Digital Signal Processing Lab

Demo 10 - Exercise 6 (Vibrato with non-sinusoidal LFO)

Saad Zubairi
shz2020

September 17th, 2025

Solution

Solution Text

Solution

In the solution, the sinusoidal LFO was replaced with a **triangle wave LFO**. Apart from this, the program will now also **save the processed output signal as a WAV file**, while still playing the signal through the audio device.

Code Changes and Additions Made

Below are the changes and additions made to the original file.

- Added code to open a new wave file for writing:

```
# Also save to file
output_wav = wave.open("output_vibrato.wav", ’w’)
output_wav.setnchannels(1)
output_wav.setsampwidth(2)
output_wav.setframerate(RATE)

```

Snippet 1: Open output wave file

- Added a **triangle wave LFO** instead of sinusoid:

```
# -------- LFO: Triangle wave instead of sinusoid --------
phase = (n * f0 / RATE) % 1.0
if phase < 0.5:
    lfo = (phase * 4.0 - 1.0)    # -1 to +1 rising
else:
    lfo = (3.0 - phase * 4.0)    # +1 to -1 falling
# --------------------------------------------------------
kr = i1 + Wd * lfo

```

Snippet 2: Triangle wave LFO

- Each computed output frame is now also written to the file:

```
output_wav.writeframes(output_bytes)

```

Snippet 3: Write frames to file

- Finally, closed the output file at the end:

```
output_wav.close()

```

Snippet 4: Close output file

Addendum: Full Code

```python
# play_vibrato_interpolation.py
# Reads a specified wave file (mono) and plays it with a vibrato effect.
# (Time-varying delay using interpolation)
# Modified: LFO is now a triangle wave, and output is saved as WAV

import pyaudio
import wave
import struct
import math
from myfunctions import clip16

# wavfile = 'decay_cosine_mono.wav'
wavfile = 'author.wav'
# wavfile = 'cosine_300_hz.wav'

print('Play the wave file: %s.' % wavfile)

# Open wave file
wf = wave.open(wavfile, 'rb')

# Read wave file properties
RATE        = wf.getframerate()
WIDTH       = wf.getsampwidth()
LEN         = wf.getnframes()
CHANNELS    = wf.getnchannels()

print('The file has %d channel(s).'         % CHANNELS)
print('The file has %d frames/second.'      % RATE)
print('The file has %d frames.'             % LEN)
print('The file has %d bytes per sample.'   % WIDTH)

# Vibrato parameters
f0 = 2          # LFO frequency in Hz
W = 0.015       # Sweep width (seconds)
Wd = W * RATE   # in samples

# Buffer
BUFFER_LEN = 1024
buffer = BUFFER_LEN * [0]

kr = 0
i1 = kr
kw = int(0.5 * BUFFER_LEN)

print('The buffer is %d samples long.' % BUFFER_LEN)

# Output stream
p = pyaudio.PyAudio()
stream = p.open(format      = pyaudio.paInt16,
                channels    = 1,
                rate        = RATE,
                input       = False,
                output      = True )

# save to file
output_wav = wave.open("output_vibrato.wav", 'w')
output_wav.setnchannels(1)
output_wav.setsampwidth(2)
output_wav.setframerate(RATE)

print ('* Playing...')

for n in range(0, LEN):

    input_bytes = wf.readframes(1)
    x0, = struct.unpack('h', input_bytes)

    kr_prev = int(math.floor(kr))
    frac = kr - kr_prev
    kr_next = kr_prev + 1
    if kr_next == BUFFER_LEN:
        kr_next = 0

    y0 = (1-frac) * buffer[kr_prev] + frac * buffer[kr_next]

    buffer[kw] = x0

    # -------- LFO: Triangle wave instead of sinusoid --------
    # Normalized phase: goes from 0 to 1 each cycle
    phase = (n * f0 / RATE) % 1.0
    if phase < 0.5:
        lfo = (phase * 4.0 - 1.0)   # -1 to +1 rising
    else:
        lfo = (3.0 - phase * 4.0)   # +1 to -1 falling
    # --------------------------------------------------------

    kr = i1 + Wd * lfo
    if kr >= BUFFER_LEN:
        kr = kr - BUFFER_LEN
    if kr < 0:
        kr = kr + BUFFER_LEN

    i1 = i1 + 1
    if i1 >= BUFFER_LEN:
        i1 = i1 - BUFFER_LEN

    kw = kw + 1
    if kw == BUFFER_LEN:
        kw = 0

    output_bytes = struct.pack('h', int(clip16(y0)))
    stream.write(output_bytes)
    output_wav.writeframes(output_bytes)

print('* Finished')

stream.stop_stream()
stream.close()
p.terminate()
wf.close()
output_wav.close()
```

Snippet 5: Full code