MAP555 - Signal processing- Practical Session 1
Digital Signal Processing

The objective of this practical session is to manipulate and understand the digital signal processing tools discussed in the course. The practical session will be done in Python 3 and it is strongly recommended to have a working Anaconda environnement. The different sections of the session should be implemented in Jupyter notebooks and it is strongly recommended to take notes in the notebook during the session.

The individual report for the session will be uploaded on moodle in python notebook format. It is expected to have a working code (reproducible figures) and a short discussion in markdown format for all the results obtained in the practicals session. The end of the report must contain a personal discussion about the session (what was hard to understand and implement, how you would do it newt time, what was new, discussion of relation with the course, personal discussion about how to use theses tools in a professional setting, ...).

Importing libraries

In this section we will use Numpy/Scipy Python libraries for handling numerical data and Matplotlib for plotting them. We will also need to have access to some function in the `scipy.signal` and `scipy.io.wavfile` submodule that have to be imported also.

```
import numpy as np
import pylab as pl
import scipy as sp
import scipy.signal
import scipy.io.wavfile
```

1 Signal generation and sampling

In this section we will generate some digital signals and see the effect of the sampling in term of reconstruction. Finally we will see how to generate audio signals and save it in readable .wav format.

1.1 Signal generation

We will generate samples from the following continuous signal

$$x(t) = \sin(2\pi f_0 t) + \cos(2\pi f_1 t)$$

where $f_0 = 2Hz$ and $f_1 = 3f_0$.

1. Implement a python function `def x(t):` that returns the values of $x(t)$ from a list of values in numpy `np.array` format (`np.sin,np.cos,np.pi`).
2. Generate a finely sampled signal at sampling frequency $f_{s0} = 1000Hz$ over $N_0 = 1024$ samples:
   - Generate an array `t0` of N time samples t of values $t = \frac{n}{f_s}$ (`np.arange`).
   - Call the function `x` ot the time vector and store the resulting values $x[n]$ in vector `x0`.

- Plot the signal with correct time axis in seconds (`pl.figure,pl.plot,pl.title`).

3. Generate a sampled signal `xn` at sampling frequency $f_s = 20$ over $N = f_s$ samples (1 sec sampling).
4. Plot simultaneously `x0` and `xn`. For `xn`, use the plot style '-o' in order to see the position of the samples.

1.2 Signal reconstruction

1. What is the necessary sampling frequency f_s ensuring that the signal $x(t)$ can be reconstructed.
2. Code a function `def rec_sinc(xs,ts,fs,t):` that reconstructs a signal at time `t` from samples `xs,ts` at frequency `fs` (`for`,`np.sinc` check documentation for `np.sinc`).
3. Plot simultaneously `x0`, [language=Python]|xn| with style '-o' and the interpolation of `xn` on `t0`. What happens on the border of the sampling window ?
4. Change the sampling frequency from $f_s = 20$ to $f_s = 10$. What happens to the reconstruction?

1.3 Audio signal generation

In this part of the Practical session we will work with audio sequences. In order to do that we will use `scipy.io.wavefile` to load and save .wav files. one can also listen to audio directly in python using the library `sounddevice` that can be installed with `pip`. Note that the generated signals will be only plotted and listened to in this section but we will study their frequency components in the next section.

1. Generate 1 second of sine wave of magnitude 0.5 and of frequency $f_0 = 425$ Hz sampled at $f_s = 8000Hz$. Save it as a wave file (`sp.io.wavfile.write`) and listen to it (or listen directly from python with `sounddevice.play`). It is the dial tone of European phones.
2. One can generate musical notes from their MIDI number m where the frequency is expressed as

$$f_m = 440 * 2^{\frac{m-69}{12}}$$

   One can see that there is 12 semi-tones in order to increase one octave. the MIDI note $m = 69$ is the $A4$ in english notation (la in french) and is the pitch standard used to tune instruments for concerts. The list of the notes and their corresponding name and frequencies is available online[1] Code a function `def get_note(m,fs,l):` that returns note `m` played for a length of `l` seconds at frequency `fs`.
3. Save the note $m = 69$ in file "A4.wav". Listen to several other the MIDI notes. What happens for $m = 117$ ($A8$) when saved at sampling frequency $f_s = 8000Hz$?
4. Code a sequence of the concatenation of notes $[70, 72, 68, 56, 63]$ (1 sec each, `np.concatenate`). Save the sequence as file "seq.wav". Do you know where this sequence come from?
5. Saturation can occur when amplifiers reach their maximum amplitude. The effect of saturation can be reproduced using a clipping in a sine. Compare the signal for note $m = 69$ at $440Hz$ for different values of clipping (`np.clip`). Save the note when using clipping in a file "A4clip.wav". What will be the effect of the saturation on the frequency content of the signal?
6. Generate the signal

$$x(t) = \sin(2\pi(f_0 t + \frac{c}{2}t^2))$$

   with $f_0 = 100Hz$ and $c = 500$ for 1 second at sampling frequency $f_s = 8000Hz$. This signal is called a chirp and corresponds to frequency modulation. Save the signal in file "chirp.wav"

[1] https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

2 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

In this section we will study the discrete Fourier transform and see how it can be used to interpret the content of signals.

2.1 Basis functions and Discrete Fourier Transform Matrix

1. Compute the Fourier basis functions for all k for signal of size $N = 32$. Plot their real and imaginary parts in a figure (`np.exp,1j,np.pi,np.real,np.imag`).
2. Compute the Discrete Fourier Transform (DFT) matrix for $N = 32$ and visualize it as an image (`pl.imshow`).
3. For a signal of length $N = 1024$ such that $x[n] = \cos(2 * \pi f_0 n)$ and $f_0 = \frac{k}{N}$ for $k < N/2$ positive integer compute the FT matrix and apply it to the signal. Then visualize its magnitude in the frequency domain.
4. Change f_0 to a large value suck that $k > N$, what happens to the spectrum?
5. Change $f_0 \in \mathbb{R}^+$ to a non integer value, what happens to the spectrum?

