DAAP Homework

Professors: Pezzoli Mirco, Massi Oliviero, Miotello Federico

Students: Di Giovanni Matteo, Mancuso Alessandro

April 12, 2025

1 Abstract

The aim of this Homework Assignment is to explore the powerfulness of Linear Predictive Coding (**LPC**) as a tool to perform speech coding, through a didactic yet insightful application.

Developed by Texas Instruments in the late '70s and marketed as a toy, the `"Speak & Spell"` was one of the first fully functional teaching machines. It was designed to help children learn correct spelling in an engaging and effective way. Once turned on and the game started, the device would pronounce a word that the user had to type using the built-in keyboard to check if the spelling was correct.

FIGURE 1: *Speak & Spell toy.*

At its core, the `"Speak & Spell"` used the `TMC0280` linear predictive coding speech synthesizer, a dedicated integrated circuit that implemented PE-LPC

(Pulse Excitation LPC). Words were generated by a lattice filter, and one of the main challenges was recording sounds in a way that allowed their reuse for multiple pronunciations. The synthesizer was driven by either a `128 kbits` excitation ROM or an LFSR (Linear-Feedback Shift Register).

What are the goals?

- Implement the LPC-10 coding algorithm in MATLAB, including the computation of LPC coefficients from speech frames;
- Reconstruct the speech signal from the LPC parameters;
- Evaluate the quality of the reconstructed signal compared to the original.

In order to perform these tasks, a modular approach was used by writing independent scripts for specific functions that we will now go through.

2 Auxiliary Functions

2.1 Pitch Detection with AMDF (Average Magnitude Difference Function)

This function receives as input a frame to analyze and returns as output the estimated pitch.

We exploited the provided paper to retrieve the following mathematical formula:

$$\mathrm{D}_m = \frac{1}{L} \sum_{n=1}^{L} |x(n) - x(n+m)|$$

where D_m is the result we want to obtain, L is the frame length and m are the indexes of the lags (with $m = 0, 1, ..., m_{\max}$).

We used two nested loops, the first carries out the summation (the inner one), the second shifts the index of the function and normalizes the result at each iteration.

To correctly delay the signal of "m" samples we padded the starting frame array with zeros (with a number equals to the number of lags) and we used the `circshift()` function to translate the frame (since `e` now finishes with lags zeros we don't have artifacts coming from the circular shift; we stop each time at `len` computations).

To correctly return the pitch we used the built-in function `find()` to compute the minimum of our **AMDF**; in doing so it could be possible to retrieve more than one value.

To avoid the problem it could be wise to extract only the first obtained one (in particular the range we gave to the function is restricted and does not contain index 1, due to the fact that $|x(n) - x(n)|$ gives always zero).

Why this function?

It's an easier and faster way of exploiting periodicity in a signal, avoiding the computation of the autocorrelation function (that would involve sums and multiplications instead of just sums and differences), where, shifting the signal and subtracting it from its original version we try to see if it repeats somehow similar to itself (minimum in the function), thus exhibiting periodicity.

2.2 Generate Excitation Signal

It takes as input arguments:

- The `voicedIdx` array that contains for each frame a value 0 or 1 indicating whether it is voiced or unvoiced;
- The `gain` array that contains for each frame the specific gain to use (recalling the model studied during lessons, it corresponds to the G factor that multiplies the excitation signal);
- The `pitch` array that contains for each frame its periodicity;
- `winLen` which is a scalar equal to the length of the window that will contain the excitation signal for each frame.

It will produce as output the excitation signal array that the decoder will use to reconstruct the speech signal.

We started by establishing the number of frames that will be considered during the computation by exploiting the length of the `voicedIdx` array.

We then created an excitation signal starting from a vector of zeros of length $nFrames * winLen$.

Through a single loop on all possible frames (with bounds indexes evaluated at each iteration) we filled the excitation array.

According to theory, if the frame was voiced we used a train of pulses, otherwise we used white noise (through a zero mean gaussian, as the variance is inside gains).

2.3 Voiced Frame Detection

This function aims at estimating if a segment is voiced or not. To do so it analyzes each frame (in a specific signal block there could be more than one) and evaluates the zero crossing:

- if the frame has very low energy or shows signs of a noisy behavior, we can conclude it is unvoiced;
- otherwise, it means the frame likely contains periodicity, and it is marked as voiced.

Some additional comments were added in the script for our own understanding (since the function was given as it is, we will avoid inserting further details in the report).

3 Scripts

3.1 Encoder

The function inside the "`encoder.m`" script receives as input the file name of the audio that we want to process and two boolean variables responsible for the plotting of results and additional low-passing for the prediction error.

It performs the computation of the needed parameters for the LPC-10 algorithm:

- The vector of coefficients (in our case a cell array that will be adjusted dynamically according to the order of the frame);
- The gains vector to later use during the computation of the excitement signals;
- The prediction error and the MSE, to evaluate the quality of the approximation;
- The pitch vector that will contain periodicities of the frames.

