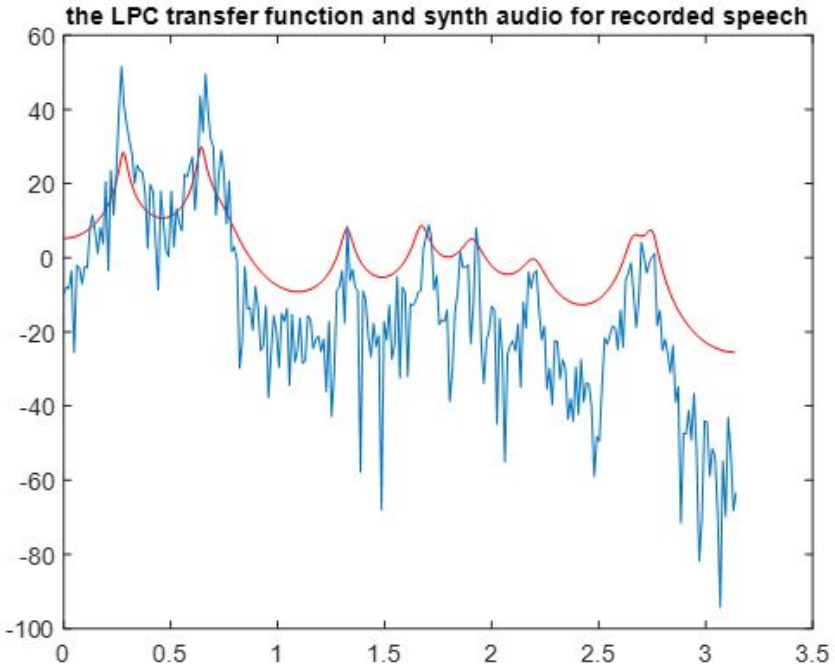
Part 2

In part two, we were tasked to record our own voice saying the sentence "The synthesized signal is supposed to be of high standard" and send the audio file through the communication channel designed in part 1. For this part, having all of the parameters easily configured from the main script was very useful. I created a section for this audio file specifically in my main script. After spending some time tweaking the parameters, the synthesized audio quality has increased dramatically. The parameters used to achieve such result are as follows:

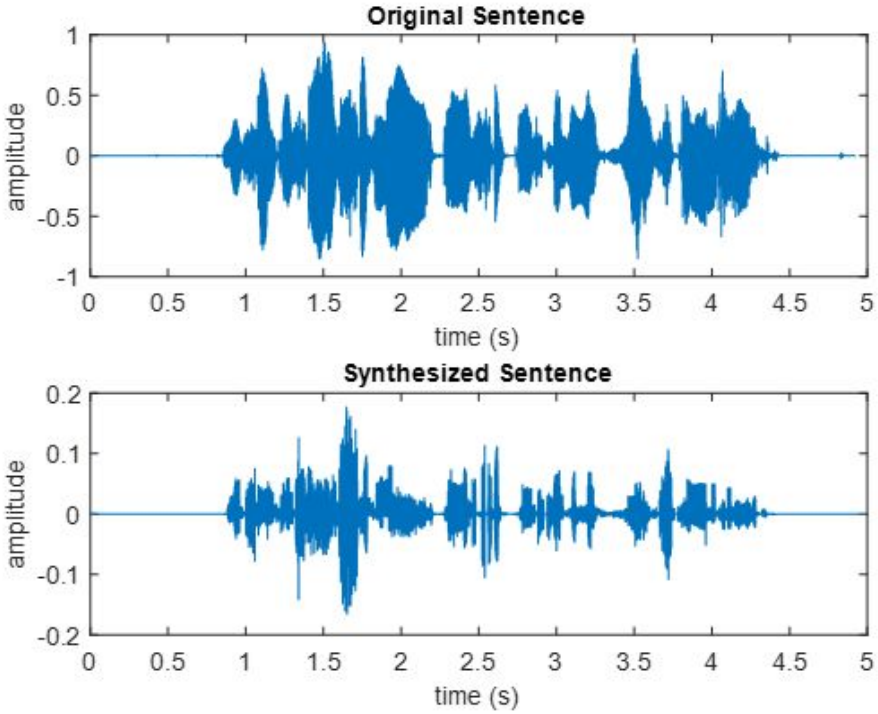
```
ACORR_OFFSET = 15;        % Lag offset for autocorrelation processing
CLIP_RATIO = .60;         % Center clipping ratio (of max amplitude)
FRAME_TIME = .015;        % Frame duration
NOISE_POWER = -12;        % Noise power for unvoiced speech base
ORDER_VOICED = 20;        % Filter for voiced frames
NOISE_GATE = .00001;      % Noise gate for silence thresholding
VOICE_THRESH = 0.35;      % Threshold for voiced frame detection
MAX_PITCH = 450;          % Maximum accepted fundamental frequency
OVERLAP = 40;             % Number overlapped samples
```

I set my maximum accepted pitch to 450 just because females tend to have a higher pitch than males. Both pitch and voicing detection using the autocorrelation were used in the part similar to part 1

The following plot represents the LPC transfer function and synthesized audio for the sentence provided over one frame. We can see how a 20th-order all-pole filter follows the envelope of the speech frame. Formants are clearly discernable.



