Speech Signal Processing
— Exercise 2 —
Frequency Domain Speech Analysis

Timo Gerkmann, Kristina Tesch

In this exercise session we focus on frequency domain signal analysis. Since we are interested in speech signals that vary greatly with time, you will implement the short-time Fourier transform (STFT) for frequency domain analysis and visualize it in a spectrogram.

Download the file *Exercise2.zip* from *moodle* which contains the signals *speech1.wav* and *phone.wav*.

1 Short-time Fourier transform

Write a Python function which computes the short-time Fourier transform (STFT) with the following form:

```
def compute_stft(v_signal: np.ndarray, fs: int, frame_length: int, frame_shift: int,
    v_analysis_window: np.ndarray) -> [np.ndarray (m_stft), np.ndarray (v_freq), np.
    ndarray (v_time)]
```

The input and output parameters are defined as follows:

`v_signal`	vector containing the time domain signal
`fs`	sampling rate in Hz
`frame_length`	frame length in milliseconds
`frame_shift`	frame shift in milliseconds
`v_analysis_window`	vector containing that contains the spectral analysis window (This vector should have the same length as the frames, i.e., `frame_length` in samples.)
`m_stft`	a matrix which stores the complex short-time spectra in each row
`v_freq`	a vector which contains the frequency axis (in units of Hertz) corresponding to the computed spectra
`v_time`	time steps around which a frame is centered (as in previous exercise)

The function should perform the following steps.

1. Split the time domain signal into overlapping blocks using the function `my_windowing` that you implemented in the previous exercise.
2. Apply the analysis window to each segment of the time domain signal.
3. Use the `fft` function provided by `np.fft` to compute the DFT for each windowed segment.
4. Only keep the lower half of the spectrum and remove the upper half. Make sure that the frequency bin at the Nyquist frequency is still included.
    - Why are the computed spectra complex conjugate symmetric?
    - What may be the advantage of only considering one half of the spectrum?
    - How can you compute the frequency for each spectral bin? How many sampling points does the spectrum have after you removed the mirrored part while including the Nyquist frequency bin?

You can use `np.fft.rfft` and `np.testing.assert_array_almost_equal` to ensure that your implementation works correctly.

5. Store the transformed frames in the rows of the output matrix `m_stft`.

2 Spectral analysis

If not stated otherwise, the following exercises should be performed for both signals.

a) Use your own function to compute the STFT and plot the logarithmic magnitude spectrogram in dB using the following parameters.

- *frame length*: 32 ms
- *frame shift*: 8 ms
- *window function*: periodic Hann window

You can create the Hann window with `v_analysis_window = get_window('hann', frame_length_samples, periodic=True)` using `scipy.signal.get_window` where `frame_length_samples` is the frame length in samples. The spectrogram can be plotted using the `matplotlib.pyplot.imshow` function:

```
fig = plt.figure()
ax = fig.add_subplot(111)
im = ax.imshow(10*np.log10(np.maximum(np.square(np.abs(m_stft.T)), 10**(-15))),
    cmap='viridis', origin='lower', extent=[v_time[0], v_time[-1], v_freq[0],
    v_freq[-1]], aspect='auto')
fig.colorbar(im, orientation="vertical", pad=0.2)
```

The extent option tells `matplotlib` to use the entries of the vector `v_time` for the x-axis and `v_freq` for the y-axis. Here, the vector `v_time` contains the time instants for each block / each spectrum and the vector `v_freq` contains the frequency bin information.

- Why is the magnitude plotted in dB? Why is it reasonable to introduce a lower limit? What is the lower limit in the command given above in dB?

b) Identify the voiced, unvoiced and silence segments in the spectrogram of the speech signal by eye.

- Describe their appearance and what distinguishes them. Is it possible to identify the different voicing types more easily in comparison to the time domain representation?

c) Produce the same plot as in a) but this time using a frame length corresponding to 8 ms and a frame shift of 2 ms. Further, create a plot for a frame length of 128 ms and a frame shift of 32 ms.

- How well can you distinguish single sinusoidal components? Short impulses? Explain the influence of the different parameter settings.

d) Only for the speech signal estimate the fundamental frequency using the auto-correlation-based method of the last exercise session. Plot the estimated fundamental frequency onto the spectrogram. The parameter setting should be the one used in a). This can be achieved calling `matplotlib.pyplot.plot` on the the same axis instance (variable named `ax` in the code snippet above).

- Do the estimated fundamental frequencies follow the harmonic structures in the spectrogram? You may also want to plot higher harmonics by multiplying your estimated fundamental frequencies with a positive integer value. This way, you can see the precision of the estimated frequencies more precisely.

3 Synthesis from the STFT domain (Inverse STFT)

Use the provided Python function `compute_istft` to synthesize a time-domain signal using overlap-add. The function has the following header:

```
def compute_istft(m_stft: np.ndarray, fs: int, frame_shift: int, v_synthesis_window:
    np.ndarray) -> np.ndarray (v_signal)
```

The input and output parameters have the following meaning:

`m_stft`	matrix containing the STFT spectra which have been generated using the function from Section 1.
`fs`	the sampling rate in Hz
`frame_shift`	frame shift in milliseconds used for the frames in `m_stft`
`v_synthesis_window`	vector containing a synthesis window function
`v_signal`	vector which contains the synthesized time domain signal

Check if the provided function works correctly with the following signal `v_test_signal = np.ones(2048)`. Assume the sampling rate is 16 kHz.

1. Generate the STFT of `v_test_signal` using your function from Section 1 with a frame length of 32 ms and a frame shift of 16 ms. Employ a $\sqrt{\text{Hann}}$-window as analysis window. This window can be obtained by applying `np.sqrt` to the previously generated Hann-window.
2. Resynthesize the signal using the `compute_istft` function. Use the periodic $\sqrt{\text{Hann}}$-window also as synthesis window. For `frame_length` and `frame_shift`, use the same values that you employed for generating the STFT. Finally, plot the synthesized signal.
   - Is it possible to perfectly reconstruct the input signal? Are there parts where a perfect reconstruction is not possible when a $\sqrt{\text{Hann}}$-window is used as analysis and synthesis window?
   - What happens, when you unset the parameter `periodic` in the window generation? Which error can you observe in the reconstructed signal? Explain the difference in the window function which causes this behavior.