About LPC
What is LPC concisely?

Linear Predictive Coding is a technique employed to represent a speech signal by modeling each sample as a linear combination of past samples.

In particular, we can assume that a signal sample $s(n)$ at time instant n can be well approximated by:

$$s(n) \approx \sum_{k=1}^{p} a_k s(n-k)$$

Where p is the prediction order and the coefficients a_k are assumed to be constant in the selected analysis window.

If we further assume that the signal can be modeled as an autoregressive stochastic process, then we can write:

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n)$$

G is a gain parameter (that we have to estimate along with the coefficients a_k) and $u(n)$ is a white noise (the excitation signal that we will compute during the decoding stage).

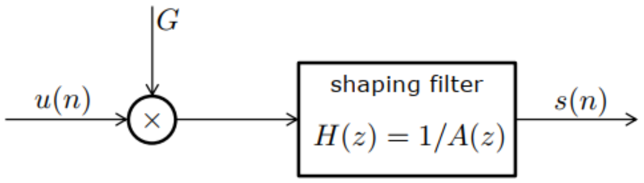

FIGURE 2: *Block Diagram for LPC system*

About the parameter P

P is the order of the LPC, the number of coefficients and past samples used to calculate each prediction. The practical meaning of P is that it determines the smoothness of the LPC spectrum. This means that a higher value of P will result in a more accurate frequency spectrum of the reconstructed signal, at the price of additional higher computational cost.

In speech synthesis though, the goal is to capture the spectral envelope in terms of peaks (fundamentals) and not the fine structure of itself, so it's necessary to find a balanced value of P that fits our needs. The general use of thumb is having a value of P according to:

$$\boxed{\frac{f_s}{1000} \leq p \leq \frac{f_s}{1000} + 4}$$

FIGURE 3: *(Reference: Slide 21 in the LPC theory slideset)*

In our case, having a sampling frequency of `8 kHz`, the value of P should be between 8 and 12. For this reason, it makes sense to have a value of P equal to 10 for voiced samples, as it allows us to capture the details of the pitch. To optimize the resources and computational time, a lower value (4) has been chosen for the unvoiced samples, as we are less interested in their frequency content and we can smooth out their representation.

`Note`: We also tried using $P = 10$ for unvoiced segments to see if there were any tangible improvements and it was the case. However, since the assignment was clear on the order to use (it says explicitly LPC-10 version), we left the original subdivision $P = 10$ for voiced ones and $P = 4$ for unvoiced ones.

Before starting to operate directly on the acquired signal we converted it to mono by just taking either one of the two channels and normalized it with respect to the peak value; a pre-emphasis filter (a 1st order FIR high-pass) was applied to remove excessive low-end to the signal and compensate for scarce intelligibility in the higher register.

As suggested, we designed an optional low-pass filter, with cutoff frequency of `800Hz`, through the `fir1()` built-in function (order 10 seemed to be reasonable for the purpose, it doesn't add a lot of delay) with which it's possible to smooth out the prediction error.

`Note`: We tried both ways and strongly recommend applying it since it performs best in the overall result (less clicks in the reconstructed audio track, it's easier to detect periodicity).

The core of the script is the main for loop in which you go frame by frame and compute all parameters.

The autocorrelation is evaluated via means of the `xcorr()` function (we're interested only in the positive part since it's a symmetric function), while the coefficients are evaluated with the help of the `levinson()` function that implements the `"Levinson-Durbin"` recursion algorithm (inverting in an efficient and fast way the autocorrelation matrix).

An additional check was included to avoid `NaN` values; this could arise if the frame is made just of silence (highly unpredictable noise) or when the `Levinson` algorithm has failed to retrieve the coefficients. In this case we set all the LPC coefficients except the first one to zero.

If the boolean variable `doPlot` is set to true you can access a plotting section of the script in which the `FFT` of the Shaping Filter, the signal and prediction error are evaluated and plotted in magnitude frame by frame.

Since the `freqz()` function returns as a second argument the frequency (omega) axis (and we've passed as parameter the sampling frequency, so no conversion is needed from rad/s to Hz) there was no need to define the axis explicitly again.

Of particular interest in the graphs is that it is possible to see the difference between the Original spectrum, depicted in blue, and the LPC estimate, depicted in red, that captures the main frequency informations (raising the order of the algorithm would make it follow the curve better at the cost of higher computational time; however, since we're not keen in reproducing also the valleys but just the peaks, the result shown below is pretty satisfying).

`Note`: If the additional lowpass on the error is activated and so is the plot section, a 4th graph showing the filtered prediction error will appear during voiced frames (smoother representation of itself).

