Speech Communication

Exercise 3

Spectrogram Computation

Use the file `exercise_3.py` as template. The function is in this file which is called for exercise 3.1 and the calculation of 3.2 should be inserted here. (Please use the files available on ISIS to write your code.)

Exercise 3.1 Windowing

Please implement the constructor `WindowSequence(samples: numpy.ndarray, sample_rate: float, window_duration_seconds: float, window_step_ratio: float)`.

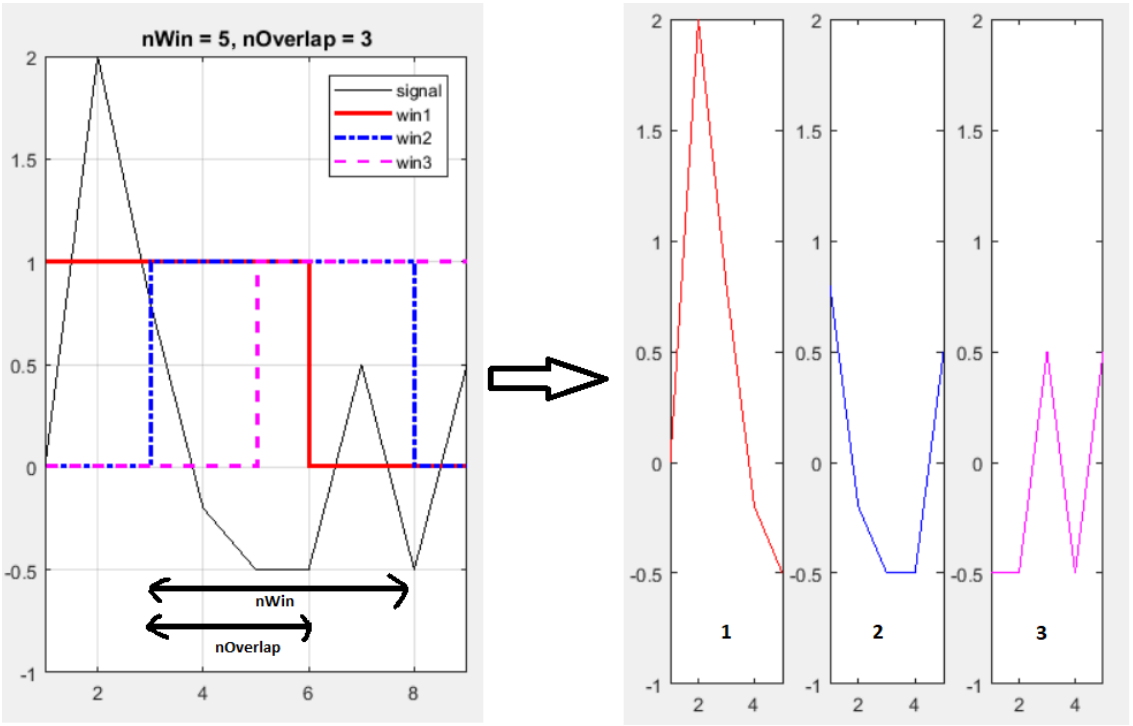
A constructed `WindowSequence` object should provide at least these attributes:

- `self.window_size`, the computed (integer) count of samples per window. Assume that each window sample adds a sampling period to the window duration.
- `self.window_step`, the computed (integer) count of samples between the offsets of two adjacent windowed segments
- `self.window_times`, for each column in `self.window_matrix` it contains the center point of time of the corresponding window's time span
- `self.window_matrix`, a matrix which contains each windowed segment of the original samples array as one column

`window_step_ratio` should specify the amount of the window size that is used for the `self.window_step`. The number of samples in each column of `self.window_matrix` is equal. Therefore, the number of rows corresponds to the size of the window and the number of columns corresponds to the number of windowed segments.

Tip: the `numpy.arange(start_value, stop_value, value_step)` function can be used to compute the offsets of the windows. `numpy.add.outer(column_array, row_array)` allows for repeating a 1D array as a column vector by adding it to each component of a second 1D array that is treated as a row vector.

Of course, you can also choose to create your matrix with your own loop or list comprehensions but this way, with `numpy` functions, it is much simpler, and easier to overview.



x

1x9 double

	1	2	3	4	5	6	7	8	9
1	0	2	0.8000	-0.2000	-0.5000	-0.5000	0.5000	-0.5000	0.5000
2									
3									

xWin

5x3 double

	1	2	3	4
1	0	0.8000	-0.5000	
2	2	-0.2000	-0.5000	
3	0.8000	-0.5000	0.5000	
4	-0.2000	-0.5000	-0.5000	
5	-0.5000	0.5000	0.5000	
6				
7				

nWin = 5
nOverlap = 3

Exercise 3.2 Spectrogram

In this exercise, a spectrogram of the audio samples should be plotted manually by using only a Fourier Transform. The spectogram shows the power density spectrum of multiple frequencies over time which equals the squared magnitude of the Fourier Transform of different windowed segments of a signal, with time-shifted window. Usually, the values are represented as coloured pixels in a 2D image.

