Speech Signal Processing
Exercise 3 — Linear Prediction

Timo Gerkmann, Kristina Tesch, Danilo Oliveira

In this exercise session, we will be working on linear prediction and its application to speech processing. In this exercise, the file `speech1.wav` is used as audio input.

1 Linear Prediction Basics

According to the source-filter model in Figure 1, a speech signal $s(n)$ can be modeled as a filtered version of some weighted excitation $v(n) = ge(n)$, which can either be a noise sequence or an impulse train. The air flow of the excitation signal passes the human vocal tract which induces a modification of the excitation signal. This modification can be described by a filter process. The transfer function of the filter is denoted as $H(z)$ in the z-domain. Commonly, $H(z)$ is assumed to be an all-pole filter of order M, giving

$$H(z) = \frac{1}{1 + \sum_{i=1}^{M} a_i z^{-i}}, \tag{1}$$

with filter coefficients a_i. In the time domain, this can be formulated as follows:

$$s(n) = v(n) - a_1 s(n-1) - a_2 s(n-2) - \ldots - a_M s(n-M) = v(n) - \sum_{i=1}^{M} a_i s(n-i). \tag{2}$$

In equation (2) it can be seen that the speech sample $s(n)$ is given as a linear combination of previous speech samples and the current excitation $v(n)$. Our goal now is to estimate the coefficients a_i given that we know all relevant speech samples $s(n-M) \ldots s(n)$.

It can be shown that estimates of a_i, denoted as $\hat{a}_i$, can be obtained by minimizing the statistical expectation of the squared prediction error $\epsilon(n)$, i. e.,

$$\underset{\hat{a}_i}{\operatorname{argmin}}\, E\{\epsilon^2(n)\}, \text{with} \qquad \epsilon(n) = s(n) - \hat{s}(n) = s(n) + \sum_{i=1}^{M} \hat{a}_i s(n-i). \tag{3}$$

In other words, we have to find those $\hat{a}_i$ that give us a prediction of $s(n)$ that is as close as possible to the true, observed $s(n)$ in the minimum mean squared error (MMSE)-sense. The solution to this minimization problem is given by a set of linear equations of the form

$$-\begin{bmatrix} \varphi_s(0) & \varphi_s(1) & \cdots & \varphi_s(M-1) \\ \varphi_s(1) & \varphi_s(0) & \cdots & \varphi_s(M-2) \\ \varphi_s(2) & \varphi_s(1) & \cdots & \varphi_s(M-3) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_s(M-1) & \varphi_s(M-2) & \cdots & \varphi_s(0) \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \\ \vdots \\ \hat{a}_M \end{bmatrix} = \begin{bmatrix} \varphi_s(1) \\ \varphi_s(2) \\ \\ \vdots \\ \varphi_s(M) \end{bmatrix} \tag{4}$$

$$-\mathbf{R}_s \hat{\mathbf{a}} = \varphi_{\mathbf{s}},$$

where $\mathbf{R}_s$ denotes a $M \times M$ Toeplitz matrix and $\varphi_{\mathbf{s}}$ is the correlation vector. The estimates of a_i that we obtain by solving (4) are referred to as linear prediction coefficients (LPCs). As we can see, the estimation of the LPCs entirely depends on the auto-correlation of the speech signal.

