COMP.SGN.120 Introduction to Audio Processing

Exercise 3
Week 47

In this exercise, you will familiarize yourself with the sound synthesis, time stretching and pitch shifting using a phase vocoder. There is a bonus problem, which is optional, no grading, to test your skills further. **The submission should consist of a Jupyter notebook with your observations and the python code**. Refer to the general notes (given in the end) before starting the exercise.

Re-cap:

- Windowing (here is a good definition with nice illustrations).
- Librosa for audio signal analysis in Python video.

Problem 1: Synthesizing a Signal Envelope with ADSR (0.5 point)

Part 1: Implement ADSR Envelope

1. Objective: Implement a Python function envelope(a, d, s, sd, r, fs) to create an ADSR envelope, which will define how the amplitude of a note changes over time.
2. Input Parameters:
    - a: Attack duration (in seconds)
    - d: Decay duration (in seconds)
    - s: Sustain level (in the range [0, 1])
    - sd: Sustain duration (in seconds)
    - r: Release duration (in seconds)
    - fs: Sampling frequency (in Hz)
3. Output:
    - t_env: A time vector, sampled at fs Hz, with a length equal to the total duration of the ADSR envelope (a + d + sd + r seconds).
    - env: The corresponding ADSR envelope values at each time step in t_env.
4. Steps:
    - **Attack**: The amplitude should increase linearly from 0 to 1 over the attack duration.
    - **Decay**: The amplitude should decrease linearly from 1 to the sustain level (s) over the decay duration.
    - **Sustain**: The amplitude should hold steady at the sustain level over the sustain duration.
    - **Release**: The amplitude should decrease linearly from the sustain level to 0 over the release duration.

Part 2: Test with a Synthesized Note

1. Objective: Generate a synthesized signal x(t) by summing sine waves with random amplitudes and phases for multiple harmonics.
2. Signal Generation:
    - Define the fundamental frequency f0 = 440 Hz (A4 note).
    - Use the equation

      $x(t) = \sum_{k=1}^{N} a_k \sin(2\pi k f_0 t + \phi_k)$
    - where a_k are random amplitudes, and ϕ_k are random phases for each harmonic k.
    - Use N = 5 harmonics for simplicity.
3. Apply the ADSR Envelope:
    - Use the function envelope(0.2, 0.2, 0.7, 0.4, 0.2, fs=10000) to generate the ADSR envelope and apply it to the signal x(t).

Part 3: Visualization

1. Plotting: Create a 3x1 subplot to visualize the results:
    - Panel 1: Plot the original signal x(t).
    - Panel 2: Plot the ADSR envelope env.
    - Panel 3: Plot the enveloped signal x(t) * env.

Problem 2: Implement a time stretching algorithm using phase vocoder (1.5 point)

In order to do this you will have to implement an analysis synthesis loop. The analysis part you did in Exercise 2, Problem 1, and the synthesis part is the exact reverse of it (An example loop is included in *exercise3.py*). In addition to it, you have to modify the DFT phase for each signal frame as shown in Fig 1. The time stretching is achieved by using different overlap values for analysis and synthesis windows. More specifically, it is given by the ratio of hop lengths (number of samples between the start of consecutive frames) for synthesis and analysis steps, i.e,

$$R = \frac{H_s}{H_a}$$

A ratio of above one implies that the signal has been stretched and vice versa. However, time stretching will lead to phase mismatch between frames in analysis and synthesis and hence we need to correct the phase (A nice illustration of this phenomenon can be found from here on slides 155-156).

Use the given audio and select square root of periodic Hann window (*scipy.signal.hann(winlength, sym=False)*) of length 32 ms as the window function. Take analysis hop length H_a to be 8 ms and R to be 1.4.

Now go through the following steps in a loop:

1. Read the audio in frames of length 32 ms, multiply it using the chosen window and compute DFT (FFT size equal to window length).

2. Save the DFT magnitude (will be used in 6) and DFT phase (will be used in 3 and next iteration of the loop).
3. Use the given function *delta_phi* to compute unwrapped phase difference between consecutive frames. It takes the phase values for the current and previous frames as arguments. For the first frame take previous analysis phase to be zero.
4. Compute the synthesis phase as following,

   *previous synthesis phase + R * delta_phi .*

   For the first frame take *previous synthesis phase* to be zero. Comment on why we need to scale *delta_phi* here.
5. Wrap the computed synthesis phase by using the given *princearg* function between (-pi, pi]. This will give you the required synthesis phase.
6. Use magnitude DFT computed in 1 and synthesis phase computed in 6 for computing inverse DFT of the synthesis frame.
7. Do overlap add reconstruction to get time domain synthesis signal. Pay attention to the fact that analysis and synthesis frames have different hop lengths while looping through the signal.

Play the reconstructed signal. Plot it in the same plot as the original signal.

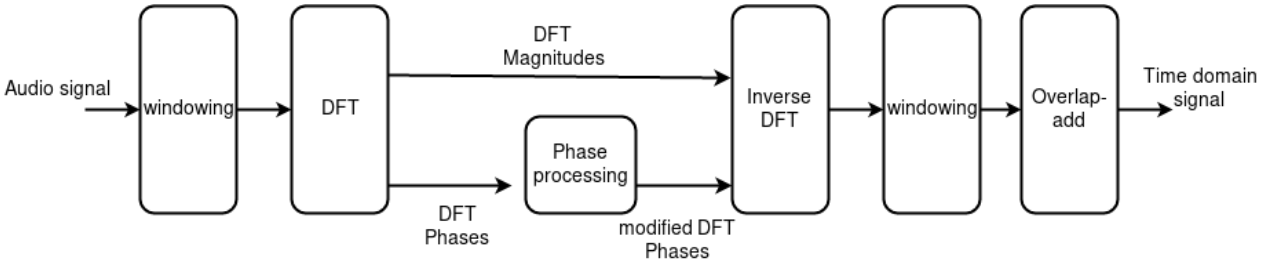

Fig 1. Processing for Time stretching using phase vocoder.

Bonus question: In the above problem add pitch shifting.(Hint see lecture slides)

General notes:

1) Only submissions in python will be graded. It is advisable to use python framework for audio signal processing not only due to its open-source nature and ready availability of useful libraries but also because it has become a popular choice for prototyping audio focused machine learning methods in recent years.

2) If you are using TUNI systems to do the exercise, it is advisable to have your own python installation so that you can install any additional libraries e.g., *librosa, sounddevice*. Download

Anaconda (*https://www.anaconda.com/distribution/*) to your own directory and install. Any additional libraries can then be installed using conda install, e.g. for *librosa* and *sounddevice*,

conda install -c conda-forge librosa
conda install -c conda-forge python-sounddevice
(-c flag here refers to the channel the library is being downloaded from)

Please ask the teaching assistant for help if you need assistance with this.

3) There may be implementation differences between libraries when it comes to reading a .wav file. For example, *librosa* supports floating-point values and rescales the input audio to [-1, 1] while *scipy* does not do that. To avoid confusion, it is advisable to use *librosa* as the main library for audio manipulation.

4) If you are using *librosa* to read audio, ensure that you put sampling rate to *None* to avoid resampling the audio to 22050 Hz which is the default behaviour in *librosa,* i.e.,

librosa .load('audio.wav', sr=None)

5) A periodic hann window can be generated, e.g., using *scipy.signal.hann(winlength, sym=False)*. While choosing a window for analysis and synthesis processing, the chosen window should satisfy constant overlap-add (COLA) condition.