COMP.SGN.120 Introduction to Audio Processing

Exercise 3
Week 47

In this exercise, you will familiarize yourself with the Frequency modulation (FM) 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 report of your observations and the python code (preferable Jupyter notebook)**. 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: Frequency modulation (FM) sound synthesis (0.5 point)

Create an FM synthesis signal with the following parameters:
Sampling rate: 16000 Hz
Carrier frequency: 880 Hz
Modulation frequency: 220 Hz
Amplitude: 1
Modulation index: 2
Signal duration: 1 s

Now plot the signal and its DFT spectrum. Play the sound.
Remember to create a time and frequency vector when plotting the results.

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.

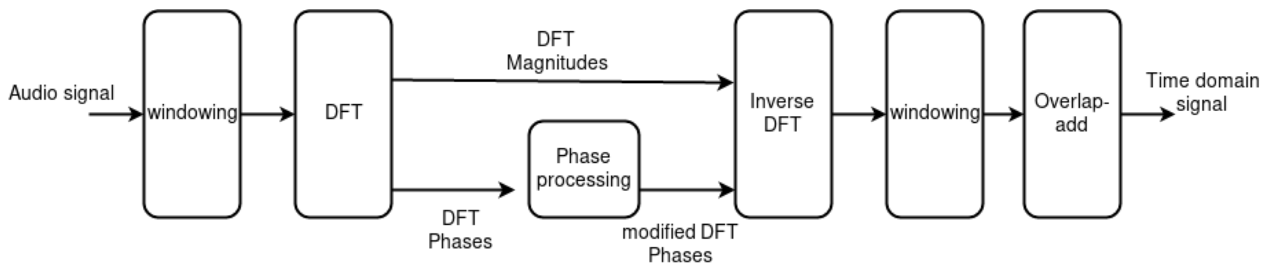

Fig 1. Processing for Time stretching using phase vocoder.

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 focussed 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.