Demo 10 Exercises: Vibrato

DSP Lab (ECE 4163 / ECE 6183)

2023

Demo files

`play_vibrato_simple.py`
`play_vibrato_interpolation.py`
`myfunctions.py`
`author.wav`

The demo program `play_vibrato_simple.py` is a simple implementation of the vibrato effect. This implementation is poor because the time-varying fractional delay is implemented by rounding the delay to an integer number of samples. For better audio quality, interpolation is usually used instead, as in the demo program `play_vibrato_interpolation.py`

Exercises

1. Modify the vibrato demo program `play_vibrato_simple.py` so it plays a stereo output signal. Use different vibrato parameters in the left and right channels.

2. Modify the vibrato demo program `play_vibrato_simple.py` so the audio input is from the microphone (not a wave file).

3. The vibrato demo program `play_vibrato_interpolation.py` uses linear interpolation. Write a version that uses quadratic (or cubic) interpolation.

4. Write a Python program to implement the flanger effect. Use interpolation for an improved result. As described in Chapter 2 of *Audio Effects: Theory, Implementation and Application*, the flanger effect is like the vibrato effect but it additionally has a direct path, as shown in the figure. The input signal should be read from a wave file.

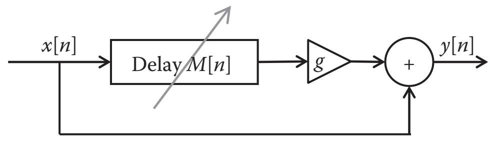

Figure 2.11
Block diagram of a basic flanger without feedback. The delay length $M[n]$ changes over time.

5. Write a Python program to implement the chorus effect. See Chapter 2 of *Audio Effects: Theory, Implementation and Application.*

6. Modify the Python demo program `play_vibrato_interpolation.py` so that the low-frequency oscillator (LFO) is a periodic function other than a sinusoid. For example, it could be a periodic triangle wave, a periodic square wave, or a periodic exponential. For example, see Fig. 2.6 on page 33 in the textbook *Audio Effects*. Comment on your observations. SUBMIT

   Write your program so that it saves the output signal as a wave file (and plays the output signal to the output audio device).

   Submit your Python program, an input wave file, and the output wave file. Submit your comments as a separate file.

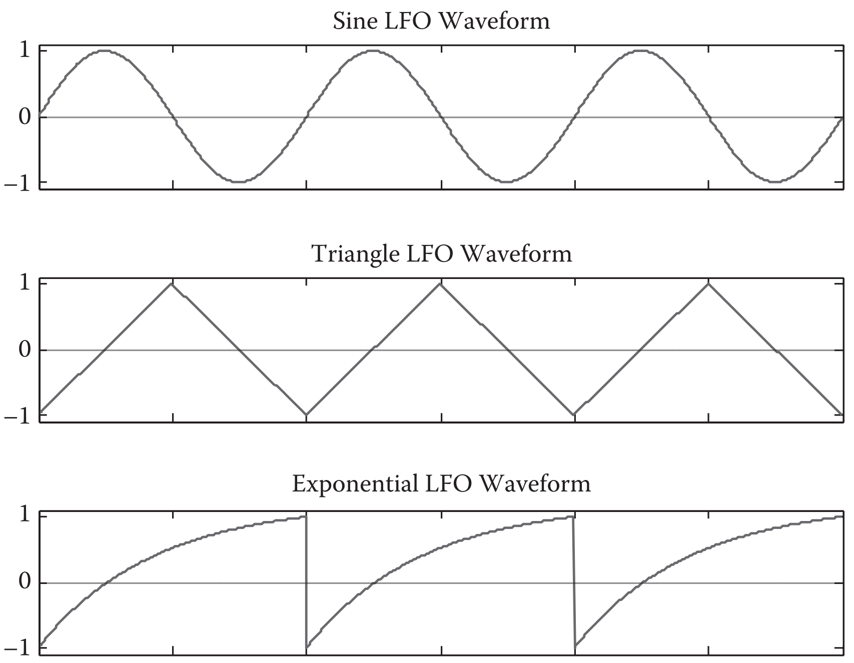

FIGURE 2.6
Three commonly used low-frequency oscillator (LFO) waveforms.

as a function of input depends on the time or, in discrete form, the sample number. This is most often accomplished by driving the effect (making some parameters explicitly a function of time) with a low-frequency oscillator (LFO). This is the case for vibrato and the other delay line-based effects described in the following sections.

LFOs do not have a formal definition, but they may be considered to be any periodic signals with a frequency below 20 Hz. They are used to vary delay lines or as modulating signals in many synthesizers, and they will be used in many of the effects featured later in the book. Like their audio frequency counterparts, LFOs typically use periodic waveforms such as sine, triangle, square, and sawtooth waves. However, any type of waveform is possible, including user-defined waveforms read from a wave table.

Figure 2.6 depicts three of the most commonly used waveforms. Different LFO waveforms are preferred for different effects. In the vibrato effect, the delay length is typically controlled by a sinusoidal LFO:

$$M[n] = M_{avg} + W\sin\left(2\pi nf/f_s\right) \tag{2.18}$$

where M_{avg} is the average delay (for real-time effects, it is chosen so that $M[n]$ is always nonnegative), W is the width of the delay modulation, f is the LFO frequency in Hz, and f_s is the sample rate. The rate of change from one sample to the next can be approximated by the continuous time derivative: