CS578: Project 1: Linear Predictive Coding

October 10th 2025

Delivery: October 31st 2025

Questions: yannis@csd.uoc.gr, kafentz@csd.uoc.gr, and/or hy578-list@csd.uoc.gr

During this project you will explore the Linear Prediction theory and an implementation in Python of a Linear Prediction based Analysis and Synthesis system of speech. In the provided Python code there are some empty command lines that are waiting for you to fill in. Once you do this, you can play with the code to do various speech modifications in an input speech signal.

In this project you will use the code in the Python file: `lpc_as_toyou.py`. You will play with a speech signal in a .WAV format named `speechsample.wav`, along with other files that are available on the project's webpage.

Specifically:

1. Analysis-Synthesis based on Linear Prediction

   Download the source code `lpc_as_toyou.py`/
   At this stage, the lines require your interventions are 50-54.

   - In 50, you require to compute the autocorrelation function of a given speech frame defined in line 46. You may use Numpy's function **correlate**. In 51, you need to pick only the positive lags of the result, given that, for a signal of length L, autocorrelation has a length of $2L + 1$, which corresponds to both positive and negative lags.
   - In line 52 you must compute the linear prediction coefficients using the Levinson recursion. Follow the steps in corresponding lecture and make sure you reverse the sign of all coefficients at the end. Do not forget to add 1.0 to your polynomial, that is, LPC coefficients should be $a_{LPC} = [1.0, \ -a_{LEV}]$.
   - In line 53 you must compute the gain of the LP filter, again as discussed in lectures.
   - In 54, you must compute the linear prediction error, or the residual signal, by inverse filtering the speech frame obtained in 46 through the estimated linear prediction filter. You may use SciPy's function **lfilter**.

Once done, you must be able to load the WAV file, perform analysis and synthesis frame-by-frame. At the end you have to save your computed speech signal (variable **out** which is the output. Listen to the original and the processed speech signal using the **play** command from **sounddevice** library.

Write down very briefly your listening impressions.

2. Follow frame-by-frame the analysis procedure

   In each analysis frame, compare the magnitude of the frequency response of the estimated linear prediction filter (using SciPy's command **freqz**) with the magnitude of the Fourier Transform (Numpy has an **fft** algorithm) of the speech frame.

   - Plot and compare on the same figure (use different colors) the two magnitude spectra. Do the above in both voiced and unvoiced areas.
   - Change the order of the Linear Prediction towards both directions (by increasing and decreasing the initial value (line 28).

   Save some interesting - according to you - plots and write down briefly your comments and observations.

3. Make modifications in the excitation signal

   - Create a whisper voice:

     Change the excitation signal (line 54) so that the synthesized output sounds like whispered speech. For this, you may want to use Numpy's **random** library to generate a sequence of random values from a normal distribution. Do the above for a lower and a higher linear prediction order than the one provided (24).

     Save the generated whispered speech files and write down briefly your comments and observations while explain shortly your choices.

   - Create a robotic voice:

     Change the excitation signal (line 54) so that the synthesized output sounds like a robotic voice that uses a constant pitch period (it is up to you to define that specific pitch period). For this, you may want to use Nympy's **ones** and **zeros** to construct an artificial excitation that contains $N-k$ zeros and k ones for a frame of length N. Make sure the ones will have some distance filled with zeros between them – you can experiment with the value of k but it is suggested to be between 2 and 4 (or, you can count the peaks of the residual signal

and place as many ones in the artificial excitation as there are peaks in the residual signal). Do the above for a lower and a higher linear prediction order than the one provided (24).

Save the generated robot speech files and write down briefly your comments and observations while explain shortly your choices.

4. Make modifications in the vocal tract

   In each frame, you can compute the poles of the estimated linear prediction filter using Numpy's **roots**. These roots are the poles of the AR filter. Select the three most significant (according to their magnitude) poles (along with their conjugates, since our filter is real and should stay that way) which we may assume they correspond to the first three formants. Check that these selected poles have frequencies close to the expected formants (for instance, by viewing the magnitude spectrum of the Fourier transform of the speech frame). Modify accordingly these formants – specifically, modify the angle of each pole by ± 10 or $\pm 20\%$ – so that you can get an elderly and a younger voice than the input speaker. To get back the filter once you have modified the poles, you may use Numpy's **convolve** to construct the AR polynomial coefficients.

   Save the generated speech files and write down briefly your comments and observations while explain shortly your choices.

5. Give us your voice

   Record a speech signal using your voice and choose one of the above modifications to be applied on your speech signal. You may use $f_s = 16000$ as sampling frequency during your recording.

   Save both signals and include them in your deliverables.

Answers may be given in Greek or in English. Return the functions you wrote by yourself plus the original (initial) Python script with the requested lines filled in, along with a report that discusses your observations.