Exercise #2:
Basic signal features, linear prediction, analysis-synthesis

Exercise summary:

This second exercise focuses on basic hands-on time-frequency analysis of speech signals, on linear prediction, and LPC-based signal analysis (LP estimation) and reconstruction (LP synthesis).

Data and existing resources: Speech audio waveform `251-136532-0016.flac`. Pre-defined MATLAB script frames `E2_main.m`, `getlpc.m` and `lpcResynthesis.m`.

Software: MATLAB.

Tasks:
1) Spectrogram, energy, and zero-crossing rate
2) Linear prediction
3) Signal re-synthesis with LPC

Deliverables: A report (*`E2_surname_report`*`.pdf`) with a title page including your name and student number + content pages with the requested figures from Tasks 1–2 and written answers to questions. In addition, two `.wav` files `synth_residual.wav` and `synth_impulse.wav` from Task 3), and three completed MATLAB script files `E2_main.m`, `getlpc.m`, `lpcResynthesis.m` all archived to `E2_firstname_surname_studentID.zip` and uploaded to Moodle.

Learning goals: Hands-on speech analysis, windowing, spectrum and spectrogram, linear prediction, analysis-synthesis.

Note 1: All exercises of the course, including this one, will consist of a written report and additional files (e.g., code, sounds) that are to be created and submitted for evaluation. In the report, always add your name and student ID to the beginning of the report. Use complete English sentences or paragraphs of text to answer the exercise questions, and number each response according to the question numbers in the exercise instructions. Some of the questions have strictly correct answers while others can have multiple valid responses.

Note 2: The exercise will be semi-automatically scored based on the variables and functions defined in the MATLAB code template. Do not change the naming of the key variables or specifications of function templates and calls defined. If you fail to complete the full assignment, in order to get partial points of correct parts, make sure your code runs up to the point that you managed to complete (e.g., getlpc() might output an intermediate variable **R** even if the final output **a** is wrong).

Task 1: Spectrogram, energy, and zero-crossing rate

Start by reading and resampling the audio signal located at `data/251-136532-0016.flac` (Step 0 in the pre-provided code in E2_main.m).

Calculate the following features for the entire utterance using 20-ms sliding window length and a 10-ms step size.

Task 1.1: logarithmic magnitude spectrogram

Task 1.2: logarithmic signal energy

Task 1.3: zero-crossing rate (ZCR).

Use Hamming windowing for the spectrogram and rectangular windowing for energy and ZCR.

Task 1.4: Create a plot with three panels: logarithmic magnitude spectrogram on the top, log-energy on the middle, and zero-crossing rate at the bottom. Remember to include correct & informative x- and y-axis labels. See Fig. 1 for an example.

Guidance: Spectrogram is based on frame-level magnitude spectra (Exercise 1), now calculated for each window position in the signal and plotted in 2-d.

Linear signal energy for x can be calculated as

$$E = \sum_i x_i^2 \qquad (3.1)$$

and log-energy as

$$\mathrm{E}_{log} = 10 \cdot \log_{10} \sum_i x_i^2 \ . \qquad (3.2)$$

Zero-crossing rate $ZCR \in [0, 1]$ is defined as the *rate* of signal sign-changes during the signal frame, where 0 stands for no change, and 1 corresponds to sign change between every consecutive sample. Usually: zcr = N_crossings / (number of samples-1).

