Speech Signal Processing
— Exercise 1 —
Time Domain Speech Analysis

Timo Gerkmann, Kristina Tesch

This exercise session is about identifying the fundamental frequency of a speech signal using the time domain representation. The approach addressed in this exercise requires the signal to be split into overlapping blocks. Speech signals are often analyzed in such short time frames and therefore, you will write a Python function which performs this step. This function will also be useful in the following exercise sessions.

For solving this exercise, basic knowledge about Python is required. Before you start solving the exercises, you need to download and extract *Exercise1.zip* from *moodle* which contains the wave files needed for this exercise. If applicable, the exercises should be performed for both signals.

1 Fundamental frequency estimation by eye

a) Load the wave files from the archive (e.g. using `load` from `librosa.core` and setting `sr=None`).
   - What is the sampling frequency of the signals?

b) Plot the signal as a function of time [s]. For this, create a vector which contains the time instants for each sample. For plotting, you can employ the `plot` command from `matplotlib.pyplot`
   - Identify the voiced, unvoiced and silence regions in the waveform. Which criteria did you use to distinguish between the three signal types?

c) Pick a single voiced segment from both signals and estimate the fundamental frequency from the waveform. The results measured this way may be helpful to verify the outcomes of the fundamental frequency estimator in part 3 of the exercise.
   - Plot your selected segments and describe your procedure. Judging based on the measured fundamental frequencies, do the signals originate rather from a male or a female speaker? Verify your findings by listening to the signals! (In the following exercises you should be able to listen to audio data contained in `numpy` (abbreviated with `np`) arrays. For this you could use the `play` function from `sounddevice`.)

2 Block processing

Write your own Python function that splits the time domain signal into overlapping frames. A prototype for this function could look like the following example assuming that the loaded signal is stored in the vector `v_signal`.

```
def my_windowing(v_signal: np.ndarray, sampling_rate: int, frame_length: int,
    frame_shift: int) -> [np.ndarray (m_frames), np.ndarray (v_time_frame)]
```

The parameter `sampling_rate` is the sampling rate in Hz, while `frame_length` and `frame_shift` should contain the frame length and the frame shift in units of milliseconds. The extracted segments are stored in the rows of the matrix `m_frames`. The elements of the vector `v_time_frame` correspond to the time instants around which the frames are centered.

- In how many frames can the input signal be split? Try to find a formula for computing the number of frames from the signal length, frame length and frame shift.

3 Fundamental frequency estimator

In this part of the exercise a fundamental frequency estimator will be implemented which operates on the overlapping frames from the previous exercise. The estimator is based on the autocorrelation function (ACF) which will be computed on every frame. This function indicates how similar a block is to a shifted version of itself. This way the time period between the glottal impulses can be identified which allows for estimating the fundamental frequency. In order to give you an idea on how the ACF will look like in a voiced segment, an example is shown in Figure 1.

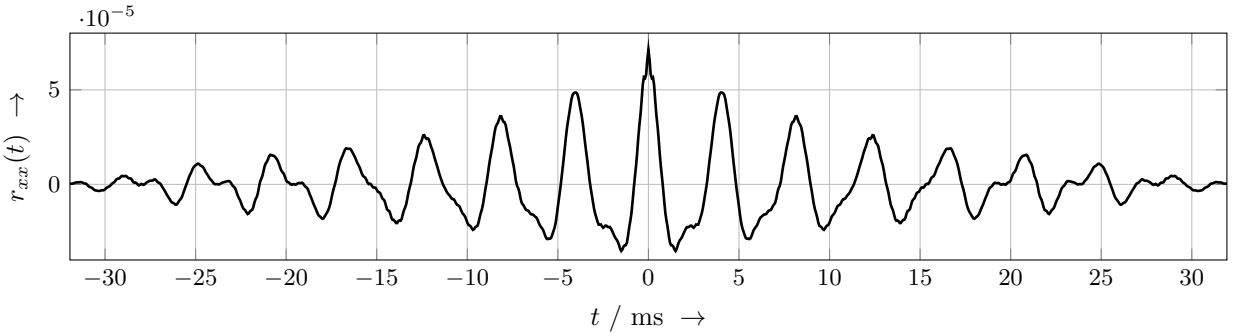

Figure 1: Example of the ACF function for a voiced signal frame (With: t: time, r_{xx}: ACF function)

Equation (1) gives the definition of the ACF $r_{xx}[\tau]$ for an input frame $x[n]$.

$$r_{xx}[\tau] = \frac{1}{N} \sum_{n=0}^{N-|\tau|-1} x[n]x[n-\tau] \tag{1}$$

N and τ denote the length of a frame and the so called lag of the ACF, respectively. As equation (2) shows, the ACF can be described by the convolution of the input frame and its time reversed version.

$$r_{xx} = \frac{1}{N} x[n] * x[-n] \tag{2}$$

a) Split the signal into 32 ms frames with a frame shift of 16 ms using your function from Section 2.

b) Compute the ACF for every frame using `np.convolve` which computes the convolution. Think about how the indices of the resulting ACF correspond to the lag τ.

c) Remove the lower half of the ACF which corresponds to the negative lags.

d) Estimate the fundamental frequency by selecting the maximum in the remaining part of the ACF. The search range should be limited to frequencies between 80 Hz and 400 Hz, so the maximum at $\tau = 0$ ms will not be identified as period length. Remember that the detected maximum corresponds to the period length!

e) Plot the time course of the estimated fundamental frequencies together with the time domain signal.

- In which parts of the signal does the fundamental frequency estimator give reasonable results and why? Do the estimated frequencies match your findings from the first exercise in Section 1?