

# **Digital Signal Processing Lab**

## Demo 61 - Exercise 2 (keyboard with plotting)

Saad Zubairi  
shz2020

October 29, 2025

## Solution

The original program used a single second-order difference equation (a resonator filter) to play a single note, triggered by a keyboard impulse. To enable the ability to play multiple notes/chords simultaneously, the system was extended to manage **12 independent filters**, one for each note in a musical octave.

### Overview of Code Modifications

The changes primarily involve replacing single variables with lists and iterating through these lists in the main processing loop (`my_update`).

- **Frequency and Filter Setup:**
  - **Frequencies:** A list `frequencies` is pre-calculated for  $k = 0$  to 11.
  - **Coefficients:** The coefficients  $(a, b)$  for each frequency are stored in `a_list` and `b_list`.
  - **Filter States:** The state memory for each filter is stored in a list of arrays: `states_list = [np.zeros(ORDER) for _ in range(NUM_NOTES)]`.
  - **Inputs:** Similarly, a list of input vectors `x_list` is used, where each vector receives an impulse when its corresponding key is pressed.
- **Key Mapping and Tracking:**
  - A dictionary `KEY_MAP` (e.g., `'a': 0, 's': 1, ...`) links keyboard keys to the note index  $k$ .
  - A dictionary `KEYPRESS_NOTES` tracks which specific note index  $k$  requires a sound-triggering impulse in the current processing block.
- **The Update Loop (`my_update`):** The single filtering operation is replaced by a loop that processes all 12 notes independently and sums the outputs.

## Addendum: Full implementation

```
1 import pyaudio
2 import numpy as np
3 import tkinter as Tk
4 from scipy.signal import lfilter
5
6 import matplotlib.figure
7 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
8 from matplotlib import animation
9
10 matplotlib.use('TkAgg')
11 # matplotlib.use('MacOSX')
12
13 print('The matplotlib backend is %s' % matplotlib.get_backend())
14
15 # BLOCKLEN: Number of frames per block
16 # BLOCKLEN = 1024
17 # BLOCKLEN = 512
18 BLOCKLEN = 256
19 # BLOCKLEN = 128
20 # BLOCKLEN = 64
21 # BLOCKLEN = 32
22
23 WIDTH = 2 # Bytes per sample
24 CHANNELS = 1 # Mono
25 RATE = 8000 # Frames per second
26 MAXVALUE = 2**15-1 # Maximum allowed output signal value (because WIDTH = 2)
27
28 BLOCK_DURATION = 1000.0 * BLOCKLEN/RATE # duration in milliseconds
29 print('Block length: %d' % BLOCKLEN)
30 print('Duration of block in milliseconds: %.2f' % BLOCK_DURATION)
31
32 # Number of notes in the octave
33 NUM_NOTES = 12
34 # Base frequency (Middle A)
35 f0 = 440.0
36 # Decay time (seconds), kept same for all notes
37 Ta = 1.2
38 ORDER = 2 # filter order
39 # --- End of Polyphony Setup ---
40
41 # Calculate frequencies for the full octave (12 notes)
42 frequencies = [f0 * (2**(k/12.0)) for k in range(NUM_NOTES)]
43 print(f'Frequencies (Hz) for the {NUM_NOTES} notes: {frequencies}')
44
45 ##### Filter parameters for all notes
46
47 # Pole radius, kept same for all notes
48 r = 0.01 ** ( 1.0/(Ta * RATE) ) # 0.01 for 1 percent amplitude
49
50 # Calculate the filter coefficients (a_k, b_k) for each note
51 a_list = []
52 b_list = []
53 # States for each filter (12 independent state vectors)
54 states_list = [np.zeros(ORDER) for _ in range(NUM_NOTES)]
55 # Input signal for each filter (impulse at the middle of the block)
56 x_list = [np.zeros(BLOCKLEN) for _ in range(NUM_NOTES)]
```

```

57
58 for f in frequencies:
59     om = 2.0 * np.pi * f / RATE
60     # Filter coefficients (second-order recursive filter)
61     a_k = [1, -2 * r * np.cos(om), r ** 2]
62     b_k = [np.sin(om)]
63     a_list.append(a_k)
64     b_list.append(b_k)
65
66 # Map keyboard keys to notes (k=0 to k=11)
67 # You can customize this mapping
68 KEY_MAP = {
69     'a': 0, 'w': 1, 's': 2, 'e': 3,
70     'd': 4, 'f': 5, 't': 6, 'g': 7,
71     'y': 8, 'h': 9, "u": 10, 'j': 11
72 }
73
74 # --- End of Filter Setup ---
75
76 # Open the audio output stream
77 p = pyaudio.PyAudio()
78 PA_FORMAT = pyaudio.paInt16
79 stream = p.open(
80     format      = PA_FORMAT,
81     channels     = CHANNELS,
82     rate        = RATE,
83     input       = False,
84     output      = True,
85     frames_per_buffer = BLOCKLEN)
86 # What if you specify a much larger value for frames_per_buffer? (try 8*BLOCKLEN)
87 # (latency)
88 # specify low frames_per_buffer to reduce latency
89
90 # --- Global state for keypresses ---
91 # Use a dictionary to track which note (by index k) was just pressed
92 KEYPRESS_NOTES = {} # Stores {k: True} for notes that need an impulse
93 # --- End of Global state ---
94
95 def my_function(event):
96     global KEYPRESS_NOTES
97     print('You pressed ' + event.char)
98     if event.char == 'q':
99         print('Good bye')
100         root.quit()
101
102     if event.char in KEY_MAP:
103         # Mark the note index k to be played in the next block
104         note_index = KEY_MAP[event.char]
105         KEYPRESS_NOTES[note_index] = True
106         print(f'Playing note k={note_index} at {frequencies[note_index]:.2f} Hz')
107
108 # Define Tkinter root
109
110 root = Tk.Tk()
111 root.bind("<Key>", my_function)
112
113 print('Press keys for sound. (Keys: ' + ', '.join(KEY_MAP.keys()) + ')')
114 print('Press "q" to quit')

