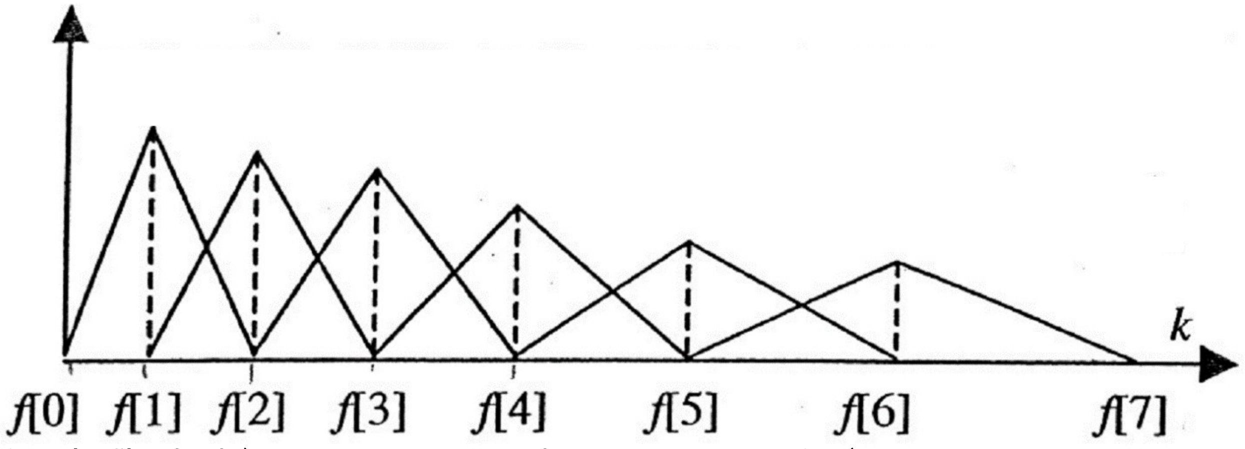
Speaker Recognition:

Computing a mel-scaled spectrum for speaker recognition

The spectrum you get via an FFT has a linear scale on the frequency axis. In applications in human computer interaction, e.g. speech or speaker recognition, however you often use a scaling of the frequencies that is similar to the human hearing abilities.
This can be realized with a mel-filter bank (mel: melodic tone height scale). For this purpose, you need to compute a short time spectrum and a triangular weighting function (see Figure 1), where the number of triangles k corresponds to the number of auditory critical bands (“Frequenzgruppen”). Within each of these frequency bands you sum up the spectral components weighted by a triangular filter.



Triangular filter bank (Huang, Acero, Hon: "Spoken Language Processing").

In the following you will have to compute the frequency support points (f in figure 1).
The following two requirements have to be fulfilled:

- The frequency support points have to be almost equally spaced in the mel-frequency domain.
- You have to find the corresponding support points in the linear frequency domain.

The template pa_02_main.m already consists of code for:

- Reading the wav-file
- Computing and plotting a spectrogram
- Plotting of mel-filtered spectrogram
- Speaker recognition prediction model

Use this script as basis to call the functions you will be implementing in E1-E4. The last part of the template is a speaker recognition model that uses 3 speech files to train the recognition model and then identifies the speakers of 3 test files. If you calculated the mel-filtered spectrogram correctly, the model will identify who the speaker of each test file was.

E1 Computation of equally spaced points on the mel-scale

Please implement the following function

```
function edges = melfreqs(fmin, fmax, k)
```

which should compute `k+2` points in the linear frequency domain that are equally spaced on the mel-scale (f in the figure above). `fmin` and `fmax` are the lowest and highest frequency in the signal (you can find them in the vector `freqs`). `k` corresponds to the number of auditory critical bands, which is k=24.

Use the following equation to compute the mel-value of a linear frequency:

$$Mel(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right).$$

Steps:

1. Transform `fmax` and `fmin` value into mel-frequency domain
2. Compute a vector of equally spaced values in the mel-frequency domain ranging from the mel-`fmin` to the mel-`fmax`.
3. Transform the values into the linear frequency domain.

Tip: To transform a mel-frequency into the linear frequency domain you will have to solve the equation from above for f.

Helpful functions: `log10, linspace`

E2 Computation of the mids of the triangular filters

Please implement the following function

```
function mids = computeMids(freqs, edges)
```

which computes the k mid-frequencies of the triangular filters.