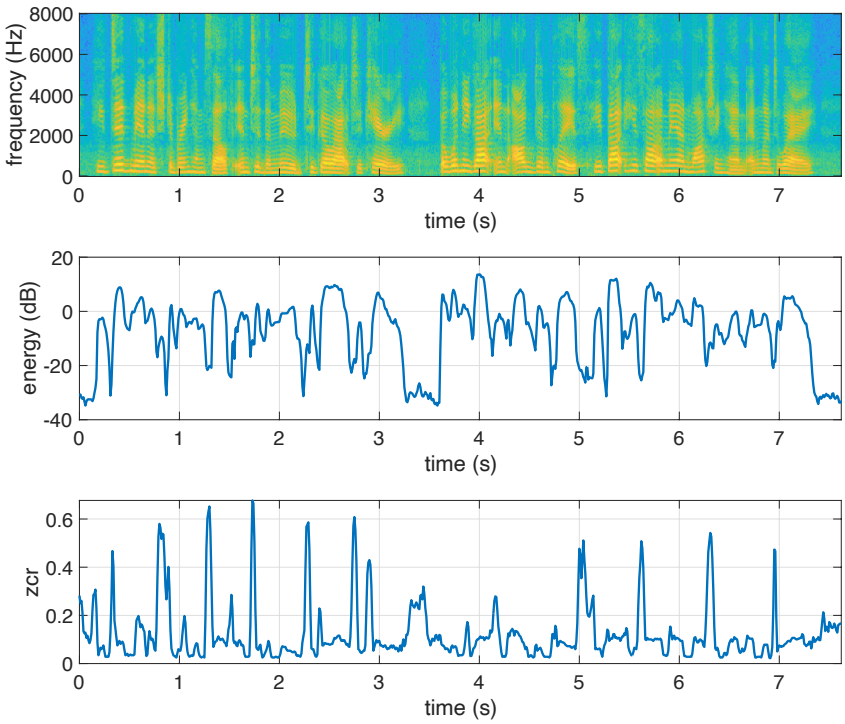

Figure 1: An example of a spectrogram (top), energy envelope (middle) and zero-crossing rate (bottom) for an English utterance.

Question 1.1: Does the use of a non-rectangular window function (e.g., Hamming window) affect zero-crossing rate estimate? How about energy?

Task 2: Linear prediction

Task 2.1: Implement LPC estimation using autocorrelation method to `getlpc()` (see the code and guidance below for help).

Task 2.2: Calculate LPC coefficients using 20-ms Hamming window and 10-ms steps from a pre-emphasized speech signal for the whole utterance and using model order of $p = 20$. Also calculate corresponding residual signal by filtering the original windowed signal with the corresponding LPC analysis filter. Calculate and store LPC synthesis filter logarithmic magnitude spectrogram and LPC residual spectrogram.

Task 2.3: Create a plot that has three panels: the top panel shows original magnitude spectrogram of the utterance, the middle pane shows logarithmic magnitude spectrogram of the LPC filters, and the bottom one shows logarithmic magnitude spectrogram of the LPC residual.

Add the plot together with your answer to question 2.1 to your report.

Guidance:

Pre-emphasis: Before calculating LPC, it is convenient to emphasize higher frequencies of the signal using a so-called pre-emphasis filter. A typical FIR pre-emphasis filter has form:

$$H(z) = \frac{1 - 0.95z^{-1}}{1} \quad .(4.1)$$

This boosts higher frequencies approx. 6 dB per octave, thereby counterbalancing the effects of spectral decay of -6 dB per octave in typical voiced speech.

LPC-analysis: LPC-analysis assumes that each observed speech signal frame s_n has been created with an IIR *synthesis filter* $H(z)$ of form

$$H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} = \frac{G}{A(z)} \quad (4.2)$$

where G is the gain, and the transfer function $A(z)$ is called as the *inverse filter*. In time-domain, this can be written as

$$s_n = \sum_{k=1}^{p} a_k \, s_{n-k} + G u_n \quad (4.3)$$

where s_n is the *system response* and u_n the *excitation*. According to Eq. (4.3), a *prediction* for each new sample can be made as a weighted sum of previous samples:

$$\hat{s}_n = \sum_{k=1}^{p} a_k \, s_{n-k} \quad (4.4)$$

hence the name *linear prediction.* Coefficients a_k, $1 \leq k \leq p$, are referred to as *prediction coefficients.* Note that inverse filter coefficients $A(z)$ in Eq. (4.2) can be derived from the prediction coefficients by simply adding 1 as an additional first coefficient and inverting the sign of all the prediction coefficients.

Now, prediction error e_n, aka. *residual*, of each sample can be written as

$$e_n = s_n - \hat{s}_n = s_n - \sum_{k=1}^{p} a_k \, s_{n-k} \quad (4.5)$$

In autocorrelation LPC, the windowed signal s_n is assumed to be infinite even if it only differs from zero for the N windowed signal values. In order to solve the prediction coefficients, the goal is to minimize the squared sum of the residual, where the sum is defined as:

$$E = \sum_{n=-\infty}^{\infty} e_n^2 = \sum_{n=-\infty}^{\infty} (s_n - \sum_{k=1}^{p} a_k \, s_{n-k})^2 \quad (4.6)$$

