Digital Audio Coding

Lab Session 1: Wave

Sébastien Boisgérault, Mines ParisTech

Feb. 12, 2015

A Word of Advice

This lab session will probably be much easier if you use the right tools for the task at hand ; getting familiar with these tools is actually worth it as they will also help you in the future lab sessions.

As a general guideline,

- use NumPy for scientific computing with arrays,
- use Matplotlib for data visualisation,

(both librairies are already integrated in the Spyder IDE)

- use `audio.wave` to read/write WAVE files,
- use `audio.io` to play/record sound data,
- use `audio.bitstream` to read/write binary data,
- use `audio.frames` to split/merge audio frames.

Synthesis of Pure Tones

We denote A_4 the analog audio signal defined by

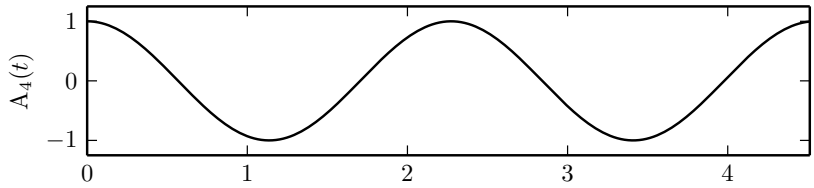
$$\mathrm{A}_4(t) = \cos(2\pi \times 440.0 \times t),\ t \in \mathbb{R}.$$

It's a pure tone with unit amplitude, frequency 440.0 Hz, and no phase.

Let Δf be the Compact Disc Digital Audio (CDDA) sample rate:

$$\Delta f = 44100 \text{ Hz}.$$

1. Assign values to the Python variables `df` and `dt` that we use respectively to denote the CDDA sampling rate in hertz and the CDDA sampling period in seconds.
2. Create the increasing NumPy array `t` of the all times in $[0, 3.0)$ that are multiples of Δt. Define the array `A4` of sampled values of A_4 at these times.
3. Plot two periods of this sampled signal against the time.



time in ms: $t \in \mathbb{Z}/\Delta f \times 1000.0$

4. Save the sampled signal `A4` as a WAVE file named `"A4.wav"` and play it.
5. Automate and generalize: the name "A_4" comes from the scientific pitch notation, a convention that defines the frequency of symbols that are made of a letter followed by a number identifying the pitch octave. The following table displays such frequencies for the letter "A":

Symbol	f (Hz)
A_0	27.5
A_1	55.0
A_2	110.0
A_3	220.0
A_4	440.0
A_5	880.0
...	...
A_9	14080.0
A_{10}	28160.0

Implement a function `make_tone` that given such a `symbol` argument:

- returns the values of a 3-sec. audio sample,
- creates a WAVE file whose name is `symbol + ".wav"`.

Sound Pressure Level and Loundness

1. Create the pure tones `A0` to `A10` and the corresponding WAVE files.
2. Sound power:
    1. Implement a function `SPL` that given a 1-dimensional array `x` returns the *sound pressure level* (SPL) of `x` in decibels:

    $$L\,[\mathrm{dB}] = 96.0 + 10\log_{10}\left\langle \mathtt{x}^2 \right\rangle$$

    where $\langle \mathtt{y} \rangle$ is the mean value of `y`.
    2. Check that the SPL of every tone is (approximately) the same.
    3. What the is analytical formula of this shared value ?
3. Listening tests:
    1. Play the audio files `"A0.wav"` to `"A10.wav"` in sequence.
    2. Is the loudness of the sounds constant across the octaves ? What does it mean ?
    3. Does the frequency of `"A10.wav"` feel very different from the frequency of `"A9.wav"` ? Can you make sense of that ?

WAVE Format Header Analysis

The WAVEform audio file format – or WAVE for brevity – is a Microsoft and IBM digital audio file format. It is a subset of Microsoft's Resource Interchange File Format (RIFF) specification for multimedia file. The WAVE format supports several types of compressions, but we will deal only with the uncompressed format (also referred to as 16-bit linear PCM, for "Pulse Code Modulation").

1. WAVE file names often have a `".wav"` extension, but this is not strictly mandatory: files content can be searched instead for a signature specific to this format.

    Search the web for a description of the WAVE header, identify the WAVE file signature and implement an `is_wave` function that takes a filename argument and returns `True` if the file is a WAVE file and `False` otherwise.

2. Implement a function `wave_info` that given a WAVE filename returns informations about the audio data, namely:

   - the number of channels
   - the data sample rate.

   This function should conform to the following example usage:

```
>>> wave_info("A4.wav")
{"num_channels": 1, "sample_rate": 44100}
```

Quantization and Signal-to-Noise Ratio

1. Load the data of the audio file `"A4.wav"` as an array of floats in the $[-1.0, 1.0]$ range named `A4_wav`.

2. Compute the *quantization error* – or *quantization noise* – `e`, defined as the difference between `A4_wav` and the original array `A4`.

3. Compute the quantization *signal-to-noise ratio* (SNR) in decibels:

$$\text{SNR [dB]} = 10 \log_{10} \text{SNR}^2 \quad \text{where} \quad \text{SNR}^2 = \frac{\langle \texttt{A4}^2 \rangle}{\langle \texttt{e}^2 \rangle}$$

Fading: Frames and Windows

The tones that we have generated so far do not start or end smoothly and the transition from one tone to another is not smooth either. Such fade in, fade out and cross-fading may be implemented with windows.

1. Extend the `make_tone` function so that it takes an optional argument `window`, a window factory : a function, that takes a length argument and returns a window of this length. A *window* is a one-dimensional array that is meant to be multiplied element-wise to the original signal.

   Search the web for the definition of classic windows. Apply for example the "Hanning window" on the pure tone `A4` and listen to the result.

2. Consider the signals `A0` to `A10`. Create a single "blended" (crossfaded) signal that merges windowed versions of these signals in sequence, with a 1.5 sec (50%) overlap between successive signals.

3. What if the successive signals are actually *frames* extracted from a common signal ? When does the merge between the frames reconstruct exactly the original signal ? This condition is referred to as Constant-OverLap-Add (or COLA).

4. Show that the standard Hanning window satisfies only roughly this condition. Define the window (factory) `hanning2` that given a length `n` returns the first `n` values of the `hanning` window of length `n + 1`. Show that this window satifies the COLA condition with a high precision.