ZRE 01 - Introductory Lab

Jan Černocký, Valentina Hubeika , FIT BUT Brno

The goal of the lab is to introduce basic operations with a sound signal, mainly using Matlab. This lab (as well as all following labs) will be run under Linux. Although Matlab is available under Windows OS as well, some of the tools (`ws`, `sox`) we are going to use are present only in Linux.

Useful Practice

It is highly recommended to make notes at each lab. In a new terminal, open your favorite text editor (Emacs, vim, kate, ...) and do all the writing into a **text file**. Then copy&pasta all the Matlab (or Shell) commands from the text file into the interpret. Following this approach, you will always have the complete sequence of the work you've done during the lab (so you can check the previous labs when needed).

1 Sound Capturing and its Visualization - WaveSurfer

For experimenting with sound in this course we will be using the easiest possible format: 1 channel, the sampling frequency of 8 kHz, linear quantization on 16 bit. We can either record the sound or download it and convert to the desired format. For both purposes we will be using WaveSurfer: `ws`[1].

1.1 Capturing from a Microphone

It can be useful to first play with `alsamixer` (sound-card mixer for ALSA sound-card driver). Alsamixer is operated using the arrow keys and the key 'M' (mute). The microphone input is denoted as 'Mic' or 'Capture' (tone them up). In some cases you will need to switch on the microphone boost (MicBoost) - it must not be 'M'.

In the preferences of the `ws` (WaveSurfer), set the sampling frequency (in our case 8000 Hz) and quantization resolution (16 bits). Open a new sound file and do the recording. Replay the sound (to check everything went good). Save it in the format WAV (with a file header containing the information on the sampling frequency and quantization) and RAW (with no header).

If you do not dispose of a microphone, you can use an example file `test.l16` (RAW) and `test.wav`.

1.2 Acquiring Sound by Converting from Different Formats

There is a lot of sound data available on the Internet (for instance in PodCast). To convert the sound into the desired format, we will use WaveSurfer. For example:

```
ws ESLPod451.mp3
```

- Select the first 11 seconds, COPY.
- Open a new sound file, PASTE.
- Change the sampling frequency, the number of channels and the resolution in the menu `Transform -> Convert`.
- Save in the format WAV or RAW.

2 Matlab - Essentials

- We will be working with Matlab (later with C) under Linux.
- Run Matlab from a terminal: `matlab`. If you want to run a console version of Matlab that uses no the Java machine:
  `matlab -nojvm` or
  `matlab -nodesktop`
- You can paste the commands only in case the cursor is on the command line »

[1] available as well for Windows from KTH in Copenhagen: `http://www.speech.kth.se/wavesurfer/`

- If the line contains `%`, Matlab interprets the right-hand side of the symbol as a *comment*, therefore does not interpret it.
- Access to the command history is available through ↑ (eventually, ↓). Matlab also provided a selective access, for example, you can select between already used `plot` commands using:
  `pl` ↑ ↑ ↑ ...
- Matlab contains a high quality on-line help:
  `help 'the function you are interested in'`

If you are new with Matlab, see the first lab of the Signal and Systems course:
`http://www.fit.vutbr.cz/study/courses/ISS/public/`
sections 1–3.

3 Loading a Speech File in Matlab

Loading a speech file in Matlab is very simple. The samples are stored in Matlab as a row vector. The RAW files (1 sample ∼ 1 short of 16 bits) are read in a similar way as in C (note the vector-like behavior of the function `fread`).

```
ff=fopen('test.l16','r');
sig=fread(ff, [1 inf],'short')/32768;
fclose(ff)
plot(sig);
```

Note, we normalize the samples to the interval ±1. Using the RAW format, we have to know the format of samples in advance (RAW contains no header).

On the contrary, the WAV file contains all the necessary information in the header, which can be easily accessed:

```
sw=wavread('test.wav'); sw=sw';
[sw,Fs,Nbits] = wavread('test.wav'); sw=sw';
```

The alternative call (second line) returns the information from the header (Fs,Nbits) additionally to the sample sequences (sw). Note, that the data matrix has to be consequently transposed (waveread has a column vector as the output). There is no normalization (to the range ±1) of the signal needed, as the waveread function does it on its own.

Tasks

1. Look at the loaded signals: `plot(sig); plot(sw);`. Do they look the same ?
2. Check, whether the WAV header contained the required values.
3. Check, whether the signals are de facto the same: `ws - sig` or `plot(ws - wsig);`
4. Replay the signals:
   `sound(ws)`
   `sound(50*ws)`
   `soundsc(50*ws)`
   Why, when using the second command, the sound is disturbed? Why not in the case of the third command? In which cases do we use function `sound` and when `soundsc`?