The output of the function from E1 gives you the frequency support points that are equally distant in the mel domain but do not necessarily exist in the vector `freqs`. The function `computeMids` has to find the closest frequency support point in the linear frequency domain. Hence, you will have to compare the frequency support points in the vector `freqs` with the edges of the triangular filter in the vector `edges`.

Inside your function you will first have to define the output vector `mids` which has the same dimensions as the vector `edges`.
Please set the following values `mids(1)=1` and `mids(1,end)=edges(1,end)`.

E3 Mel-filter bank

You already computed the mid values of the triangular filter (`mids`). Now use this vector to compute the increasing and decreasing parts of the triangle $H_t(freqs)$ via the following equation.
For t = 2 to k+1, with `t` being the index of the vector `mids`:

$$H_t(freqs) = \begin{cases} 0 & \text{for: } freqs < mids(t-1) \\ \dfrac{2\cdot(freqs - mids(t-1))}{(mids(t+1) - mids(t-1))\cdot(mids(t) - mids(t-1))} & \text{for: } mids(t-1) \le freqs < mids(t) \\ \dfrac{2\cdot(mids(t+1) - freqs)}{(mids(t+1) - mids(t-1))\cdot(mids(t+1) - mids(t))} & \text{for: } mids(t) \le freqs \le mids(t+1) \\ 0 & \text{for: } freqs > mids(t+1) \end{cases}$$

Please implement the following function

```
function H = computeFilter(mids, freqs)
```

which uses the equation from above to compute the triangular filters. Plot all k triangular filters over the frequency into one figure and add labels to both axis. k equals the number of auditory critical bands of the human auditory system (24).
(After the computation of the triangular filters the vector $H_t(freqs)$ contains k+2 filters. With the first and the last being dummy values. Delete those rows before you plot the filter.)

E4 Mel-filtered spectrum
Please implement the following function

```
function melSpec = melFilter(spec, H)
```

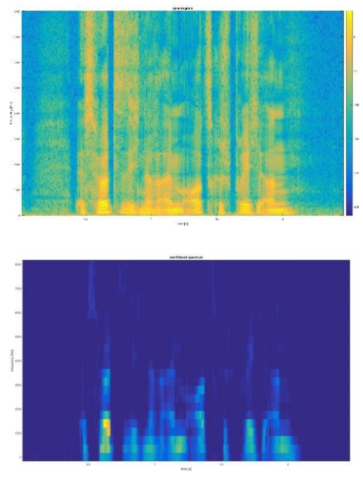
which uses the triangular filter H from exercise E3 and the linear spectrum of the signal `spec` to compute the mel-filtered spectrogram.

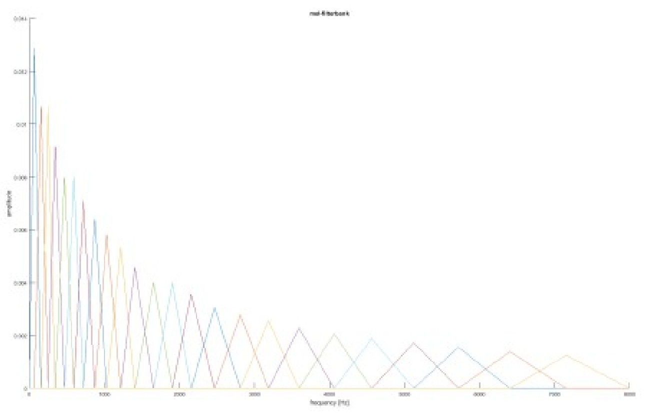
To this end, you have to compute a matrix multiplication between the triangular filter H and the absolute values of the spectrum of the signal `spec`.

<u>Remember:</u> To perform a matrix multiplication of the matrices A and B, the number of columns of A has to equal the number of rows of B. In our case the matrices should have the following dimensions:

- H: 24 x 257
- spec: 257 x 176

As a hint, the plotted results should look similar to this (note I used a different audio file):





Speaker Recognition
The second part of the script is a speaker recognition model. Firstly, the recognition model is trained with 3 speech files, from 3 different speakers. Then the model identifies the speakers of 3 unknown files (with the same speakers as before) automatically. The model uses ceptrum coefficients that are calculated from the mel-filtered spectrogram and then applies them to a GMM (Gaussian Mixture Models) classifier. You don't need to worry about how the recognition model works in detail. If your functions from E1-E4 calculate the mel-filtered spectrogram correctly, the model will identify the correct speakers. If it fails however, you probably made a mistake somewhere. You may listen to the files to check whether you can distinguish the different speakers : `sound(x1,fs)`, `sound(x1_test,fs)`, `sound(x2,fs)`, …