2.2 Fast Fourier Transform

1. Time a computation of the DFT with and without the Fourier Matrix pre-computation for N=1024 (`time.clock()`).
2. Time the computation of the FFT for the same signal (`np.fft.fft`) and compare it to the two other timings.
3. Compute and store the computational time of DFT, DFT+matrix and FFT for $N = 2^k$ sampled logarithmically from $k = 2$ to $k = 12$. Plot ina loglog plot the different computational times as a function of N (`pl.loglog`). Discuss.

2.3 Interpreting signals

For all signals described below, do the following steps:

1. Load the signal in memory (`sp.io.wavefile.read` for .wav file of `np.load` for .npz file) and store both the signal `x` and the sampling frequency `fs`.
2. Plot the signal in time with proper x axis (time in seconds).
3. Plot the magnitude of the signal FFT with corresponding real life frequency centered around 0 (`np.fft.fft,np.fft.fftfreq,np.fft.fftshift`).
4. Interpret and discuss the properties of the signal in frequency using information provided in the signal description below. Recover physical parameters such as time constant of fundamental frequencies of the signals when possible. You can zoom on part of a plot using `pl.xlim([xmin,xmax])`.

Here is the list of signal and their corresponding filename Some have been saved in previous section of the practical session and the others can be downloaded from moodle ()`data_TP1.zip`):

- `A4.wav` contains the MIDI note $m = 69$.
- `A4clip.wav` contains the MIDI note $m = 69$ with clipping corresponding to a saturation effect.
- `seq.wav` contains the sequence of note generated in section 1.3 .

- `chirp.wav` contains a chirp frequency modulation signal. What are the instantaneous frequencies in this signal? What is the support of its spectrum?
- `uku.wav` and `uku2.wav` contain one note badly played on a ukulele by your professor. What are the notes played? What are their corresponding MIDI number.
- `drum.wav` is a recording of a drum playing containing both bass drum and cymbal corresponding to a low frequency and high frequency signal.
- `stairway.wav` and `stairwayb.wav` contains 10 seconds of the start of a well known song where the second file has been corrupted by noise. Zoom in on the low frequencies and find the mode. What is the midi note corresponding to this mode that is the most played note in the sequence? What is the frequency support of the noise?
- `ecg.npz` is the recording of an ElectroCardioGram. Can you see the average beats per minute in the spectrum? Was the signal recorded in Europe or in the US? Where is the noise situated in the frequency domain?
- `conso.npz` is the recording of of the usage in Watt of the Drahi-X Novation Center building with a sampling period of 1 min for about 4 weeks. Recover the days and weeks in the shape of the temporal signal. In the frequency domain, zoom in on the low frequencies in particular those corresponding to 1 day and 1 week periodicity.

3 Digital filtering

In this section we will study several digital filters and apply them on signals.

3.1 Ideal filtering

1. Load the signal in the file "stairwayb.wav". We want to attenuate the noise by cuting the whole frequency band where noise is present.
2. Compute the FFT of the signal and plot its magnitude in the Fourier domain. Select a cutoff frequency f_c for an ideal low-pass filter.
3. Apply an ideal low ideal filter with a frequency cutoff f_c (`abs`,`<`,`np.fft.ifft`,`np.fft.ifftshift`). Listen to the filtered signal. Keep in mind that saving a wav file in float format clips the values between -1 and 1 so the signal should be scaled properly in order to avoid saturation.
4. Use an ideal filter to select only the note with the lower frequency in signal "seq.wav". Listen to the filtered signal to check that only on note remain.
5. Compute the inverse Fourier transform of the ideal low pass filter frequency response in order to get its circular convolution impulse response. Do the same for the ideal high-pass filter. Check for both that the the 0 frequency is respectively passed and cut by computing the static gain.

3.2 Digital filter design

In real life applications, one often need to design causal filters. This is done by estimating coefficients for an FIR or IIR filter of finite order approximating continuous time systems such as Butterworth of Chebychev filters.

1. Compute the coefficients of a discrete FIR Butterworth filter for a normalized cutoff frequency of $f_c = 0.2$ (for a sampling frequency of 1) of order $n = 2$ (`sp.signal.butter`).

2. Implement a function `def freq_resp(a,b,f)` that returns the frequency response to an IIR filter `a,b` for a list of frequencies `f`. Plot the frequency response for butterworth filter of orders $n = 1, 2, 3, 4$.

3. Apply the filter to the noisy signal "stairwayb.wav" (`sp.signal.lfilter`). What is the equivalent cutoff frequency in Hz? At which order the filter is strong enough to attenuate well the noise?

4. Compute the coefficients of a discrete FIR Chebychev filter of type 1 for a normalized cutoff frequency of $f_c = 0.2$ (for a sampling frequency of 1) or order $n = 2$ and allowing ripples of 1dB in the bandpass (`sp.signal.cheby1`). Plot the frequency response of both Buterworth an Chebychev filter of same order on the same plot.

5. Apply the Chebychev filter to the noisy signal "stairwayb.wav". What happens for an order $n = 50$? Does it happen for the Butterworth filter of the same order?

3.3 Source separation and denoising

1. Design filters (IIR of ideal) that allow a coars sseperation of the different sources in signals : "drum.wav", "seq.wav".

2. Design filters for the signals "ecg.npz" and "conso.npz" so as to better see the important signal (respectively heart beats and power usage).