4 Mean Value and Mean Value Normalization

Estimation of the mean value and consequent mean value normalization (subtraction of the direct component, dc-offset) is a relevant operation in speech processing. Off-line calculation (when we have the complete signal) is done according to the following formula:

```
mean(sw)
% mean normalization
sc = sw - mean(sw);
% check
mean(sc)
```

When processing the signal on-line (new values keep coming in), the n-th sample estimation of the mean value is :

$$m[n] = k\, m[n-1] + (1-k)s[n],$$

where k is a constant close to 1.

```
k = 0.99;
N = length(sw);
sconline = zeros(1,N);
mntolook = zeros(1,N);
mn_1=0;
for ii=1:N,         % ... don't use 'i' as a cycle variable
                    % complex unit :-(
  mn = k * mn_1 + (1-k) * sw(ii);
  sconline(ii) = sw(ii) - mn;
  mntolook(ii) = mn;
  mn_1 = mn;
end
```

Matlab cycles are very slow and are not optimal. In some case, such as this, there is no other way of computing.

Tasks

1. Compare the original and the centralized signals: `plot (1:N,sw,1:N,sconline)`
2. Look at the course of the estimated mean value over time: `plot (1:N,mntolook);`
3. ... and compare it to the offline estimation: `mean(sconline)`
4. Why do the values of the mean value calculated on-line oscillate at the regular (off-line calculated) mean value? How would you smother the oscillation? What disadvantage would it have?

5 Selection and Storing of a Signal

Selecting of a part of a signal can be done by indicating boundary indices: `x = sw(1000:2000);`

Storing in the WAV format: `wavwrite(x,8000,16,'vyrez.wav');`

Storing in the RAW format:

```
ff=fopen('vyrez.l16','w');
fwrite(ff, round(x * 32768), 'short');
fclose(ff)
```

Note, 'de-normalization' is required, in order to convert the data into type `short`'s.

Tasks

1. Select a voiced segment, listen to it using `sound(x)`. Repeat the procedure with the unvoiced segment and the transitional segment (a plosive, 'p'). Can you determine the sound (phoneme) from a short segment?
2. Save the selected segment in the WAV and RAW format and load it in the WaveSurfer. Check if everything looks correct.

6 Segmenting a Signal into Frames

Due to the quasi-stationary behavior nature of the speech signal, we cannot really process the signal as the whole. Therefore, we are forced to divide the signal into smaller segments, frames. The typical length of the frame is 20 ms with the 10 ms overlap of the (neighboring) segments. Frame length in samples:

```
l = 20e-3; r = 10e-3;
ls = round (l * Fs)
rs = round (r * Fs)
```

Cutting the signals into frames that are then stored in the matrix columns `swram`:

```
% frame shift
fs = ls - rs;
%number of frames - without deduction - more during the lecture
Nram = 1 + floor((length(sw) - ls) / fs)
% allocation for the frames - will be in columns
swram = zeros(ls, Nram);
% and filling:
odtud=1; potud=ls;
for ii=1:Nram,
  ramec = sw(odtud:potud)';
  swram(:,ii) = ramec;
  odtud = odtud + fs;
  potud = potud + fs;
end
```

Tasks

1. Loot at a voiced and an unvoiced frame (for instance, the 13-th an the 100-th frames of the test.l16 file, respectively):
   `plot (swram (:,13));`
   `plot (swram (:,100))`
2. Listen to the selected frames — is 20 ms enough to recognize the sound (phoneme) ?
3. Wrap the segmentation into a function — we will be using it later in the lab. Store the function into your `$HOME/matlab`. This directory is implicitly contained in the Matlab path.

7 Spectrogram

Spectrogram is a useful tool for the speech analysis. No theory so far, just to compare the original signal and its spectrogram:

```
subplot(211);
plot(sw); axis tight;
subplot(212);
specgram(sw,256,Fs);
```

Tasks

1. Why do we use the axes alignment `axis tight` ?
2. Which sounds correspond to the high/low energy at the higher/lower frequencies?

8 Signal Generating, Filtering, Frequency Analysis

In this part of the lab, we will be working with a generated (artificial) signal. We will generate a mixture of the two harmonic signals ($f_1 = 440$ Hz and $f_2 = 1$ kHz) with the sampling frequency $F_s = 8000$ Hz. The normalized frequency is :

$$f' = \frac{f}{F_s}$$

. Cosine is generated using:

$$x[n] = C\cos(2\pi f' n)$$

the duration of the generated signal will be 2 seconds.

```
% signal generation - mixture of 2 cosines, duration of 2 sec.
Fs = 8000; f1 = 440; f2=1000;
n = 0:15999;
s = 0.49*cos(2 * pi * f1/Fs * n) + 0.49*cos(2 * pi * f2/Fs * n);
plot (s); soundsc(s)
```

For the frequency analysis, we will be using first 1024 samples of the generated signal. Note, we will be looking only at the frequency interval of $[0, F_s/2]$. (The other half is symmetric.)

```
% frequency analysis - select 1024 samples
x = s(1:1024);
f = (0:511) / 1024 * Fs;
X = fft(x); X = X(1:512);
subplot (211); plot(f,abs(X)); subplot (212); plot(f,angle(X));
```

