L1. Audio signals handling using Matlab functions

1. Audio signals

Audio signals generally refer to signals that can be perceived by humans. Audio signals usually come from a sound source that vibrates in the audio frequency range (20Hz-20kHz). The vibrations set the air in motion to form “pressure waves” that travel at approximately 340 m/s. The ears can receive these pressure signals and send them to the brain for recognition.

There are many ways to classify audio signals. If we consider the source of audio signals, we can classify them into two categories:

• *Sounds produced by living beings:* human voices, dog barking, cat meowing, frog croaking, etc. In particular, bioacoustics is an interdisciplinary science that investigates the production of sounds by living things and their reception by them.
• *Sounds from non-living things:* Sounds from car engines, thunder, slamming doors, musical instruments, etc.
If we consider the way audio signals are repeated, they can be classified into two categories:
• *Quasi-periodic sounds:* the waveforms are almost periodic, so that we can detect the repetition period (pitch). Examples of such sounds include the monophonic playback of most musical instruments (such as piano, violin, guitar, etc.) and speech in certain areas or human singing.
• *Aperiodic sounds:* waveforms are not made up of obvious repeating patterns (periodic shapes), so we cannot perceive a stable repetition frequency. Examples of such sounds include thunder, clapping, unvoiced parts of human speech, etc.
In principle, we can classify each short segment of speech (also known as a frame, with a length of about 20 ms) into two types:
• *Sound segment:* These are produced by the periodic vibration of the vocal cords, so the fundamental periods can be observed in a frame. Moreover, due to the existence of the fundamental period, its value can be estimated.
• *Unvoiced segment:* These are not produced by the vibration of the vocal cords but by the rapid flow of air expelled through the vocal tract. Because these sounds are produced by a noise, such as rapid airflow, the fundamental period cannot be observed and no stable frequency can be detected.
Audio signals actually represent the variation of air pressure as a function of time, which is continuous in both time and amplitude. When acquiring signals for storage in a computer, there are several parameters that must be taken into account:
1. **Sampling rate:**This is the number of sample points per second, in units of Hertz (Hz). A higher sample rate indicates better sound quality, but the storage space required is also greater. The commonly used sampling rates are as follows:
- 8 kHz: voice quality for phones and toys;
- 16 KHz: Commonly used for speech recognition;
- 44.1 KHz: CD quality;
2. **resolution**: The number of bits used to represent each sample of the audio signal. Commonly used resolutions are:
- 8-bit: The corresponding range is 0 - 255 or + 127... -128 or
- 16-bit: The corresponding range is -32768 - +32767.

In other words, each sampled point is represented by an 8- or 16-bit integer. However, in MATLAB, all audio signals are converted to floating point in the range [-1, 1], for more efficient manipulation. If the original integer values are desired, the floating point values need to be multiplied by 2nbits-1 , where nbits is the resolution in bits.

3. **No. of Channels**: Mono for single channel and stereo for 2 stereo channels.

2. Basic acoustic characteristics of the voice signal

When analyzing audio signals, the short-term analysis method is usually adopted, because audio signals are more or less stable over a short period of time, about 10-30 ms. When analyzing frame by frame, there may be overlaps between neighboring frames to capture the fine change in audio signals. It is important to remember that each frame is the basic unit for the analysis to be done. In each frame, the following acoustic characteristics can be observed/defined.

• Volume: This characteristic represents the intensity of the audio signal, which is correlated with the amplitude of the signals. Sometimes it is also referred to as the energy or intensity of the audio signals.

• Pitch: This characteristic (height) represents the repetition frequency of audio signals, which can be represented by the fundamental frequency (FF or F0), corresponding to the fundamental period of the audio sound signals.