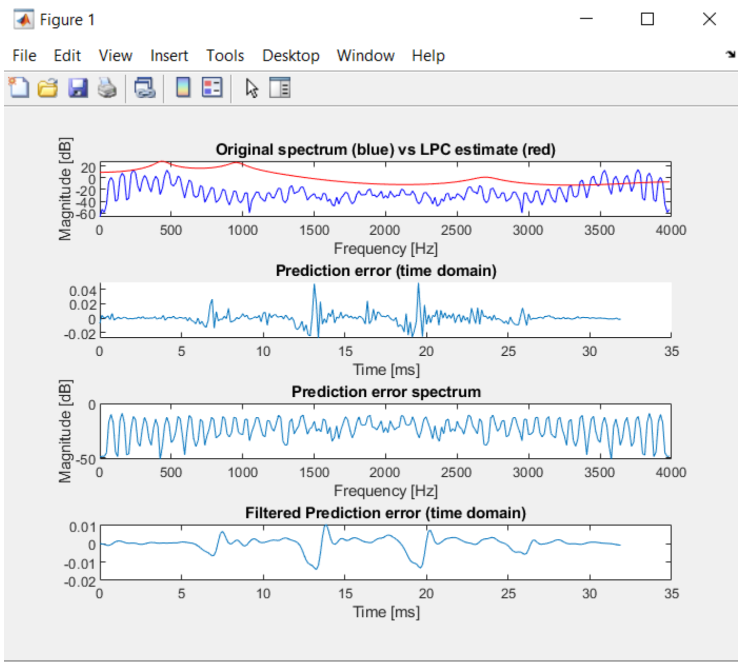

FIGURE 4: *Stacked plotting for the "O" vowel (The last plot is the "smoothened prediction error").*

The last step before exiting the for loop was computing the gain; since the part for the voiced fragments was already completed, we just wrote the section regarding unvoiced ones. It was just a matter of taking the MSE (variance of the zero mean white noise) and then the square root of it, since we're interested in the standard deviation.

The final issue to address was to save data in a `.mat` file to be opened later in the Decoder part.

3.2 Decoder

In the script "`decoder.m`", the load function retrieves the LPC parameters computed during the encoding stage through the name of the `.mat` file, and uses them to reconstruct an estimate of the original signal.

The function `generateexcitationsignal()` is called to create an excitation signal that will then be modeled according to the coefficients.

In our model (and in speech synthesis in general), the shaping filter represents the transfer function of the vocal tract, while the excitation signal represents the source at the glottis, thus recreating an approximation of human speech.

To reconstruct the signal we operated following the Overlap and Add framework paying attention to the satisfy the COLA condition (Constant - OLA, where the overlapping window has to sum to a constant value except for some transient behavior).

Addressing the suggestion on which STFT parameters to choose we decided to leave both the window length (N) and the hop size (H) to be the same value. However, it would be interesting to test some other combinations respecting the constraint $H < N$ and in particular, considering an Hanning window, taking an H value of $N/2$ or $N/4$.

We also applied a de-emphasis filter to attenuate excessive noise (similar to the one in the decoder, with the exception that we're taking the abs of the filter itself, so the zero moves being still minimum phase).

The final task is to play the reconstruction using the `soundsc()` function (that automatically normalizes the audio to reproduce in the range $[-1, 1]$) with specified sampling frequency `Fs` (otherwise it would be `8192 Hz` by default).

3.3 Test

Last operation was to build a script to try out everything we did in a single procedure to check the quality of the implemented algorithm.

In the script "`test_lpc.m`" lies an interactive console program that allows the user to test the performance of the encoder and decoder when given different tasks.

When the script is ran, the user can choose between two modes:

- Playing a word or a single letter;
- Playing whole recordings.

After validating all inputs, the script performs the given task using the functions `encode()` and `decode()` defined in the above commented scripts.

Playing letter by letter

The user can type a maximum of 16 letters taken from the alphabet.
The input string is then analyzed letter by letter and encoding is computed for each individual one. After encoding all letters, these are decoded and played one by one. In order to have a smoother encoding/decoding process, a caching system was implemented to save all encoded `.m` files into a `/encoded` folder.

This version gives overall poor results due to the different duration of the letters. A possible way to fix this would be to manually act on the given samples and crop them to be all of the same length/duration (this however could lead to artifacts such as clicks in the reconstructed speech signal).

Playing whole recordings

The user can only select one of the possible recordings present in the `/input` folder. After that, the whole recording is encoded, decoded and then played. This performs quite good, in particular with the `"ces.wav"` audio file; on the contrary the `"Invo.mp3"`, having a much lower frequency register and muffled recorded overall sound, is severely distorted in the reproduction (becomes clearer if we increase the order of LPC up to $P = 128$).

4 Considerations

In conclusion, we can confidently assess that the desired result was achieved, even though the overall quality of the reconstruction is not always the best in every situation, letter, word or recording. For example, encoding/decoding the single `"e"` letter doesn't provide a meaningful output, it is just noise (as it is not even recognized as a vowel, a voiced sound).

Changing the order of the algorithm at the small price of more computational time would surely help us achieve better performances. Also working with a standardized alphabet, in terms of duration and pronunciation, would be better for the sake of the toy implementation.