8.1 Filtering

Here, the task is to extract the component on the frequency 1 kHz. In Matlab, there is an operating function `ellip` (and the auxiliary function `ellipord`) for defining the IIR filters. Check the function help. The filter is a band-pass filter. Note: whilst we usually normalize the signal relative to the sampling frequency, in Matlab, we do normalization relative to the Nyquist frequency ($F_s/2$) !

```
% desight a filter to select 1 kHz:
Fs2 = Fs / 2;  % this is our normalization frequency
Wp = [900/Fs2 1100/Fs2];
Ws = [800/Fs2 1200/Fs2];
Rp = 3; Rs = 20;
[N, Wn] = ellipord (Wp, Ws, Rp, Rs)
[B,A] = ellip(N,Rp,Rs,Wn)
```

We shall look at the frequency characteristics and the location of the zeros and poles of the proposed filter:

```
% frequency characteristics
freqz(B,A,512,8000);
% poles and zeros
zplane (B,A)
```

. . . now filter the input signal:

```
% filtering of the signal
sf = filter (B,A,s);
plot (sf); soundsc(sf)
```

Tasks:

1. Replay the output signal and do the frequency analysis (again, using 1024 samples). Did we manage to discard the 440 Hz frequency?
2. You may see some residuals on the 440 Hz of the spectrum.

9 Frequency Analysis of Speech

Record a speech segment on $F_s = 8000$ Hz. Do the mean normalization (store the normalized segment into the matrix `sw`) and segment it into frames of 160 samples each without the overlap. (store it into the matrix `sr`). If your segmentation function does not work, use the function `frame.m`.

```
sw=wavread('test.wav'); sw=sw';
sc = sw - mean (sw);
sr = frame (sc, 160, 0);
```

Select a voiced frame:

```
x = sr(:,10);
```

We will do the frequency analysis according to the lectures:

1. on the whole bandwidth, from 0 to F_s.
2. on the half of the bandwidth, from 0 to $F_s/2$.
3. using zero-padding for the resolution smoothing in frequency domain.
4. power spectra density (PSD).
5. logarithm of the PSD.

```
% frequency analysis + power spectral density.
f = (0:159) / 160 * Fs; X = fft(x);
subplot (211); plot(f,abs(X)); subplot (212); plot(f,angle(X));

% half of the period
f = (0:79) / 160 * Fs; X = fft(x); X = X(1:80);
subplot (211); plot(f,abs(X)); subplot (212); plot(f,angle(X));

% zero padding
f = (0:511) / 1024 * Fs;
X = fft([x' zeros(1,1024-160)]); X = X(1:512);
subplot (211); plot(f,abs(X)); subplot (212); plot(f,angle(X));

% power spectral density + zero padding
f = (0:511) / 1024 * Fs
X = fft([x' zeros(1,1024-160)]); X = X(1:512); Gdft= 1/160 *abs(X) .^ 2;
subplot (111); plot(f,Gdft);

% logarithm of the power spectral density + zero padding v dB:
Gdftlog = 10 * log10 (Gdft);
subplot (111); plot(f,Gdftlog);
```

During lectures, you will find out that the spectrum of speech is defined by the fundamental frequency (fine structure) and the articulation tract configuration (rough structure). Can you see this on the calculated spectrum?

10 Hand-Made Spectrogram

In the section 7 we run the function `specgram` as a 'black box'. The function in fact calculates the power spectral density for each frame – see `help specgram`. We can calculate such a spectrogram 'by hand':

```
Nr = size(sr,2); % number of frames
% allocation of the matrix containing the spectrogram(single spectra will be contained in columns)
sg = zeros (512,Nr);
for n = 1:Nr,
  x = sr(:,n); %plot(x); pause;
  X = fft([x' zeros(1,1024-160)]); X = X(1:512); Gdft= 1/160 *abs(X) .^ 2;
  sg(:,n) = Gdft';
end
imagesc (10 * log10(sg)); axis xy;
```

Tasks

1. The last line plots the image – find out why we have to use function `axis`.
2. Calculate the spectrogram using the in-built Matlab function and depict it in the alternative figure:
   `figure(2); specgram(sm)`
   Determine, whether they differ. If so, adjust other parameters of the function `specgram` so the pictures agree.

The parameters of the function `specgram` are the following:
`B = specgram(A,NFFT,Fs,WINDOW,NOVERLAP)`

Solution: `figure(2); specgram(sc,1024,8000,boxcar(160),0)`

3. Modify the parameters of the spectrogram so
   - we can clearly see the fundamental frequency folds;
   - we have a good resolution on the time axes;
   - can we have both at a time?
4. Generate a black-and-white spectrogram. Try to use the following:

```
c = colormap ('gray');
colormap (flipud(c))
brighten(magicka_konstanta)
```

. . . or use the function `contrast`.