• Stamp: is a multidimensional attribute that depends on several physical variables. Often timbre is defined in a purely negative manner as "everything that is not intensity, pitch, or spatial perception". Timbre is defined as "that attribute of auditory sensation by which a listener can judge that two sounds presented similarly and having the same intensity and pitch are different". There are, first, the frequency content and spectral profile of the sound. Because the human ear has limited frequency resolution, the spectral composition vector can often be reduced to a vector representing the amount of instantaneous acoustic energy in each critical band (Plomp 1970), without much loss of perceptual information. Finally, the temporal envelope of an instrumental sound, including the attack, decay, and modulation of the stable portion, influences the perceived timbre to such an extent that it can make the sound of an instrument unrecognizable (Berger 1964).

The basic procedure for extracting acoustic features can be as follows:

1. Frame division is done, so that a sequence of audio signal is divided into a set of frames. The time duration for each frame is about 10-30 ms. If the frame duration is too long, we cannot capture the fast time-varying features of the audio signal. On the other hand, if the frame duration is too short, then we cannot extract valid acoustic features. In general, a frame should contain several fundamental periods of the audio signal. Usually, the frame size (in number of samples) is equal to powers of 2 (such as 256, 512, 1024, etc.), so it is suitable for fast Fourier transform.
2. If it is desired to reduce the differences between neighboring frames, overlapping is allowed. Typically, the overlap is usually between 1/2 and 2/3 of the original frame. The larger the overlap, the greater the number of calculations.
3. Assuming that the audio signals in a frame are quasi-stationary, we can extract various parameters or features such as the number of zero crossings, volume, pitch, MFCC, LPC, etc.

This paper aims to provide a brief overview of the MATLAB functions that can be used to capture, store, or play audio signals, as well as basic methods for processing audio signals. The Matlab functions described will be accompanied by examples.

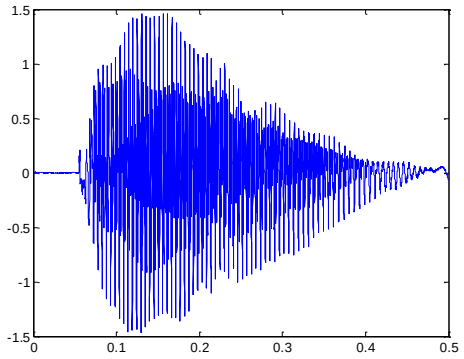
3. Audio signal processing using Matlab functions

This chapter introduces some of the most important functions within the Matlab environment, functions that are used for audio signal processing. Thus, the following functions will be presented for:

1. Reading and displaying audio files;
2. Playback and display of audio signals;
3. Recording from the microphone;
4. Saving .wav files.

3.1 Reading .wav files

In MATLAB, you can read .wav files using the "audioread" command. The following example reads the file "boy.wav" and displays its waveform.



Waveform of the utterance "boy"

```
[y, fs]=audioread('boy.wav');
sound(y, fs);            % play audio signal
time=(1:length(y))/fs;   % X-axis time vector
plot(time, y);           % display waveshape in time
```

In the above example, "fs" is the sampling frequency, which is 16000 in this case. This indicates that there are 16000 samples per second when the sound was recorded. The "y" vector is a column vector that contains the samples of the voice signal. We can use the command "sound(y, fs)" to play the audio signal read from the file. "time" is a time vector, where each element corresponds to the time of each sample. Therefore, we can plot "y" vs. "t" i.e. y(t), to visualize the direct waveform.

Most audio signals are digitized with a resolution of 8 or 16 bits. If we want to know the bit resolution of the input .wav file, we can use additional output parameters to the "audioread" function to obtain information, such as:

```
[y, fs, nbits]=audioread('girl.wav');
```

TABLE 1: Audio formats available to be read by the Matlab audioread command

Audio File Format	Description	File extension
WAVE	Raw audio	.wav
OGG	OGG vorbis	.ogg
FLAC	Lossless audio compression	.flac
AU	Raw audio	.au
AIFF	Raw audio	.aiff,.aif
AIFC	Raw audio	.aifc
MP3	MPEG1 Layer 3, lossy compressed audio	.mp3
MPEG4 AAC	MPEG4, lossy compressed audio	.m4a, .mp4

Moreover, if we want to know the duration of an audio file, we can directly use the "length(y)/fs" function. The following example can obtain the most important information from a .wav file.

