Matlab practice

Today we will see how to create a spectrogram (and some other things) in Matlab-bal. Using a python environment would have also been possible, I will give some tips for this at the end of the lecture.

1. Accessing Matlab (legally):

Matlab is available for free for all the teachers and student of our university. You will neEd an EDUID for this. You can get it here (with a neptun ID): <https://www.eduid.u-szeged.hu/reg/>

The access to Matlab is described here (sorry, in Hungarian):  
<https://u-szeged.hu/download.php?docID=105051>

We can choose between downloading and installing Matlab, or using the online version.

I will use the latter. In this case, you can access it here for the first time: <https://www.mathworks.com/academia/tah-portal/university-of-szeged-31495897.htmlA>

And after registrataion, use matlab.mathworks.com

In this lecture, we will use the and old package called Voicebox. You can ownlowad it from here:   
<http://www.inf.u-szeged.hu/~tothl/speech/voicebox.zip>

Download and unzip it, then start Mablab. If you use the isntalled version, set the working directory to the folder where you unzipped the files. If you use the online version, you have to upload the files to the Matlab drive.

1. Reading in a sound file, and displaying the spectrogram

Reading the sound file:

[wav,FS,WMODE,FIDX]=readwav('SA1.WAV');

Input: SA1.WAV: a sound file

Output: Y: the samples

FS: sampling rate

WMODE, FIDX: further parameters from the sound file header (we won’t need them)

Drawing the spectrogram:

spgrambw(wav, FS); (using the default parameters)

spgrambw(wav, FS, ‘’); where some possible parameters in the ‘’:

i: inverted grayscale

j: color scale

m: logarithmic (mel) frequency scale

w: displays also the wave form

Of course, you may also modify some other parameters (eg. window size and step size)

3. Spectrogram with a resolution and frequency scale adjusted to human hearing

If we calculate th espedtrogram the following way, we obtain the spectral data in B, its size on the time and frequency axes in T and F:

[T,F,B]=spgrambw(wav, FS, ‘’);

For this file, we get 1760 spectral vectors, and the height (spectral resolution) will be 257.

From the linear frequencz scale we can switch to a mel-scale using the *m* switch. However, we will still have 257 spectral values. We discussed in the lecture that for speech recognition there is no need for such a high resolution image, fewer samples (eg. 20-40 values) would be enough. We could downsample the frequencz axis by keeping only every 4-5the value, but then we would totally lose the information in between. Thus, we will fuse neighboring frequency channel by applying a sort of weighted averaging. We do it on the spectrogram with linear frequency axis, so to get the results on the mel-scale we will take fewer samples at higher frequencies, using wider and wider fusing. With these weighting curves we (very roughly) simulate the tuning curves of the hair cells.

So, let’s first create a standard spectrogram:

[T,F,B]=spgrambw(wav, FS, '');

First we display only the weights (with only 20 bands for visibility), then we save the weights into an array (in this case, creating 80 bands):

melbankm(20,512,FS);

m=melbankm(80,512,FS);

If you check the size of *m*, you see that it contains 80 weight vectors with 257 components in each vectror, which fits the height of our spectrogram

We can apply the weights to the spectrogram by a matrix multiplication:

melspectrogram=m\*B';

Let’s display the results (on a logarithmic amplitude scale – for spgramw it is the default)

imagesc(log(melspec));

The image is upside down, so we need this command: set(gca,'YDir','normal')

Remark: there is a much easier way to get the same result: we can apply the built-in matlab function called melSpectrogram (but I wanted to show how the weighting works):

melSpectrogram(wav, FS,'NumBands',80)

In practice, this melSpectrogram can be used as the input for a speech recognizer – this is very popular in neural network-based systems. Usually the number of frequency channels is set to   
26-40, but more recently a larger resolution of 80 channels is also polular.

4. Speech analysis using a (gammatone) filter bank:

Creating a spectrogram and then fusing frequency bands by triangular weighting is just a quick approximation of the “filters” in our ear. A more professional (but slower) is to creat a special filter bank, and then to filter the original speech signal using this filter bank. One such well-designed filter bank is the so-called gammatone filterbank:

First we have to calculate the filter parameters (we cover the frequency axis from 100Hz 8000Hz using 1 ERB wide filter at a a step size of 0.35 ERB):

[b,a,fx,bx,gd]=gammabank(0.35, FS, '', [100 8000]);

The size of the a and b parameter arrays show that we obtained 83 filters.

Pefrorming the filtering and displaying the result:

filterbank(b,a,wav,gd);

5. Creating a Mel-spectrogram using HTK:

From the next practice, we start to use the HTK speech recognizer. HTK has its own modul for feature extraction, for example, we can extract the melspec parameters from SA1.waw:

HCopy.exe –C preprocess.config SA1.wav SA1.htk

Where the config parameters are set so that we will obtain the rsults of a mel filter bank consisting of 100 filters (using the triangular weighting presented earlier).

melspec=readhtk\_lite('SA1.htk');

We can display the result using the toolbox (if you use the online Matlab, don’t forget to upload SA1.htk-t to the drive first…):

imagesc(melspec');

set(gca,'YDir','normal')

6. 2 methods to extract and display the fundamental frequency (pitch) curve:

fxrapt(wav,FS);

fxpefac(wav,FS,'','g',''); or fxpefac(wav,FS,'','G','');

7. Example for formant tracking:

Mustafa-Bruce formant tracker (not part of Voicebox):

mb\_ftracker(wav, FS);

Similar signal processing modules for python:

librosa (for python), pyAudioAnalysis, SurfBoard