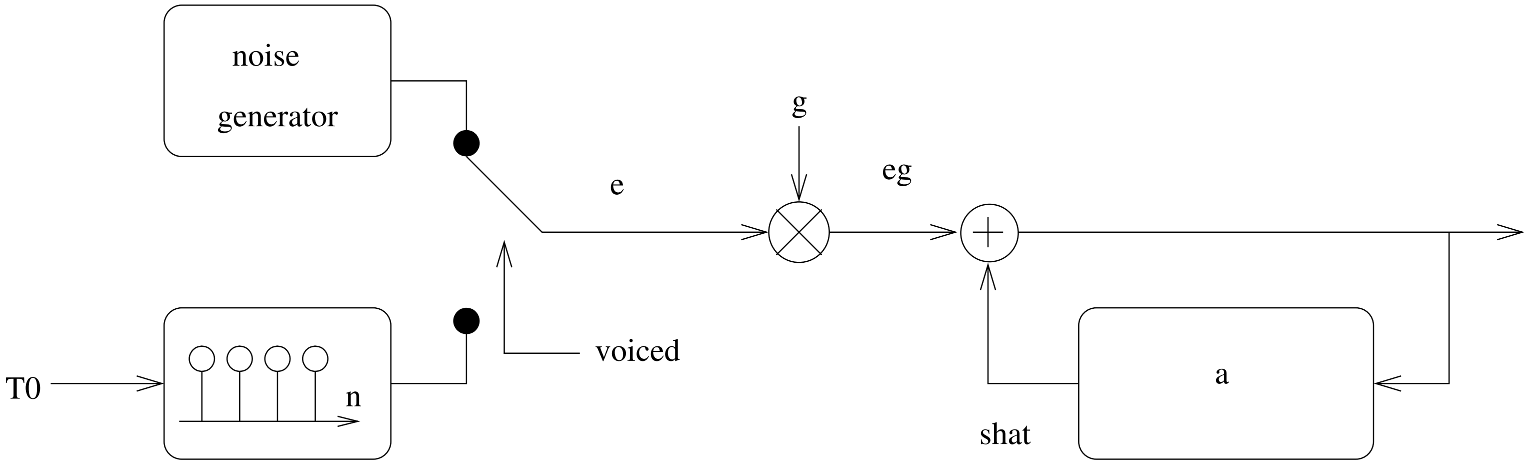

Fig. 1: Block diagram of the source filter model with sample index n, excitation signal $e(n)$, gain g, fundamental frequency f_0, fundamental Period T_0, LPCs $\hat{\mathbf{a}}$, and speech signal $x(n)$.

2 Assignments

1. Load the speech file `speech1.wav`.

2. Select one unvoiced and one voiced speech segment from the signal, each with a length of 32 ms. You may reuse your segmentation from Assignment 3 of Exercise 1 and/or your knowledge about the speech signal obtained in assignment 1b) of Exercise 1. Apply a Hann window of the same length to both segments.

3. Compute the $M = 12$-order LP coefficients by solving equation 4. Use the functions `np.correlate` and `scipy.linalg.solve_toeplitz` to compute the autocorrelation vector $\varphi_\mathbf{s}$ and the correlation matrix $\mathbf{R}_s$ respectively. Store the coefficients in a vector `a`.

4. a) Make a plot of the frequency response (amplitude as well as phase) of the estimated vocal tract filter $H(z)$ for both, the unvoiced and the voiced speech segment. To do this, the command `scipy.signal.freqz(1, np.concatenate(([1], a)), numPoints, whole=True, fs=sampling_freq)` might be helpful. For the number of frequency-points (`numPoints`), use the segment length in samples. Make sure that the axis descriptions of your plots is meaningful.

   b) Why do you use `np.concatenate(([1], a))` and not only `a`?

5. Compute the discrete Fourier transform (DFT) of the windowed segments using `S=np.fft.rfft(...)`. Plot the amplitude of `S` in dB together with the amplitude of the corresponding filter $H(z)$ in dB in one plot.

6. a) For both segments, compute the residual signal by using the inverse filtering statement `e = scipy.signal.lfilter(np.concatenate(([1], a)), 1, s)`. Plot the residual signal `e` together with the corresponding signal segment.

   b) Explain differences in `e` between the voiced and unvoiced segment.

   c) Explain why `scipy.signal.lfilter(np.concatenate(([1], a)), 1, s)` yields the residual signal.

7. a) Why are the logarithmic amplitudes of `H` and `S` (plots of assignment 5) not on the same level?

   b) How can you modify `H` to achieve a better match? **Hint:** Experiment with the energy of the residual `e`.

   c) For the **voiced** speech segment, plot the amplitude of the modified filter H together with the amplitude of S in dB to check if your modification is correct.

8. Play with the order of the predictor ($M = 2 \cdots 20$). Describe differences in $H(z)$ and explain reasons for that.

9. From the speech production model it is known that speech undergoes a spectral tilt of -6 dB/octave. To counteract this effect, a pre-emphasis filter of the following form is used

$$y(n) = s(n) - \alpha s(n-1). \tag{5}$$

a) Compute the LP coefficients for the pre-emphasized **voiced** speech segment using function `scipy.signal.lfilter` and $\alpha = 0.95$. Compare the results with and without pre-emphasis.

b) What is the advantage of pre-emphasizing the speech signal?