>> information = audioinfo('girl.wav');

```
>> information = audioinfo('girl.wav');
>> information

information =

             Filename: 'C:\Users\eugen\Documents\MATLAB\Colea\girl.wav'
    CompressionMethod: 'Uncompressed'
          NumChannels: 1
           SampleRate: 16000
         TotalSamples: 8000
             Duration: 0.5000
                Title: []
              Comment: []
               Artist: []
        BitsPerSample: 16
```

From the above example waveform, it can be seen that the audio signal is normalized, with the amplitude having values between -1 and 1. However, each sample is represented by an 8/16-bit integer. How are they related to each other? First of all, we need to know the following convention:

a) If a .wav file has 8-bit resolution, then each sample is stored as an unsigned integer between 0 and 255 (28-1).
b) If a .wav file has a resolution of 16 bits, then each sampled point is stored as a signed integer between -32768 (215) and 32767 (215-1).

Since almost all variables in MATLAB have the data type of "double", therefore all samples are converted to a floating point number between [-1 and 1), for easy manipulation/processing. Therefore, to retrieve the original integer values of the audio signals, one can proceed as follows.

a) For 8-bit resolution, multiply "y" (the value obtained by wavread) by 128 and then add 128.
b) For 16-bit resolution, multiply "y" (the value obtained by wavread) by 32768.

You can also use the "wavread" command to read a stereo .wav file. The returned variable will be a 2-column array, each containing the audio signals from a single channel.

3.2 Audio signal playback

Once we can read the audio file into MATLAB, we can start processing the audio signals by changing their intensities, or changing their sampling rates, and so on. After the audio signals are processed, they need to be played back for audio inspection. The basic command for playing audio signals is "sound or play". The following example can load an audio data stream from the file "handel.mat" and immediately play the loaded audio signal.

```
load handel.mat    % load signal stored in handel.mat
p = audioplayer(y, Fs); % creates an audioplayer object for signal Y, using sampling rate Fs.
play(y, Fs);    % play audio signal;
```

```
[y, fs]=audioread('do.wav');
sound(y,fs);    % Play the signal with the original amplitude
                % without the Fs parameter plays at 8kHz
```

Since the playback volume is determined by the amplitude of the audio signals, we can change the amplitude to change the volume, as follows.

```
sound(3*y, fs); % Play the amplified signal 3 times
```

In the example above, although we have an increase in amplitude by a factor of 3, the intensity perceived by the human ear is not by a factor of 3. This serves to illustrate that the perception of loudness is not linearly proportional to amplitude. In fact, it is proportional to the logarithm of the amplitude.

When the input signals are outside this range (too high or too low), we can hear the "broken sound". You can try this by running the command "sound(1000*y,fs)" to hear the result.

If we change the sampling rate during playback, it will affect the duration as well as the perceived pitch. In the following example, we will gradually increase the sampling rate so that a short sound with a pitch frequency similar to the sound of the Disney cartoon character Donald Duck is heard.

```
[y, fs]=audioread('church.wav');
p=audioplayer(y,0.5*fs)
play(p); % play sound at half speed
p=audioplayer(y,2*fs)
play(y, 2.0*fs); % play sound at 2x speed
```

On the other hand, if we gradually reduce the sampling rate, we will have longer sounds with a lower fundamental frequency, and eventually it will sound like a cow mooing.

```
[y, fs]=audioread('church.wav');
sound(y, 1.0*fs); % play signal with 1:1 speed
sound(y, 0.5*fs); % play signal with 0.5:1 speed
```

If we invert the audio signal by multiplying it by -1, the perception will be exactly the same as the original. This also serves to demonstrate that human perception of the audio signal is not affected by phase. However, in the case of inverting the audio signal on the time axis, then it will sound like an unknown language as in the following example.

```
[y, fs]=audioread('hall.wav');
P=audioplayer(-y,fs) ;
play(y);          % play reversed signal on Y axe (up-down)
p=audioplayer(flipud(y), fs); % play mirrored signal in time(left-right)
                              %("reverse speech")
```

Downsampling an audio file.

