Speech Signal Processing
– Exercise 4 –
Vocoder and Quantization

Timo Gerkmann, Kristina Tesch, Danilo Oliveira

1 Introduction

In the upcoming 3 exercise sessions, we will build our own LPC-vocoder! Firstly, the analysis stage is implemented, which extracts all parameters necessary for a decent description of a speech signal. Secondly, the parameters are send to a synthesis stage that reconstructs the speech signal based only on the extracted parameters. Several experiments will be carried out to investigate the importance of each of the parameters to the overall performance of the LPC-vocoder. Finally, the problem of efficient data transmission is tackled in terms of quantization of the parameters. Before starting with the actual exercise, download the archive `Exercise4.zip` from `STiNE`.

2 LPC-Vocoder: Analysis

2.1 Segmentation

To start, load the speech signal `female8khz.wav` in a `numpy` array and store its sampling frequency. In the sequel, we will use the variable names x for the speech signal and f_s for the sampling frequency. Before we start to analyze the speech signal, we first split the signal into overlapping segments with a length of N samples. Each segment overlaps with the previous frame by $L = N - R$ samples, like depicted in Fig. 1. Use a segment length of 32 ms and a segment shift of 8 ms. You may use the function `m_frames, v_time_frame = my_windowing(x, fs, N, R)` from the first exercise session.

- Why do we segment the signal prior to analysis instead of processing the whole signal at once?
- Is a segment length of 32 ms appropriate? Why or why not?

2.2 Signal power

In the following, we want to estimate a set of parameters for each of the segments. We begin with creating a function named `compute_power` that takes a signal segment as an input variable and returns the signal power,

$$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^{N} x(n)^2, \tag{1}$$

where n is the sample index and $x(n)$ corresponds to the current signal segment. Compute the signal power for each segment and store it in a vector. Display the standard deviation ($\sqrt{\sigma_x^2} = \sigma_x$) together with the waveform of the speech signal in one plot.

2.3 Voiced / unvoiced decision

Write another function, `is_voiced`, to also analyze if a segment is voiced or not. This problem is more complicated than you might expect and lots of methods were proposed over the years - with varying success. Here, we use a simple

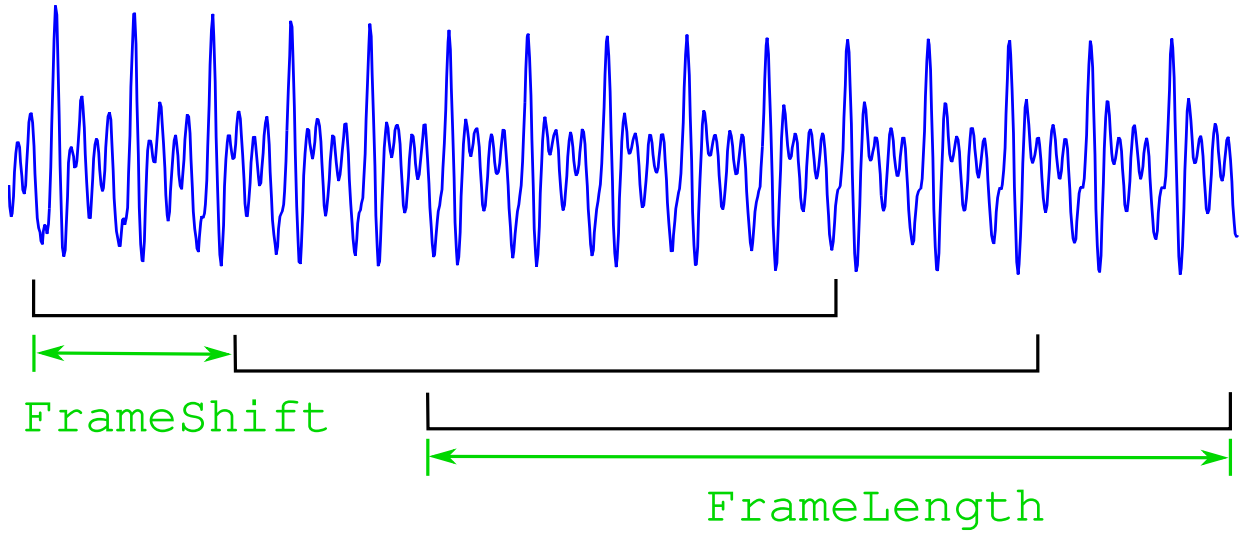

Fig. 1: Symbolic segmentation of a speech signal into segments of length N, overlapping by $L = N - R$ samples.

