Digital Signal Processing Lab

Demo 52 - Exercise 1 (Bandpass filter with real-time plotting)

Saad Zubairi
shz2020

October 8, 2025

Solution

To solve this, we can start with the `demo 52 - plotting audio prog_05.py` file included with the demo as it contains the animation functions implementation and take the recursive filter from the `demo 06 - filter wave file wave_filter_python.py` file.

The following are the additions and changes made the file for this implementation:

- The filter specifications are implemented wthe constants as follows:

```
    # Bandpass Filter Coefficients
b0 =  0.008442692929081
b2 = -0.016885385858161
b4 =  0.008442692929081

a1 = -3.580673542760982
a2 =  4.942669993770672
a3 = -3.114402101627517
a4 =  0.757546944478829

# Initialization of Delay Elements
x1 = 0.0
x2 = 0.0
x3 = 0.0
x4 = 0.0
y1 = 0.0
y2 = 0.0
y3 = 0.0
y4 = 0.0

```

Snippet 1: Filter initialization

- The block level modulation for the signal is changed to incorporate the filter application:

```
    for n in range(BLOCKLEN):
        x0 = input_block[n]

        y0 = b0*x0 + b2*x2 + b4*x4 - a1*y1 - a2*y2 - a3*y3 - a4*y4

        # Update delays
        x4, x3, x2, x1 = x3, x2, x1, x0
        y4, y3, y2, y1 = y3, y2, y1, y0

        # Clip to 16-bit
        output_block[n] = int(clip16(y0))

```

Snippet 2: Block level output processing

Most of the other implementation largely remains the same. Since this is filter's response is LTI, and the coefficients are constants, we do not need to incorporate `theta` in this.

The demo video is attached with this assignment as

1 Addendum

Here's the full code for the solution:

```
import pyaudio
import struct
import wave
import matplotlib
from matplotlib import pyplot
from matplotlib import animation
import math

def clip16( x ):
    # Clipping for 16 bits
    if x > 32767:
        x = 32767
    elif x < -32768:
        x = -32768
    else:
        x = x
    return (x)

matplotlib.use('TkAgg')
# matplotlib.use('MacOSX')

print('The matplotlib backend is %s' % pyplot.get_backend())       # Plotting
    backend

# Specify wave file
import os
wavefile = os.path.join(os.path.dirname(__file__), 'author.wav')
wf = wave.open(wavefile, 'rb')

# Read wave file properties
RATE        = wf.getframerate()     # Frame rate (frames/second)
WIDTH       = wf.getsampwidth()     # Number of bytes per sample
LEN         = wf.getnframes()       # Signal length
CHANNELS    = wf.getnchannels()     # Number of channels

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)

# Bandpass Filter Coefficients
b0 =  0.008442692929081
b2 = -0.016885385858161
b4 =  0.008442692929081

a1 = -3.580673542760982
a2 =  4.942669993770672
a3 = -3.114402101627517
a4 =  0.757546944478829

# Initialization of Delay Elements
x1 = 0.0
x2 = 0.0
x3 = 0.0
x4 = 0.0
```

```python
y1 = 0.0
y2 = 0.0
y3 = 0.0
y4 = 0.0

# Audio Parameters
BLOCKLEN = 256
BLOCK_DURATION = 1000.0 * BLOCKLEN / RATE  # duration in milliseconds
print('Block length: %d' % BLOCKLEN)
print('Duration of block in milliseconds: %.2f' % BLOCK_DURATION)

# Audio Stream Setup
p = pyaudio.PyAudio()
PA_FORMAT = p.get_format_from_width(WIDTH)

stream = p.open(
    format = PA_FORMAT,
    channels = CHANNELS,
    rate = RATE,
    input = False,
    output = True,
    frames_per_buffer = BLOCKLEN)

# Plot Setup
fig1 = pyplot.figure(1)
fig1.set_figwidth(8.0)
fig1.set_figheight(6.0)

ax1 = fig1.add_subplot(2, 1, 1)
ax2 = fig1.add_subplot(2, 1, 2)

[g1] = ax1.plot([], [])
[g2] = ax2.plot([], [])

def my_init():
    g1.set_xdata([1000 * i / RATE for i in range(BLOCKLEN)])
    g1.set_ydata(BLOCKLEN * [0])
    ax1.set_ylim(-32000, 32000)
    ax1.set_xlim(0, 1000 * BLOCKLEN / RATE)
    ax1.set_xlabel('Time (milliseconds)')
    ax1.set_title('Input Signal')

    g2.set_xdata([1000 * i / RATE for i in range(BLOCKLEN)])
    g2.set_ydata(BLOCKLEN * [0])
    ax2.set_ylim(-32000, 32000)
    ax2.set_xlim(0, 1000 * BLOCKLEN / RATE)
    ax2.set_xlabel('Time (milliseconds)')
    ax2.set_title('Output Signal (Bandpass Filtered)')

    return (g1, g2)

# Animation Update Function
def my_update(i):
    global x1, x2, x3, x4, y1, y2, y3, y4

    input_bytes = wf.readframes(BLOCKLEN)

    # Rewind if end of file
    if len(input_bytes) < WIDTH * BLOCKLEN:
```

```
        wf.rewind()
        input_bytes = wf.readframes(BLOCKLEN)

    input_block = struct.unpack('h' * BLOCKLEN, input_bytes)
    output_block = [0] * BLOCKLEN

    # Filter Processing (Recursive)
    for n in range(BLOCKLEN):
        x0 = input_block[n]

        y0 = b0*x0 + b2*x2 + b4*x4 - a1*y1 - a2*y2 - a3*y3 - a4*y4

        # Update delays
        x4, x3, x2, x1 = x3, x2, x1, x0
        y4, y3, y2, y1 = y3, y2, y1, y0

        # Clip to 16-bit
        output_block[n] = int(clip16(y0))

    g1.set_ydata(input_block)
    g2.set_ydata(output_block)
    output_bytes = struct.pack('h' * BLOCKLEN, *output_block)
    stream.write(output_bytes, BLOCKLEN)

    return (g1, g2)

my_anima = animation.FuncAnimation(
    fig1,
    my_update,
    init_func = my_init,
    interval = 10,
    blit = True,
    cache_frame_data = False,
    repeat = False)

fig1.tight_layout()
pyplot.show()

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

print('* Finished')

```

Snippet 3: Full implementation