The solution to $\arg_{a_k} \min \{E\}$ can be found by finding where partial derivatives of E with respect to the prediction coefficients a_k are zero:

$$\frac{\partial E}{\partial a_k} = 0, \qquad 1 \leq k \leq p \quad (4.7)$$

This leads to so-called normal equations (Rabiner & Schafer, 1978, pp. 400-402):

$$\sum_{k=1}^{p} a_k r(i-k) = r(i) \quad (4.8)$$

where $r(i)$ are autocorrelation terms, formally:

$$r(i) = \sum_{n=-\infty}^{\infty} s_n s_{n-i} \quad (4.9)$$

Since autocorrelation is even $r(-i) = r(i)$, and in our case only differs from zero for N samples and is needed up to p in Eq. (4.8), it is sufficient to calculate

$$r(i) = \sum_{n=i}^{N-1} s_n s_{n-i}, \quad 0 \leq i \leq p \quad (4.10)$$

Normal equations in Eq. (4.8) can then be written in matrix form as

$$\begin{bmatrix} r(0) & r(1) & r(2) & \dots & r(p-1) \\ r(1) & r(0) & r(1) & \dots & r(p-2) \\ r(2) & r(1) & r(0) & \dots & r(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & r(p-3) & \dots & r(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ \vdots \\ r(p) \end{bmatrix}, \quad (4.11)$$

which can be denoted in vector notation as:

$$\mathbf{Ra} = \mathbf{r} \quad (4.12)$$

In order to solve Eq. (4.12) for prediction filter coefficients **a**, we only need to create autocorrelation matrix **R** and autocorrelation vector **r** (note the difference in indexing!), and then solve

$$\mathbf{a} = \mathbf{R}^{-1}\mathbf{r} \quad (4.13)$$

where $\mathbf{R}^{-1}$ is the inverse of **R**.

It is worth mentioning that since **R** is a symmetric Toeplitz matrix (all diagonal values identical), Eq. (4.12) could be also solved for **a** using so-called *Levinson-Durbin recursion*. However, this recursion is not supposed to be used in the present exercise.

Finally, gain *G* can be calculated as (Rabiner & Schafer, 1978; pp. 404–407):

$$G = \sqrt{r(0) - \sum_{k=1}^{p} a_k r(k)} \quad . \quad (4.14)$$

Note that *G* is just a scalar and does not affect the shape of the spectrum. It simply reflects the energy difference between the original signal and the signal structure captured by LPC.

Note that solution to Eq. 4.13 returns LPC *prediction filter coefficients* (a FIR filter), whereas we are often interested in spectral analysis or synthesis of speech using the LPC. Therefore conversion to IIR LPC *synthesis filter* can be obtained by:

$$\mathbf{a}_{\text{synth}} = [1\ -a_1\ -a_2\ \ldots\ -a_p] \quad (4.15)$$

In order to calculate logarithmic magnitude spectrum of the synthesis LPC filter for the original window of *N* samples, one can simply take FFT of the A(*z*) using *N*-point FFT (i.e., padding the A(z) with zeros to have *N*-length vector). Then the logarithmic spectrum of the synthesis filter is obtained as:

$$20 \cdot \log_{10}\{H(k)\} = 20 \cdot \log_{10}\left\{\frac{G}{|A(k)|}\right\} = 20 \cdot \log_{10}(G) - 20 \cdot \log_{10}\{|A(k)|\} \quad (4.16)$$

***MATLAB-functions*:** `filter(B,A)` –function allows you to operate filters of form *H*(*z*) = *B*(*z*)/*A*(*z*) without having to manually implement the filtering recursion.

Cheat-sheet: you can check your `getlpc()` results against the built-in Levinson-Durbin based `lpc()` –function, but you are not allowed (and should not anyway!) use pieces of `lpc()` in your own implementation.

Question 2.1: Qualitatively describe the characteristics of the resulting LPC and residual spectrograms (e.g., compared to each other and to the original signal spectrogram). How do the two change if you start increasing the LPC order from the default 20?

Task 3) Signal re-synthesis with LPC

Task 3.1: Use the LPC-coefficients, residual, and gain information extracted in Task 2) to synthesize the original utterance back from the components. Use two types of excitation:

A) original residual as the excitation, and
B) impulse train as the excitation, using a meaningful F0 of your choice. See Appendix A for conceptual guidance.

Follow the provided skeleton code in `lpcResynthesis.m` for more detail step-by-step help. Save the synthesized waveforms as form `synth_residual.wav` and `synth_impulse.wav` (both mono 16 kHz).

Task 3.2 (optional, not graded): you can try to combine impulse-based and noise-based excitation to have different excitation style for voiced and unvoiced frames. For instance, you can use zero crossing rate (ZCR) extracted in Task 1 to roughly determine voiced and unvoiced segments, as ZCR for unvoiced frames is expected to be higher than that of voiced frames.

