Institut Mines-Télécom

Practical work on the conversion of sampling rate and STFT

Roland Badeau

This practical work aims to carry out filtering in a multirate system (with an application to the conversion of sampling rate), and to understand and implement a perfect reconstruction filter bank for audio equalization. The files related to this practical work are available on the eCampus website of the teaching unit.

1 Conversion of sampling rate

We want to achieve the conversion of sampling rate from $Fs = 48kHz$ to $Fs = 32kHz$.

1. Describe and draw the digital processing chain that will permit you to achieve such a conversion (in particular you need to specify the characteristics of the filter $H(z)$ that you will have to use).

2. Synthesize an impulse reponse $h(n)$ appropriate for this conversion of sampling rate. You can use the following implementations of the Remez method:

   - Matlab function `firpm`: `[h,delta] = firpm(odd order,2*[0,nu_c,nu_a,.5],[L L 0 0])`
   - Python function `remez`: `h = scipy.signal.remez(even length,[0,nu_c,nu_a,.5],[L 0])`

   Plot the frequency response $|H(e^{2i\pi\nu})|$ to ensure your synthesis is correct (we expect a difference of at least 50 dB between the pass-band and the stop-band).

3. Program the simplest digital processing chain that will permit you to achieve the conversion. You can use the Matlab function `filter` or the Python function `scipy.signal.lfilter`.

The purpose of the rest of this section is to get an optimal implementation of this conversion of sampling frequency and to compare its performance with that of a direct implementation.
The optimal implementation will be achieved by means of polyphase decompositions (of type I and type II). The use of noble identities and of the equivalence below (Figure 1) will permit you to apply the filtering operations to signals at a lower sampling frequency.

4. Check and exploit the equivalence between the two diagrams of Figure 1.

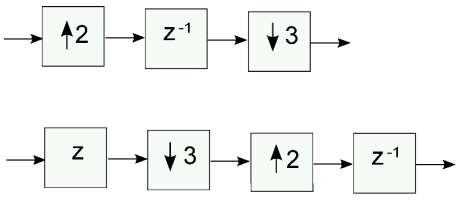

Figure 1: Equivalence

5. Program the efficient implementation of this conversion of sampling rate by using two successive polyphase decompositions.

6. Compare the results obtained with the two implementations and compare the running times of the two approaches (you can use the Matlab functions `tic` and `toc`, or the Python function `time.time()`).

2 STFT audio equalization

2.1 STFT analysis

In Matlab, you will use the program `stft.m`, which provides a framework for the computation of the short time Fourier transform (STFT). In Python, you can use the provided notebook template.
The definition of the STFT that is referred to as "low-pass convention" is given in discrete time by :

$$W_x(\lambda, b) = \sum_{n\in\mathbb{Z}} x(n)w(n-b)e^{-j2\pi\lambda n}, \tag{1}$$

where $w(n)$ is the analysis window in discrete time, which is supposed summable, real and symmetric.

1. We choose $w(n)$ as a Hann (Hanning) window of length N_w. Plot the DFT of w, i.e. $\hat{w}(k/M)$ where $M, M \geq N_w$ is the order of the DFT. What is the width of the main lobe as a function of N_w ?

2. Note that the expression (1), taken at fixed λ, can be written as a convolution and deduce an interpretation of the STFT in terms of filtering. Explain the role of the corresponding filter (low-pass? band-pass? high-pass?). As a linear phase FIR filter, specify its type (type 1,2,3,4?) according to its length (even or odd).

3. Another definition of the STFT, referred to as "band-pass convention", is given by

$$\tilde{X}(\lambda, b) = \sum_{n\in\mathbb{Z}} x(n+b)w(n)e^{-j2\pi\lambda n}. \tag{2}$$

   Explain this latter designation and compute $\tilde{X}$ as a function of W_x. Which one of these two conventions correspond to the computation implemented in `stft.m` or in the notebook template?

From now on, the computation of the STFT will be carried out by means of fast Fourier transforms (FFT), by only considering causal windows of finite length, lower than the order M of the DFT. In order to simplify, the STFT will now be indexed by the index k of the frequency channel located around $\lambda = k/M$ and the temporal index u of the considered frame, i.e.

$$\tilde{X}(k, u) = \sum_{n\in\mathbb{Z}} x(n+uR)w(n)e^{-j2\pi\frac{kn}{M}}. \tag{3}$$

where R represents the temporal shift between successive analysis frames.

4. Based on the Matlab script `stft.m` or on the Python notebook template, for $M = 32$ and $R = 1$, compute the signal $x_k(u) = \tilde{X}(k, u)$, for $k = 3$. Is it real or complex ? Check the interpretation in terms of filtering by using the Matlab functions `filter` and `spectrogram` or the Python functions `scipy.signal.lfilter` and `scipy.signal.spectrogram`. Listen to $\mathrm{Re}(x_k)$.

2.2 Reconstruction

The reconstruction is achieved by an *overlap-add* operation, which will be written as:

$$y(n) = \sum_{u\in\mathbb{Z}} y_s(u, n-uR),$$

where $y_s(u, n) = \mathrm{DFT}^{-1}[\tilde{X}(k, u)](n)\; w_s(n) = \dfrac{1}{M}\sum_{k=0}^{M-1} \tilde{X}(k, u)e^{j2\pi\frac{kn}{M}} \times w_s(n)$

5. Show that a sufficient condition for perfect reconstruction is $f(n) = 1\ \forall n$ where $f(n) = \sum_{u\in\mathbb{Z}} w(n - uR)w_s(n - uR)$. Use the program `ola.m` or function `ola` in the notebook template to check this condition for the product window $w_p(n) = w(n)w_s(n) = h(n)^2$, where $h(n)$ is a Hann window correctly normalized, and for a 75% overlap.

6. In the program `stft.m` or in the notebook template, implement the resynthesis according to the overlap-add approach described in question (e), and check the validity of perfect reconstruction.

2.3 STFT audio equalizer

We exploit the previous results to use the STFT as an equalizer with $M/2 + 1$ real channels (the order M is supposed even). We will then get the synthesis frames as $y_s(u, n) = \mathrm{DFT}^{-1}[\tilde{Y}(k, u)](n)\ w_s(n)$, where $\tilde{Y}$ is obtained by assigning weights to every channel, i.e. $\tilde{Y}(k, u) = w_k\tilde{X}(k, u)$. The perfect reconstruction condition then ensures that $y(n) = x(n)$ when $w_k = 1\ \forall k$.

7. Implement the equalizer and test it.