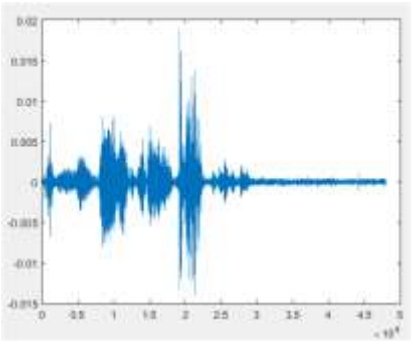
Downsampling by a factor of two is achieved with an audio file by removing every 2nd sample. An illustrative example of implementing downsampling by a factor of 2 is shown below

```
[y,fs]=audioread('hall.wav);
sound(y,Fs);
y2 = downsample(y,2);
sound(y2,Fs);
subplot(2,1,2);plot(y2);
subplot(2,1,1);plot(y);
linkaxes;
```

3.3 Recording sounds from the microphone

This example shows how to record microphone input, play back the recording, and store the recorded audio signal in a numeric array. First, you need to connect a microphone to the system.

In the example below, "duration*fs" is the number of samples to be recorded. The recorded samples are stored in the variable "y", which is a vector of size 48000x1. The data type of "y" is double and the memory space occupied by "y" is 256Kbytes. In this example the number of channels is 1 and the data type of the sampled points is double.



```
fs=16000;                % sampling rate
nbits=16;
nChannels=1;
duration=3;              % recording duration
arObj=audiorecorder(fs, nbits, nChannels);
fprintf('Apasa orice tasta pentru a incepe inregistrarea a %g secunde...',
duration); pause
fprintf('Se inregistreaza...');
recordblocking(arObj, duration);
fprintf('Inregistrare terminata.\n');
fprintf('Apasa orice tasta pentru a reda inregistrarea...'); pause;
fprintf('\n');
play(arObj);
fprintf('Se afiseaza forma de unda a semnalului inregistrat\n');
y=getaudiodata(arObj);   % take sampled audio data
plot(y);                 % display waveshape
```

Press any key to start recording for 3 seconds...Recording...Recording finished.

Press any key to play the recording...

The waveform of the recorded signal is displayed.

3.4 Saving/writing audio files

We can write audio files using the MATLAB command "audiowrite".

The command is: **audiowrite(FILENAME,Y,FS)**
writes Y data to an audio file specified by the filename FILENAME, with a sampling rate of FS Hz. Stereo data must be specified as a two-column array. Multichannel data must be specified as an N-column array.

In this example, we will store the audio data with type "uint8" in the file "test.wav". We will then invoke the appropriate application to play the file. Since the variable "y" for the "wavwrite" command should be of type double in the range [-1, 1], some conversions need to be performed if the recorded data is of other data types, such as "single", "int16", or "uint8". Here is the table for the conversion.

```
fs=11025;                % Sampling frequency
nbits=16;
nChannels=1;
duration=2;              % Recording time
arObj=audiorecorder(fs, nbits, nChannels);
waveFile='test.wav';      % The wav file to be saved
fprintf('Press any key to start recording for %g seconds...', duration); pauses;
fprintf('Recording...');
recordblocking(arObj, duration);
fprintf('Recording completed.\n');
fprintf('Press any key to save the recording to %s...', waveFile); pause;
fprintf('\n');
play(arObj);
fprintf('Displaying the waveform of the recorded signal\n');
y=getaudiodata(arObj);  % Get sampled audio data
plot(y);                % Display waveform
audiowrite(waveFile,y, fs);
fprintf('Finished writing file %s\n', waveFile);
fprintf('Press any key to play the written signal %s...\n', waveFile);
folder(['start', waveFile]);  % Start the application for the .wav file
```