approach, which is counting the number of zero crossings within in one segment.

- Explain differences in the creation of speech for voiced and unvoiced sounds.
- Why might the number of zero crossings provide valuable information for the voiced/unvoiced decision?
- Think of a way how you can efficiently detect a zero crossing between two samples. How can you realize that in Python?
- Count the number of zero crossings in each segment and normalize them by the segment length N. Apply a threshold to decide if the segment is voiced or not. Try to find an appropriate value for the threshold that allows for a reliable classification. Do not expect it to be 100% perfect, but try your best.
- `is_voiced` should return 1 for voiced segments and 0 for unvoiced segments.
- Plot the voiced/unvoiced decision together with the waveform of the speech signal in one plot. Scale the V/UV decision if necessary. This visualization can be very helpful when checking your implementation or threshold.
- In general: are all speech sounds either voiced or unvoiced? Can you think of other speech sounds? Examples?

Compute the voiced/unvoiced decision for all segments (`np.apply_along_axis` might be helpful) and store it in a vector.

2.4 Fundamental frequency estimation

In voiced sounds, the vocal cords periodically open and close. The frequency at which they open and close is called the fundamental frequency, f_0.

You have already implemented a method to estimate the fundamental frequency based on the autocorrelation function (ACF) in the first exercise session. We can now reuse it here. Put your estimator into a Python function of the form

```
def estimate_f0(m_frames: np.ndarray, fs: int) -> [np.ndarray (v_f0_estimates)]
```

where `v_f0_estimates` is a vector containing the fundamental frequency estimates in [Hz] for each signal segment.

- Plot your fundamental frequency estimate together with the spectrogram of the clean speech. For this, the y-axis needs to be in Hz. Use the function `compute_stft` from exercise 2 to compute the spectrogram.

2.5 Linear prediction coefficients / linear predictive coding

In the lecture, you learned that the form of the vocal tract is crucial to speech production and that it can be modeled by concatenating sections of lossless tubes. This model can well be described in terms of reflection coefficients for each section border. A closely related description can be obtained in terms of an all-pole filter, i.e. its filter coefficients, known as the linear prediction coefficients (LPCs). A common, since very efficient, method to compute the LPCs is the so-called Levinson-Durbin algorithm. Although not necessary at this point, it is worth mentioning that LPCs can be transformed into reflection coefficients and vice versa. This will come in handy in Section 4.

- Shortly outline the Source-Filter-Model for speech production.
- Compute the LPCs of the signal segments based on the code you have from exercise session three. For that, write a Python function `def compute_lpc(m_frames: np.ndarray, M: int)->[np.ndarray (m_lpc)]` that returns the LP coefficients of each signal segment in the matrix `m_lpc`, which is of size `[num_frames x M]`.
- Choose a suitable model order M for linear prediction for a signal with an audio bandwidth of 4 kHz. Give reasons for your choice.

Now we have all parameters we need to decently describe our speech signal. In the next section, we will try to reconstruct the actual waveform of the speech signal based only on this parameter set.

3 LPC-vocoder: synthesis

In this section, lets assume that the speech signal is recorded and analyzed on one device, the transmitter, and only the parameters, but not the signal itself, are send to another device, the receiver. Now we want to reconstruct the speech signal only from the provided parameters with the help of a LPC-vocoder. We will proceed step-by-step, incorporating one parameter a time, analyzing the influence of each of the parameters to the overall result.

