Digital Signal Processing Lab

Demo 20 - Guitar

Saad Zubairi
shz2020

October 20, 2025

Solution

Our program implements the Karplus-Strong plucked-string algorithm in real time using Python and PyAudio.
A circular buffer is initialized with white noise to represent the pluck excitation. Each output sample is generated by reading one value from the buffer, then replacing it with the average of the first two samples multiplied by a damping factor K to simulate energy loss.
The code streams each sample directly to the audio device, ensuring real-time playback without using block processing or `lfilter`.

Key points:

- Sampling rate: Fs=8000 Hz
- Damping factor $K \approx 0.998$ controls sustain
- Frequency determined by buffer size textbfN
- Circular buffer implementation ensures efficient real-time behavior

This setup synthesizes a natural-sounding plucked string tone entirely in Python using direct sample-by-sample processing.

Addendum

Full Code

```python
import numpy as np
import pyaudio
import struct

def karplus_strong(
    frequency=150.0,
    Fs=8000,
    K=0.998):

    duration=10*50/frequency
    #print(duration)
    print("* Playing")
    N = int(Fs / frequency)
    buffer = np.random.uniform(-1, 1, N)
    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16, channels=1, rate=Fs, output=True)

    num_samples = int(duration * Fs)
    MAXVALUE = 2**15 - 1

    # circular buffer index
    idx = 0

    for i in range(num_samples):
        y = buffer[idx]
        next_idx = (idx + 1) % N
        avg = 0.5 * (buffer[idx] + buffer[next_idx])
        buffer[idx] = K * avg  # feedback
        idx = next_idx
        # convert to bytes and play
        sample = struct.pack('h', int(y * MAXVALUE))
        stream.write(sample)

    stream.stop_stream()
    stream.close()
    p.terminate()
    print("* Finished")

karplus_strong(110,K=0.998)
```

Snippet 1: Complete implementation