3.5. Volume control in audio files

The loudness of audio signals is the most prominent characteristic for human auditory perception. In general, there are several similar terms that are commonly used to describe the strength of audio signals: volume, intensity, or energy. For the sake of consistency, here we will use the term "volume" to describe intensity. Basically, volume is an acoustic characteristic, which is correlated with the amplitudes of the samples in a frame. To define quantitative volume, we can use two methods to calculate the volume of a given frame:

1. The sum of the samples in absolute value from each frame:

$$volume = \sum_{i=1}^{n} |s_i|$$

where s(i) is the ith sample in a frame, and n is the frame size. This method requires only integer operations and is suitable for low-end systems such as microcontrollers.

2. Or the one given by the formula:

$$volume = 10 * log_{10} \sum_{i=1}^{n} s_i^2$$

This method requires floating-point calculations, but is (more or less) linearly correlated with human perception of the intensity of audio signals. (http://www.phys.unsw.edu.au/~jw/dB.html)

Some of the characteristics of sound intensity are presented below:

• For recording in a quiet place using a unidirectional microphone, the volume of audible sounds is usually higher than that of non-audible sounds, and the volume of non-audible sounds is usually higher than that of ambient noise (however, this is not applicable to recording via an omnidirectional microphone);

• Volume is greatly influenced by microphone settings, especially microphone gain;

• Volume is usually used for voice activity detection.

Before calculatingvolume, an offset removal will usually be performed (by simply subtracting the frame average from each sample) to remove a potential DC.
Formethod 1, we will usually apply median subtraction to eliminate the offset.
Formethod 2, we will apply, subtracting the mean to adjust the zero.

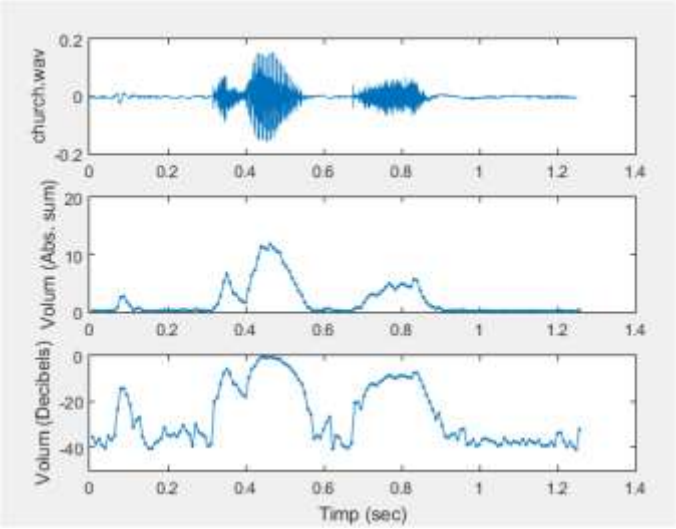
The use of the two methods for calculating volume is illustrated by the following example.

```
waveFile='church.wav';
frameSize=256; overlap=128;
[y, fs]=audioread(waveFile);
fprintf('The duration of %s is %g sec.\n', waveFile, length(y)/fs);
frameMat=buffer(y, frameSize, overlap);
frameNum=size(frameMat, 2);     % Calculate volume with method 1
volume1=zeros(frameNum, 1);
forumi=1:frameNum
    frame=frameMat(:,i);
    frame=frame-median(frame);% zero-justified
    volume1(i)=sum(abs(frame));% method 1
end

volume2=zeros(frameNum, 1);     % Calculate volume with method 2
forumi=1:frameNum frame=frameMat(:,i);
    frame=frame-mean(frame);    % zero-justified
    volume2(i)=10*log10(sum(frame.^2)+realmin);% method 2
end
sampleTime=(1:length(y))/fs;
frameTime=((0:frameNum-1)*(frameSize-overlap)+0.5*frameSize)/fs;
subplot(3,1,1);
plot(sampleTime, y);
ylabel(waveFile);
subplot(3,1,2);
```

```
plot(frameTime, volume1,'.-');
ylabel('Volume (Abs. sum)');
subplot(3,1,3); plot(frameTime, volume2,'.-');
ylabel('Volume (Decibels)'); xlabel('Time (sec)');
```