3.1 LPCs and constant excitation signal

First, we want to utilize only the LPCs, modeling the influence of the vocal tract. For this, we filter a stationary excitation signal $e(n)$ with the LPC coefficients.

- Create two excitation signals
  1. unvoiced: white Gaussian noise via `np.random.randn`
  2. voiced: pulse train with a frequency of 100 Hz.
     [Tip: `voiced_ex = np.zeros(num_samples)` and `voiced_ex[::fund_period_in_samples] = 1`]
- Segment the excitation signal $e(n)$ into non-overlapping frames of length R (the same segmentation will be used implicitly in all upcoming experiments)
- Filter each segment with the corresponding LPCs using the function `filter_adaptively` provided by the package downloaded from `STiNE`. It allows for improved adaptive filtering by introducing a filter state storing necessary information of the last filtering. It can be used as

  ```
  segment_out, filter_state = filter_adaptively(np.array([1]), LPCs, segment,
      filter_state_in).
  ```

  Note that `filter_state` from the last segment is used as an input parameter. It is then modified by `filter_adaptively` and the new `filter_state` is returned.
- Put together the separate segments to obtain the whole signal.
- Listen to both of the signals, voiced only and unvoiced only. Can you understand what is said?
- Display the spectrogram of the two synthesized signals.

3.2 Voiced/unvoiced excitation

Now we want to go a step further and switch between the voiced and unvoiced excitation signal based on the voiced/unvoiced parameter extracted from the speech signal in Section 2. Use the same excitation signals as in the previous experiment, but use the pulse train for voiced sounds and the noise for unvoiced sounds. Don't forget to still apply the linear prediction filter. Listen to the resulting signal. If you encounter perceivable errors in the voiced/unvoiced decision, try to adapt the threshold in Section 2.3, reanalyze the speech signal and try the new parameter set.

3.3 Amplitude modulation

As the next step, we also incorporate the signal power into our signal synthesis.

- Modify the amplitude of each signal segment by means of a real-valued gain g so that it has the same power as the corresponding input speech signal segment in Section 2.
- How do you have to choose the gain g to achieve this?
- Listen to the synthesized signal. What is the improvement achieved by applying g?
- Display the spectrogram of the synthesized signal, compare it to the spectrogram of the last section and explain differences.

3.4 Variable f_0

We already modeled the vocal tract, incorporated the signal power, and switched between voiced and unvoiced excitation signals. However, during voiced sounds the fundamental frequency is still speaker independent and constant. Therefore, we now incorporate the fundamental frequency estimates obtained from the speech signal.

- For unvoiced sounds, just keep on using white Gaussian noise.
- For voiced segments, things become a bit more complicated to implement. Instead of the constant pitch signal, we now have to adapt the distance between two pulses based on the fundamental frequency estimate. Further we have to take care that we avoid inconsistencies between adjacent segments due to changes in the fundamental frequency.
  - Create a counter that you increase for every sample of a voiced segment.
  - Test if the counter is larger than the current fundamental period in samples.
  - Every time this is the case, set the counter to zero and insert a pulse into the excitation signal at the corresponding sample of the segment.
  - Keep the value of the counter also if the segment ends, do not reset it to zero.
- As before, chose the excitation signal based on your voiced/unvoiced decision and apply the linear prediction filter and gain g.
- Listen to the resulting signal. Does the signal quality improve?
- Now that we incorporated all of our parameters, do you still hear artifacts or errors in the synthesized speech signal? If this is the case, try to find and explain reasons for that.
- Display the spectrogram of the synthesized signal, compare it to the one of the last section, and explain differences.

A block diagram of the final synthesis stage is depicted in Fig. 2.