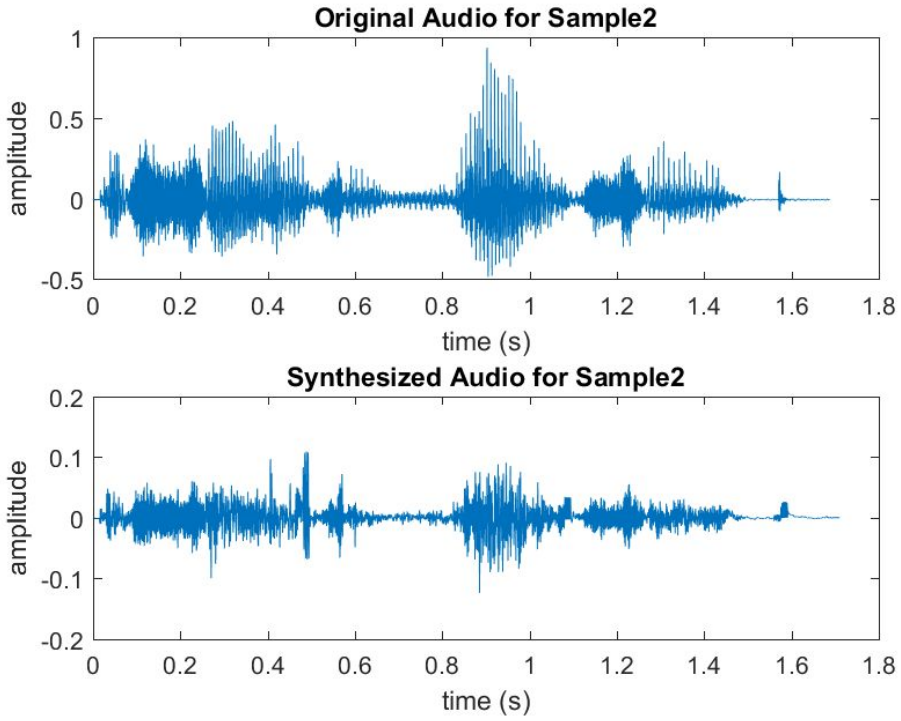
The following plot represents the original sentence audio and the received audio



Results

Synthesizer analysis

The plot below represents the original speech for ***Sample2.wav*** and the synthesized version of the signal. It is clear that the synthesized signal is well reconstructed with smaller magnitude than the original. The segment sounded very intelligible after playing it using ***soundsc.***



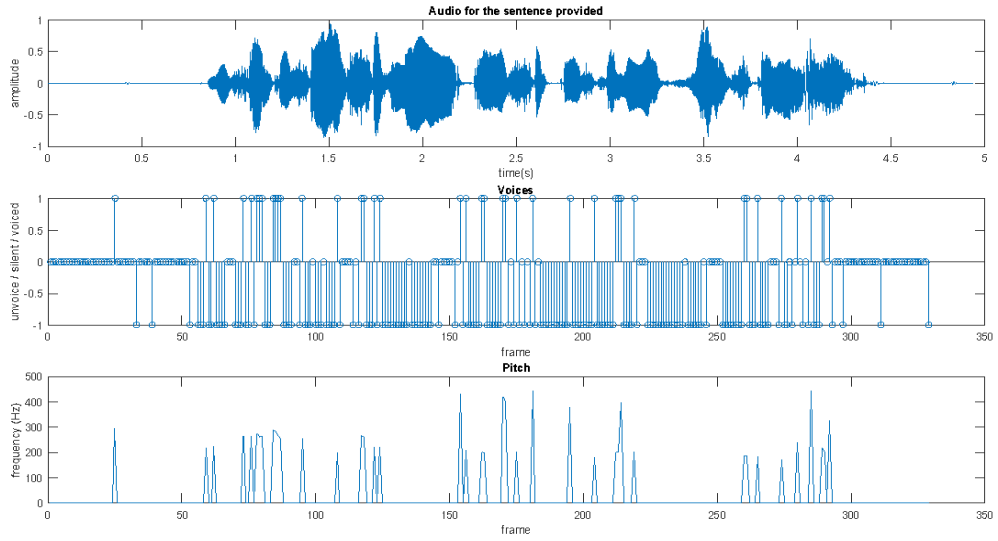
The results folder contains a plot of the original speech segment and the received segment for each of the audio files provided and my recorded segment.