Please finish the code by adding following steps:

1. Load the audiofile with `librosa.load(…)` (keep the original sample rate)
2. Define the window duration in seconds and the window step ratio and call the `WindowSegments()` constructor. Save the `WindowSegments` object in the variable `windows`. `window_duration_seconds` should be 0.02 and the `window_step_ratio` should be 0.5 (50%).
3. Transform the computed windowed segments into the frequency domain using the `scipy.fft.fft(matrix, n, axis)` function from Python module `scipy`. The `axis` determines the dimension along which the FFT is applied to the matrix.

   As known from the script, the FFT can be computed fastest for M = 2^m number of samples. We should use an `n` for the size of the FFT which is a power of 2 instead of the actual window size. You can use the `round_to_pow_of_2()` function which has been provided to the code.
4. For the purpose of plotting the spectrogram we need the times for the x-axis and the frequencies for the y-axis. The `window_times` were already computed with the Exercise 3.1. Redefine the time values as `times`. Compute the variable `frequencies` using `numpy.linspace()`. The first frequency is 0, the last frequency should be the first frequency of the 2nd half of the window. This frequency is half the sampling rate.

   Why is the highest frequency $\frac{f_S}{2}$? (For the answer, see the next point.)
5. The FFT also computes the spectrum of negative frequencies. These values are redundant due to symmetry of the FFT and therefore we don't need them. Reduce the number of matrix rows at the end such that they match the number of frequencies from point 4.

   Tip: you can use `numpy.delete()` or a slice of the old matrix.
6. Finally, compute the power spectral density of the spectrum values in the remaining matrix, which is the squared amplitude, in dB. Name it `power_spectral_density_db`.

Tip: 20*log10(abs(y))

You can leave the remaining part of plotting the data into a diagram to the existing code in the exercise sheet.

Exercise 3.3 Spectrogram with Hann window

In the lecture you've learned about the Hann window.

Let `M` be the length of the Window.

Now implement a function `hann_window(window_size: int) -> numpy.ndarray` that generates a Hann window as a column Vector of the shape `(M+1, 1)`, according to following formula from the script:

$$h_T(k) = \alpha + \beta \cdot \cos\left(k\frac{2\pi}{M}\right) \qquad (4.12)$$

Most commonly, one of the following window functions are used:

- *Hamming window*: $\alpha = 0.54, \beta = -0.46$
- *Hann window*: $\alpha = 0.5, \beta = -0.5$
- *Rectangular window*: $\alpha = 1, \beta = 0$

The output should have a 2D shape `(M+1, 1)` as opposed to a 1D shape `(M+1,)` because broadcasting rules do not work for operations between 1D arrays and matrices. With "broadcasting" rules, a column vector is automatically repeated to a matrix for pointwise operations. This way, you can multiply a column vector's component to a corresponding matrix row as if it would be a scalar multiplication.

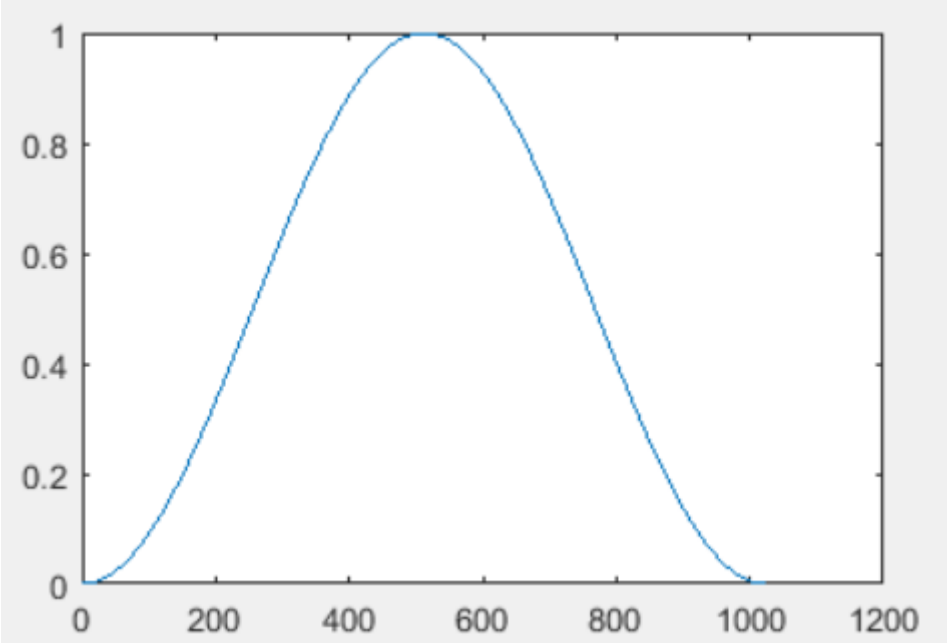


Figure 4: Hann Window Example with M = 1000

Now, repeat Exercise 3.2, but this time apply a Hann Window to each column in `window_matrix` before plotting the spectrogram. (Note: you can reuse and modify the WindowedSignal object for this purpose.)

Compare the spectrogram with and without Hann window. What do you notice?

Play around with the script and try different window sizes and step ratios. What changes in the spectrogram do you notice?