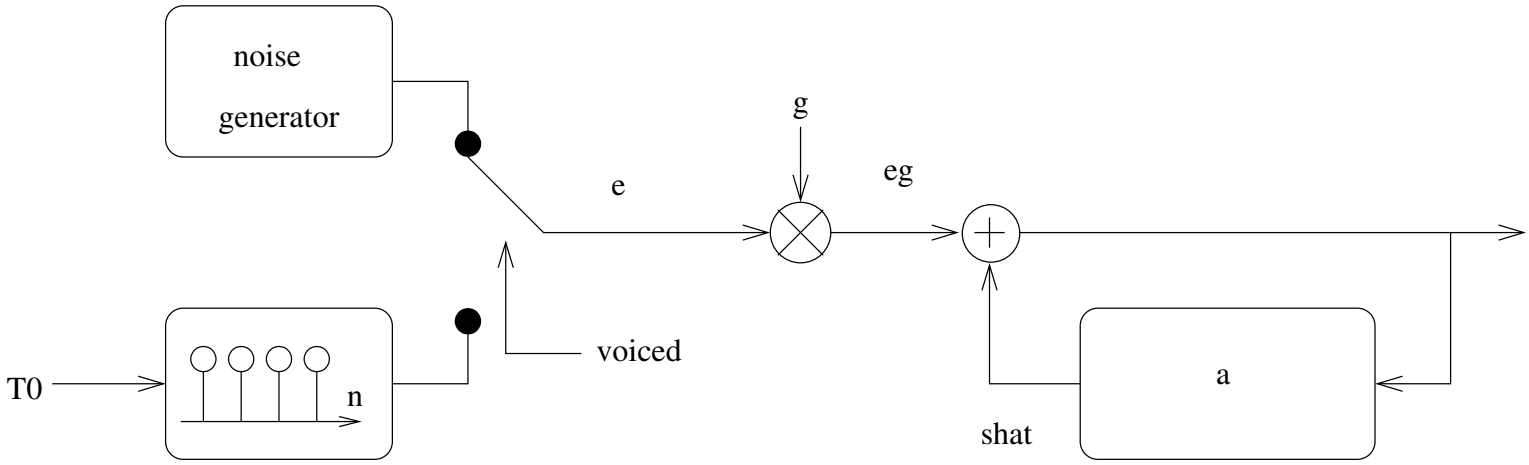

Fig. 2: Synthesis stage of a LPC-vocoder with sample index n, excitation signal $e(n)$, gain g, fundamental period T_0, LPCs $\hat{\mathbf{a}}$, and synthesized signal $x(n)$.

3.5 No LPC

To investigate the importance of the LPCs, in the following experiment we simply leave out the LPC synthesize stage while all other steps are applied as before.

- Intuitively, how does leaving out the LPC synthesize step influence the spectrum of the signal?
- Display the spectrogram of the synthesized signal, compare it to the one of the last section, and explain differences. Do your observations confirm your considerations made before?
- Can you understand what is said?
- Judging from your listening experience, how important is the vocal tract for speech intelligibility?
- After all these experiments, can you judge which of the parameters are most critical for speech intelligibility? Quality? Speaker recognition? You are allowed (and encouraged) to perform some more experiments here to make your decision.

4 Quantization

In this section, we will investigate the problem of how to quantize the vocoder parameters efficiently to reduce the data load that has to be transmitted from the analysis stage (Section 2) to the synthesis stage (Section 3).

4.1 Implementation of a uniform scalar quantizer

Before actually quantizing the different parameters of the vocoder, we implement two rather general Python functions for uniform quantization of scalar values. Firstly, we implement the encoder stage, mapping an input signal onto quantization indices.

- Create the function

```
def quantize_encoder(x: np.ndarray, num_bits: int, x_range: float, x_center: float)-> [
    np.ndarray (quant_idxs)]
```

  which maps the input signal vector `x` onto indices `quant_idxs` corresponding to discrete signal levels ranging from `(x_center-x_range/2)` to `(x_center+x_range/2)`. The number of levels is given by the number of bits `num_bits` used.