The two methods of calculating intensity are only an approximation of human perception. However, sound intensity is based on human perception and there could be significant differences between "calculated intensity" and "perceived intensity". In fact, perceived intensity is greatly affected by the frequency as well as the timbre of the audio signals.



Volume calculation using methods 1 and 2 ("church")

4. Progress of the work. Report.

Based on what you previously studied in chapters 1-3, solve the following problems and requirements and prepare a report with results, screenshots, code, etc.

4.1. Get information from a mono audio file. Write a MATLAB script that can read the file "church.wav" and will display the following information in this script:
Number of sampling points. Sampling rate
Resolution Bits Number of channels Recording time (in seconds)

4.2. Wave Recording. Write a MATLAB script to record 10 seconds of speech, such as "My name is X....Y.... and I am a first year master's student in TM at the Communications department of the Technical University of Cluj-Napoca". Save the recording as myVoice.wav. Other recording parameters are: Sampling frequency = 16 kHz, bit resolution = 16 bits. The script will allow you to indicate the answers to the following questions, within the MATLAB window. How much space is occupied by audio data in the MATLAB workspace? What type of data does the audio data have? How do you calculate the amount of memory needed from the recording parameters? What is the size of the myVoice.wav file? How many bytes are used in myVoice.wav to record data other than the audio data itself?

4.3. Audio signal manipulation: Write a MATLAB script to record your utterance of "today is my birthday". Try to explain the playback effect you observe after trying the following operations on audio signals:
Multiply the audio signals by -1. Invert the audio signals on the time axis. Multiply the audio signals by 10. Replace each sample with its square root. Replace each sample with its square. Clip/limit the signal so that samples in the range [-0.5, 0.5] are set to zero. Modify the waveform so that samples in the range [-0.5, 0.5] are set to zero, and samples outside the range of [-0.5, 0.5] are shifted towards zero by the amount of 0.5.

4.4. Sampling rate experiments: Write a MATLAB script to record yourself saying "my name is ***" at a sample rate of 16 kHz and 8-bit resolution, or use an existing file. Try resampling the audio signals by decreasing the sample rates to 11 kHz, 8 kHz, 4 kHz, 2 kHz, 1 kHz, and so on. At what sample rate do we start to have difficulty understanding the content of the utterance?

4.5. Experiments with adding noise: Write a MATLAB script to record the utterance "my name is *** ", with a sampling rate of 8 kHz and 8-bit resolution. We can add noise to audio signals by using the following sequence:

```
k = 0.1;
y2 = y + k*randn(length(y), 1);      % add noise
sound(y2, 8000);                     % Playback sound
plot(y2);
```

Increase the value of k by 0.1 each time and answer the following questions.
a) At what value of K do we start having difficulty understanding the rendered content?
b) Plot the waveforms at different values of k. At what value of k will we start having difficulty identifying the fundamental period?

4.6 Study the detectSpeech function and practice VAD on the file from 4.4.
https://se.mathworks.com/help/audio/ref/detectspeech.html

5. References

https://www.mathworks.com/help/matlab/import_export/record-and-play-audio.html

http://mirlab.org/jang/books/audioSignalProcessing/

http://mirlab.org/jang/matlab/toolbox/sap/

http://mirlab.org/jang/matlab/toolbox/utility/

https://se.mathworks.com/help/audio/ref/detectspeech.html