Metronome

Shrey Mehta

***Abstract*—The task is to make an infinite loop beat generator whose beat frequency can be modified in real-time without closing the current output audio signal.**

We will be given the input signal and the initial beat frequency. So, in the initial input signal, we have two cases:

- One beat audio signal
- Multiple beat audio signal

If we have the audio signal with a single beat, we already know the time period and the frequency of the input audio signal. But, in the case of the signal with multiple beats, we will need to use the auto-correlation method (since we have the entire signal), to estimate the time period of the input signal and deprecate the input signal to only one time period.

Now, if we want to run the beats in the infinite loop, we keep repeating the beats over the period of time. The main adjustments come when the beat frequency is changed suddenly, and we need to adjust the output audio signal in real-time. We are not supposed to close the current output stream and generate a new one, but we have to change the value of the next beat according to the new adjusted frequency.

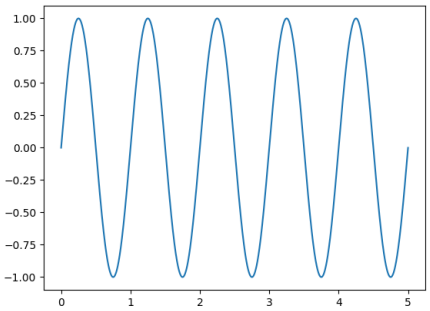
We can follow the following steps in order to achieve this:

1) Take the input signal $\{x[n]\}$ as input (use pyaudio)
2) **Find the time period** (T): In the case of a one-beat signal, the value of auto-correlation will come out to be 0, so, in that case, the value of T is the duration of the input signal, else we can estimate the value of T using the peaks of the auto-correlation plot.
3) **Formulate output signal**: Consider the output signal to be of the form

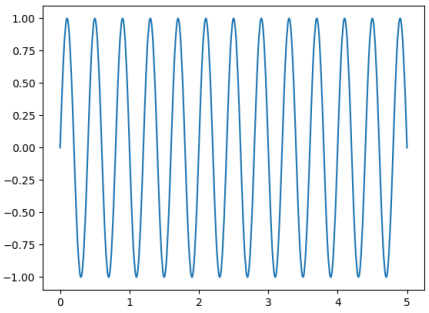
$$\text{x'[n]} = \text{x}[(\text{n*T*N}_0)_N]$$

where N_0 is the current value of the beat frequency

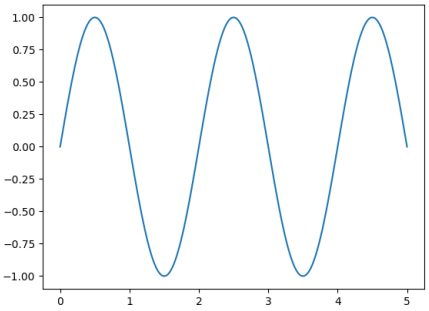
Example:
Consider the input signal to be a sinusoidal wave of the form.



We can see that the time period of the signal (T) is 1 sec. If at any moment of time, we change the beat frequency to 5 per second, our signal, according to the formulation would look like this (compression of the original signal):



If we change the beat frequency to 0.5 per second, our signal, according to the formulation would look like this (extension of the original signal):



Ensuring non-breakage of the signal: In order to ensure that the signal doesn't break when its frequency of it is varied in real-time, we have maintained the array storing the amplitudes at the intervals $\frac{1}{f_s}$, where f_s is the sampling rate, so we can just **update the array** before giving the output for that iteration. We have maintained an infinite while loop which initially checks for any changes in the beats per second by the user, then updates the array accordingly and outputs the audio of that iteration. The only delay is the insignificant computation delay during the input and output of the signal. We can use **parallel processing** for computing the loop for updating the array (use Dask) to minimise the delay.

Image processing: Since the input signal may be of any kind (i.e. may contain random noise or may not be expressible in terms of sinusoids), we can multiply the input signal $x[n]$ with a smoothening window function, like the Hann window or the Gaussian window. It has been seen Hann window is used commonly to avoid sudden fluctuations in the signal, so it can be useful in our case as it makes the output sound more continuous and pleasant for the user.

$$\text{w[n]} = 0.5\big(1\text{-cos}\big(2\pi n/(N-1)\big)\big)$$

So, after finding out the time period in step 2, we can smoothen the audio signal by using

$$\text{xs[n]} = \text{x[n]w[n]}$$
$$\text{x'[n]} = \text{xs}[(\text{n*T*N}_0)_N]$$