- How many quantization levels (or indices) can you realize with n Bits?
- Lets assume you have an input signal with values ranging from -2 to 2 and you choose `x_range = 4`, and `num_bits = 2`. How should you choose `x_center`? What are the expected quantization levels? The step size between two quantization levels? Use these findings to check your implementation.
- Hint: The encoding stage of a uniform scalar quantizer can be implemented very efficiently in terms of only a single division and flooring per signal sample, no for loops or comparisons are needed!

Secondly, after transmitting the indices to the receiver, we also need a decoder function

```
def quantize_decoder(quant_idxs: np.ndarray, num_bits: int, x_range: float, x_center: float)
    ->[np.ndarray (quantized_x)]
```

that synthesizes the quantized output signal `quantized_x` based on the indices `quant_idxs`.

- Create a test signal for testing your implementation. Use a ramp function (-5:0.01:5).
- Set `x_range = 6`, `x_center = 0`, and `num_bits = 2`. First encode then decode the signal and compare the input signal to the quantized output signal by displaying both in one plot.
- Does your implementation behave as expected? What happens if you change `x_center` to 1? Discuss the advantage/disadvantage for signals with many values around 0.
- As for the encoder, also the decoder can be implemented without any loops or comparisons!

Now that we have a uniform quantizer available, we can start quantizing the parameters of the vocoder.

4.2 Quantizing the fundamental frequency

The fundamental frequency of the speech should be quantized with as few bits as possible without a loss of signal quality.

- How do you choose `x_range` and `x_center` based on the maximum and minimum fundamental frequency used in the estimation process (Section 2)?
- Try different numbers of bits for quantization. Use the quantized values (obtained after encoding and decoding) for vocoder speech synthesis. How many bits do you need at least to avoid degradation of the signal quality?
- Plot the histogram of the non-quantized and of the quantized fundamental frequency estimates in one plot (`num_bins =500`).

How would you propose to quantize the voiced/unvoiced decision? How many bits do you need?

4.3 Quantizing the signal energy

- Plot the histogram of the signal energy with `num_bins=50`. Find appropriate values for `x_range` and `x_center` based on the histogram.
- Again, like for the fundamental frequency, find the minimum number of bits that is needed to avoid audible signal distortions.
- Now, take the logarithm of the energy values. Plot a histogram and find `x_range` as well as `x_center`.
- How many bits do you need now? What is the reason for taking the logarithm?
- What is the better choice her: linear or logarithmic quantization?
- What happens if the signal energy of another input signal is much larger than for our test signal `female8khz.wav`? Can you come up with a solution to that problem?

4.4 Quantizing the LPCs

Last but not least, we want to quantize and transmit the LPCs.

- Lets assume that we do not want to transmit the LPCs directly, but the corresponding reflection coefficients r. What would be the advantage of that?
- However, typically neither the LPCs nor the reflection coefficients are transmitted. Instead, *log area ratios*

$$LAR_i = \log \frac{A_i}{A_{i+1}} = \log \frac{1 + r_i}{1 - r_i}, \tag{2}$$

  are quantized and transmitted, with A_i being the area of the i^{th} tube segment and r being the reflection coefficient. What is the advantage of quantizing LAR instead of the reflection coefficients? Disadvantage?
- Compute the log area ratios LAR by first transforming the LPCs to reflection coefficients, using `poly2rc`. Then, obtain LAR from r either via the formula above or via `rc2lar`. On the receiver side, for obtaining the quantized LPCs, the inverse functions `lar2rc` and `rc2poly` can be used.
- As before, find `x_range`, `x_center` and the number of bits needed for quantization results free of audible artifacts. State reasons for your choice.
- Compare the spectrogram of the quantized vocoder output to that of the not-quantized one and of the input speech signal. Explain differences.

4.5 Data rate

Finally, we want to compare the data load that has to be transmitted when using your quantization setup.

- For each parameter, how many bits per frame do you need? Per second?
- What is the overall bit rate in kbit/second? Also show how you came to your result.
- What is the bit rate for a direct quantization of the waveform with 16 bit per sample?
- What is the price to pay for the reduced bit rate?