Speech Communication

Exercise 2

Voiced and unvoiced segments

The rate of zero-crossings over time can be used to detect voiced and unvoiced segments in the signal. Write a script that uses the result from Exercise 1.2 (download the solution from the ISIS site) to split the signal “13ZZ637A.wav” up into voiced and unvoiced parts. Define two variables, one for unvoiced segments, one for voiced segments. As a simplication, we assume that voiced and unvoiced segments alternate.

Exercise 2.1: Zero-crossing rate, upper and lower threshold

Initialize the following two variables:

```
#Above this threshold the segment is assumed to be unvoiced
voiced_upper_limit = 50

#Below this threshold the segment is assumed to be voiced
unvoiced_lower_limit = 25
```

Please plot a diagram of the zero-crossing rates over time and visualize both thresholds, `unvoiced_lower_limit` and `voiced_upper_limit`, as horizontal lines. Color the upper threshold in green, the lower threshold in blue. You should make use of the defined function `compute_zero_crossings(…)` and `compute_zero_crossings_rate(…)` that were borrowed from exercise 1.

You may use the `Axes.axhline(self, position_value, color="<colour-specifier>")` and `Axes.axvline(self, position_value, color="<colour-specifier>")` methods to add straight lines in your plot. The colour specifier in Matplotlib can be one letter that represents a colour or a colour code.

Exercise 2.2: Find and split the voiced and unvoiced segments

This task will need an iteration over the zero-crossing rates.

Now, the main iteration has two alternating loop bodys (which can be switched with a branch). Each of them can add one index to a common list. The list holds the positions which delimit the segments. It is sufficient to use one `for`-loop.

The first loop body case iterates over a voiced segment and finds the start of the next unvoiced segment. It is found when the zero-crossing rate *first reaches a value above the upper limit*. Then, the most recent previous index is added to the list *where the rate was equal or above the lower limit*. Tip: Python provides a method `list.append(self, value)` to add new elements at the end of a list.

The second loop body case iterates over an unvoiced segment. It finds the end of the unvoiced segment by searching for the *next index whose zero-crossing rate is below the lower limit*. This index is added to the list. It continues with the first loop body, repeating the loop procedure until all zero-crossing rates had been iterated over.

or

1) Decrease your index until you find a value below `unvoiced_lower_limit`. This is the start of the unvoiced segment.

2) Increase the index until you find a value below `unvoiced_lower_limit`. This is the end of the unvoiced segment.

Voiced segments occur in between the unvoiced segments.

Use `numpy.split(array, list_of_indices)` to separate the sound samples at the found indices in your list. The list odd segments are voiced (the first, third, etc.), the list of even segments are unvoiced.

Visualize the segment separation with a red vertical line for each start of a voiced segment and a black vertical line for each start of an unvoiced segment in the diagram.

Exercise 2.3: Create wav-files and play the segments

Finally, combine all voiced segments together into a single array `wav_voiced` and do the same for all unvoiced segments, `wav_unvoiced`, e.g. by using `numpy.concatenate`.

There is the Python module `soundfile` which contains a `soundfile.write(filename, array, sample_rate)` function to save the samples as a `.wav` file. Another module `sounddevice` exists to provide a function `sounddevice.play(sample_array, sample_rate)` to play audio data in a convenient way.

The `sounddevice.play()` function does not wait until one audio file finished playback and therefore will immediately start all audio playback at once. This problem can be solved with `sounddevice.wait()`.