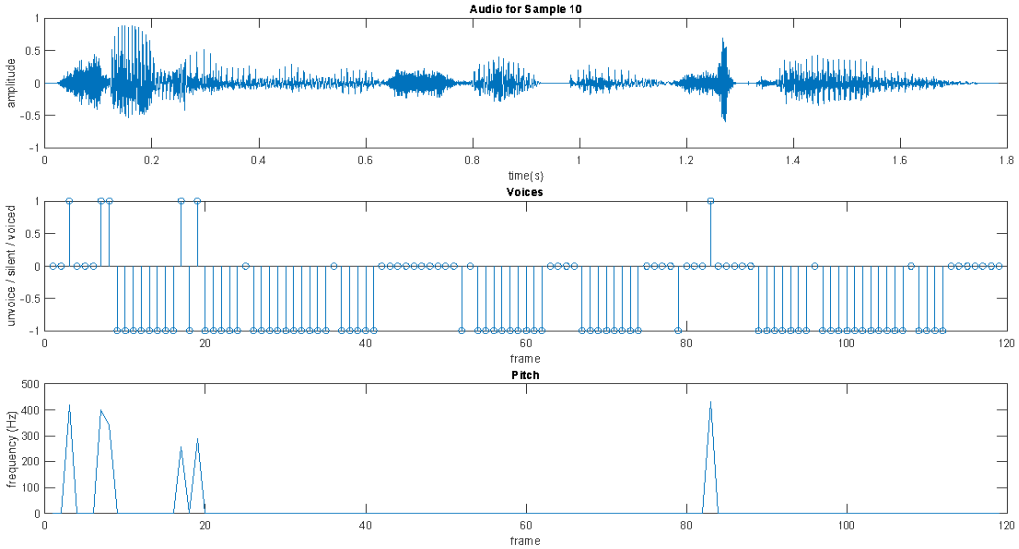
Bonus points

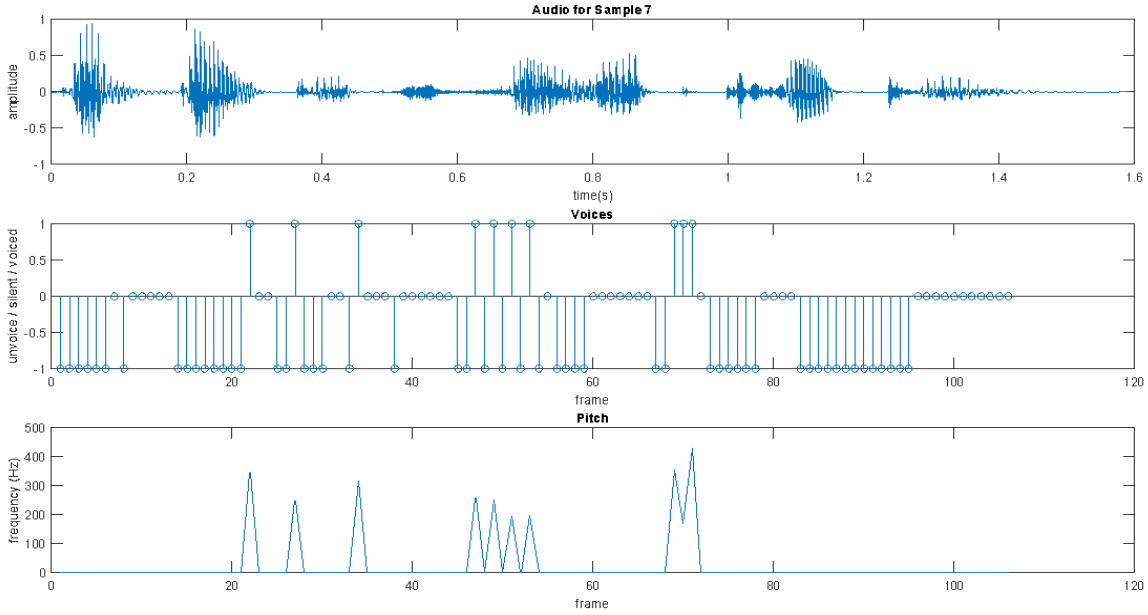
Both bonus questions were used in the LPC encoder function to acquire pitch and voicing information. The modified autocorrelation algorithm was used to determine the voicing and pitch in this project. I just split the pitch and the voicing into two separate functions as specified. They both take the signal and the sampling rate as immediate input arguments. Other arguments used (below) were declared as global variables for convenience

```
global ACORR_OFFSET     % Lag offset for autocorrelation processing
global CLIP_RATIO        % Center clipping ratio (of max amplitude)
global FRAME_TIME       % Frame duration
global OVERLAP           % Overlapped samples
global NOISE_GATE        % Noise gate for silence thresholding
global VOICE_THRESH     % Threshold for voiced frame detection
```

```
global MAX_PITCH        % Maximum accepted fundamental frequency
```






Audio for Sample 7
amplitude
time(s)
Voices
unvoice / silent / voiced
frame
Pitch
frequency (Hz)
frame