# Digital Signal Processing Lab

Demo 12 - Exercise 1 (bandpass filter with block input-output)

Saad Zubairi
shz2020

October 8, 2025

## Solution

In this solution, we simply modify the `wave_filter_python.py` such that it reads and writes the signal in blocks instead of one signal value at a time without using Numpy.

To process blocks instead of singular signal values, we made the following changes in the code:

- We first define the block duration (a value between 10 and 60ms). For the purposes of this demo, we will choose a value of 30ms.

```
    BLOCK_DURATION = 0.03    # 30 ms block duration
    BLOCKLEN = int(RATE * BLOCK_DURATION)
    num_blocks = int(math.floor(signal_length / BLOCKLEN))

```

- We then process each block of the input bytes as per the samples in each of the block in the loop:

```
    for n in range(num_blocks):

```

- For each of the blocks, we will first read that block, convert it to binary data using struct.unpack, and then prepare the output list. Similar pipeline as we did for the singular signal values but a key difference to note here is that since we are dealing with multiple signal values for one block, a scaler representation is insufficient, therefore the binary data is converted to a `tuple` of ints.

```
    # read one block of input
    input_bytes = wf.readframes(BLOCKLEN)

    # if not enough data (end of file), break
    if len(input_bytes) < BLOCKLEN * width:
        break

    # Convert binary data to tuple of ints
    input_block = struct.unpack('h' * BLOCKLEN, input_bytes)

```

- We then prepare our output list, and process the block sample by sample in a nested for loop.

```
    # Prepare output list
    output_block = [0] * BLOCKLEN

    # PROCESS BLOCK SAMPLE-BY-SAMPLE
    for i in range(BLOCKLEN):
        x0 = input_block[i]

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

        # shift delay values
        x4, x3, x2, x1 = x3, x2, x1, x0
        y4, y3, y2, y1 = y3, y2, y1, y0

        # clip and store
        output_block[i] = int(clip16(y0))

```

- And then of course, we pack the block in the binary format and write it to the audio stream.

```
    output_bytes = struct.pack('h' * BLOCKLEN, *output_block)

    # Write block to audio output
    stream.write(output_bytes)

```

This solution ensures the same output as produced by the original individually processed signal values method.

## Addendum

Attached here is the full code:

```
# wave_filter_python_blocks.py
# Modified from wave_filter_python.py to process audio in blocks (10-60 ms)

import pyaudio
import wave
import struct
import math

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

wavefile = 'author.wav'

print('Play the wave file %s.' % wavefile)

# Open wave file (should be mono channel)
wf = wave.open( wavefile, 'rb' )

# Read the wave file properties
num_channels    = wf.getnchannels()     # Number of channels
RATE            = wf.getframerate()     # Sampling rate (frames/second)
signal_length   = wf.getnframes()       # Signal length
width           = wf.getsampwidth()     # Number of bytes per sample

print('The file has %d channel(s).'            % num_channels)
print('The frame rate is %d frames/second.'    % RATE)
print('The file has %d frames.'                % signal_length)
print('There are %d bytes per sample.'         % width)

# Difference equation coefficients
b0 =  0.008442692929081
b2 = -0.016885385858161
b4 =  0.008442692929081

# a0 =  1.000000000000000
a1 = -3.580673542760982
a2 =  4.942669993770672
a3 = -3.114402101627517
a4 =  0.757546944478829

# Initialization
x1 = 0.0
x2 = 0.0
x3 = 0.0
x4 = 0.0
y1 = 0.0
y2 = 0.0
y3 = 0.0
```

```
y4 = 0.0

p = pyaudio.PyAudio()

# Open audio stream
stream = p.open(
    format      = pyaudio.paInt16,
    channels    = num_channels,
    rate        = RATE,
    input       = False,
    output      = True )

BLOCK_DURATION = 0.03   # 30 ms block duration
BLOCKLEN = int(RATE * BLOCK_DURATION)
num_blocks = int(math.floor(signal_length / BLOCKLEN))

print('Processing in blocks of %d samples...' % BLOCKLEN)
print('* Playing...')

for n in range(num_blocks):

    # read one block of input
    input_bytes = wf.readframes(BLOCKLEN)

    # if not enough data (end of file), break
    if len(input_bytes) < BLOCKLEN * width:
        break

    # Convert binary data to tuple of ints
    input_block = struct.unpack('h' * BLOCKLEN, input_bytes)

    # Prepare output list
    output_block = [0] * BLOCKLEN

    # PROCESS BLOCK SAMPLE-BY-SAMPLE
    for i in range(BLOCKLEN):
        x0 = input_block[i]

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

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

        # Compute output value
        output_block[i] = int(clip16(y0))

    # Pack block into binary format
    output_bytes = struct.pack('h' * BLOCKLEN, *output_block)

    # Write block to audio output
    stream.write(output_bytes)

print('* Finished')

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