Guidance:

For synthesis, use the overlap-add principle. Step size of the LPC extraction window is 10-ms (160 samples at 16 kHz), whereas each vector of LPC coefficients encodes 20-ms of speech (i.e., window length of N = 320 samples). In the overlap-add synthesis, the signal is created using the same 10-ms window hop size as in the analysis stage and the overlapping parts are added together. If window size is twice the step size, each final sample will consist of two added (overlapping) samples from consecutive window positions.

When using impulse train excitation, a long continuous impulse train should be first generated for the whole utterance instead of generating a separate pulse train for each window position. Then, for each signal position to be synthesized, the corresponding segment of the long excitation is always used in the synthesis filtering step. This ensures that the phase of the impulse train does not change between consecutive synthesis windows, improving sound quality. To create an impulse train, you can create a zero-signal of suitable length (number of samples) and then insert 1s at intervals corresponding to T = 1/F0.

Gain is not needed for synthesis when using the original residual of the utterance, as the energy of the signal not captured by the LPC filter is, by definition, present in the residual. In contrast, impulse-based synthesis has a constant-energy excitation signal (the impulse train), and hence gain is needed.

Remember that inverse filter coefficients (IIR) are used for synthesis and prediction coefficients (FIR) for residual calculation (and hence also for deriving the LPC-solution).

Also, remember to remove the effect of pre-emphasis from the final synthesized signals!

Question 3.1: In terms of speech quality, what is the key difference between the residual and impulse -based re-synthesis of the utterance?

Question 3.2: What are potential advantages of separating the vocal tract (LPC coefficients) and the excitation of the speech signal?

Returning the exercise

Collect all produces plots and answers to your questions into a report document *E2_surname_report*.pdf. Combine the report with the synthesized synth_residual.wav and synth_impulse.wav, and the three completed MATLAB script files E2_main.m, getlpc.m, and lpcResynthesis.m into a .zip file named E2_firstname_surname_studentID.zip, and upload the package to Moodle.

Appendix A: Conceptual background, excitation generation

In normal speech, excitation of voiced speech is generated by sustained lung pressure, which, due to Bernoulli effect, causes vibration of the vocal folds in the larynx. This results in a periodic signal consisting of pulse-like “puffs of air”, technically characterized by its *glottal volume velocity waveform*. Inverse of the period *T* of this pulsation is known as the fundamental frequency (F0) of speech. This excitation signal transmits energy to the vocal tract, which then filters the excitation according to its geometric and physiological properties.

In speech processing, the simplest approximation for a glottal excitation is simply an impulse train. An impulse train consists of zeros and ones, where the ones are spaced at every *T* in time in order to obtain the desired F0 for the signal.

For more natural sounding speech, glottal excitation could also be modeled as *pulses* instead of impulses, especially in terms of temporally *asymmetric pulses* similar to real glottal volume velocity waveforms (see, e.g., Liljencrantz-Fant model of excitation generation: https://ccrma.stanford.edu/~jos/SMAC03S/Liljencrantz_Fant_Derivative_Glottal_Wave.html or http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.643.5278&rep=rep1&type=pdf). Besides added naturalness, the use of such models provides control in terms of F0 but also *phonation style*.

In case of linear prediction, the excitation signal can also be a residual signal obtained by filtering the original speech with an LP-based vocal tract filter estimated from the same signal window. However, this only applies to analysis-synthesis cases where the original signal is known, and is not applicable to, e.g., speech synthesis where the signal is generated from text. In case of speech coding, the residual is not actually transmitted and used in detail, but its key characteristics are quantified, quantized, and transmitted to the synthesis side.

A fourth option is to use actual pre-recorded glottal excitation signals as the waveform. These signals are obtained from the process of *glottal inverse filtering* (GIF), which is a technique for removing (or separating) the effects of vocal tract and lip radiation from a recorded speech signal. This is then an approximate reconstruction on the actual phonation style of a real speaker. Since the excitation in the recording has a natural F0 value specific to that speaker and recording situation, adjustment of the F0 can be achieved by up-/down-sampling the excitation signal. Since the original GIF-based excitation signal has a finite length (as it is difficult to sustain phonation at a fixed F0 for long periods of time), length of the excitation signal can be extended by concatenating additional periods of the excitation to the end of the actual excitation. However, a naïve approach to concatenation will make the excitation sound unnatural for the extended sections, as the concatenation will introduce artificial exact repetitions and potential discontinuities/jitter to the periods that human hearing will pick up. In this exercise, the glottal waveform has been artificially extended in time to support much higher F0s for voiced segments than what the original one would enable.