```

```

115
116 # Define figure
117
118 my_fig = matplotlib.figure.Figure()
119 my_ax = my_fig.add_subplot(1, 1, 1)
120 # Plot will show the summed output signal
121 [g1] = my_ax.plot([], [])
122 my_ax.set_ylim(-32000, 32000)
123 my_ax.set_xlim(0, BLOCKLEN * 1000.0 / RATE) # Time axis in milliseconds
124 my_ax.set_xlabel('Time (milliseconds)')
125 my_ax.set_title('Summed Output Signal (Chord)')
126
127 my_canvas = FigureCanvasTkAgg(my_fig, master = root) # create Tk canvas from
    figure
128 C1 = my_canvas.get_tk_widget() # canvas widget
129 C1.pack() # place canvas widget
130
131 # Define animation functions
132
133 M1 = np.int64(BLOCKLEN/2) # Location for the impulse
134
135 def my_init():
136     t = np.arange(BLOCKLEN) * 1000/RATE
137     g1.set_xdata(t)
138     return (g1,)
139
140 def my_update(i):
141     global states_list
142     global x_list
143     global KEYPRESS_NOTES
144
145     # Initialize the total output signal for the current block
146     y_total = np.zeros(BLOCKLEN)
147
148     # 1. Apply impulse to inputs of the filters corresponding to pressed keys
149     for k in range(NUM_NOTES):
150         if k in KEYPRESS_NOTES and KEYPRESS_NOTES[k]:
151             # Apply impulse to the input of the k-th filter
152             x_list[k][M1] = 10000.0 # Impulse magnitude
153             # Reset the keypress flag for this note
154             KEYPRESS_NOTES[k] = False
155
156     # 2. Filter the input signal x_k using the k-th filter's coefficients (
    a_k, b_k) and states
157     [y_k, states_list[k]] = lfilter(b_list[k], a_list[k], x_list[k], zi =
    states_list[k])
158
159     # 3. Sum the output of the k-th filter to the total output
160     y_total = y_total + y_k
161
162     # 4. Reset the impulse in the input for the next block
163     x_list[k][M1] = 0.0
164
165     # 5. Process the total output signal
166     y_total = np.clip(y_total, -MAXVALUE, MAXVALUE) # Clipping
167     g1.set_ydata(y_total) # update plot with the summed
    signal
168     stream.write(y_total.astype('int16').tobytes(), BLOCKLEN)
169     return (g1,)

```

```

170
171 my_anima = animation.FuncAnimation(
172     my_fig,
173     my_update,
174     init_func = my_init,
175     interval = 20,    # milliseconds (what happens if this is 200?)
176     blit = True,
177     cache_frame_data = False,
178     repeat = False
179 )
180
181 Tk.mainloop()
182
183 # Close audio stream
184 stream.stop_stream()
185 stream.close()
186 p.terminate()
187
188 print('* Finished')

```

Snippet 1: Polyphonic Difference Equation Synthesis Code