Processamento Digital de Sinais
Instituto Superior Técnico
Author: Margarida Silveira. Partly based on a lab assignment by João Sanches
Revised: Catarina Barata, André Fonte

Lab Session Planning

Prepare the work before the lab session. The items that you should address are marked below in bold font, e.g. **R1.a)**. These items should be answered as comments in the Python code that you developed. **The code and answers must be submitted in Fénix until the Saturday after the lab session at 23:59**.

Required Python Libraries: `numpy, scipy, matplotlib, pyDub, pyaudio`

1. Sampling and Aliasing

Digital computers are powerful tools for storing and processing signals. However, they operate on discrete-time data, not on continuous-time waveforms. As a result, real-world continuous signals such as speech, music, or biomedical measurements must first be converted into discrete-time sequences through sampling before they can be processed digitally. This conversion can be lossless if the signal is sampled at a sufficiently high rate. According to the **sampling theorem**, a band-limited signal can be perfectly reconstructed from its samples provided that the sampling frequency is greater than twice the highest frequency present in the signal. This critical threshold is known as the **Nyquist rate**. If the sampling frequency is too low, frequency components above the Nyquist limit cannot be represented correctly and instead appear at incorrect lower frequencies. This phenomenon, known as aliasing (or spectral folding), leads to irreversible distortion: high-frequency content is misrepresented as low-frequency components, altering the perceived signal and degrading its quality. To avoid distortion, **anti-aliasing filters** can be used. These filters remove high-frequency components before sampling, thereby preventing spectral folding and information loss.

1. Consider a continuous-time chirp[1] (with t measured in seconds),

$$x_c(t) = \cos\left[2\pi\left(\frac{1}{3}k_1 t^3\right)\right] \tag{1}$$

with instantaneous frequency $\Omega(t) = 2\pi(k_1 t^2)$. In this work, we will use $k_1 = 1000$.

Build a vector `x` by sampling the chirp $x(t)$ at a rate of 8000 samples per second in the interval $[0, 2]$ seconds. Implement the sampled signal $x(n) = x_c(nT)$ yourself; do not use any Python ready-made function for chirp creation. **Note:** Ensure that the type of the vector `x` is `float32`.

R1.a) Listen to the obtained signal (don't forget to specify the frame rate used for sampling). Comment on the relationship between what you heard and the signal that you created in `x`.

R1.b) Compute and display the spectrogram of `x`. Use a Hanning window of size $N = 64$, with an overlap of $3 * (N/4)$ and $4 * N$ as FFT length.

R1.c) What is the relationship between the spectrogram and the sound you heard?

2. Sample the signal $x(n)$ by obtaining the following signal: $y(n) = x(2n)$. Listen to $y(n)$ and observe its spectrogram. Note that the sampling frequency of $y(n)$ differs from that of $x(n)$, so this parameter should be changed in both the command used for listening and the spectrogram creation. In the `spectrogram` command, you should also adjust the window length to the new signal's sampling rate, so that the window duration, measured in seconds, remains the same.

R2.a) Indicate the sampling frequency of signal $y(n)$. Justify your answer.

R2.b) Explain what you have heard and observed.

3. Apply the following **anti-alias** filter to the original signal `x` by means of the following command:

[1] A chirp is a signal whose frequency varies monotonically (increasing or decreasing) with time.

```
from scipy.signal import firwin,lfilter
h = firwin(101,0.5,pass_zero = 'lowpass')
xf = lfilter(h,[1.0],x)
```

These commands generate and filter the signal using an order-100 low-pass FIR filter with a cut-off frequency of $0.5\,\pi$ (expressed in the angular-frequency scale of the discrete-time Fourier transform).

R3.a) Sample the filtered signal `xf` so that the new sampling rate is `Fs/2`. Listen to the result and visualize the spectrogram, using the appropriate sampling frequency. **Note:** Ensure that the type of the vector `xf` is the same as `x`. Explain the differences in what you heard and observed, relative to what you heard and observed in item 2.

2. Filtering in the frequency domain

4. The goal of this exercise is to use the Fourier transform to remove noise from a signal consisting of two sinusoids corrupted by additive Gaussian white noise, and study how filtering in the frequency domain affects the time-domain signal.

R4.a) Load the numpy file **sum_of_sines.npy** and plot the obtained signal. Can you identify the two frequencies in the signal?

R4.b) Compute the signal's frequency spectrum using the Fast Fourier Transform (FFT) algorithm. Plot the magnitude spectrum. Comment on what you observe. Indicate the frequencies (in Hz) of the two sine waves.

R4.c) Filter out the noise directly in the frequency domain. Compare the frequency spectrum of the original and the filtered signals.

R4.d) Use the inverse FFT to obtain the filtered signal in time. Plot both the original and the filtered signals in the same plot. Comment on what you observe.

R4.e) Compare the result you obtained with the result of filtering the original signal, in the time domain, with a running average (moving-average) filter.