Demo 6 #4

4. The Python demo program implements the fourth-order difference equation with 8 variables to store past values (i.e., 8 delay units). This is the direct form implementation. But a fourth-order difference equation can be implemented using just 4 variables to store past values (i.e., 4 delay units). The canonical form can be used for this purpose. See the block diagram in Fig. 7.2.4 on page 274 of the text book 'Introduction to Signal Processing' by Orfanidis
   http://www.ece.rutgers.edu/~orfanidi/intro2sp/orfanidis-i2sp.pdf
   The software implementation of the canonical form is shown in Equation 7.2.5 on the same page. Modify the Python demo program to implement the difference equation using the canonical form. Instead of 8 delay units, this form should have just 4 delay units. Verify that the output produced by this implementation is the same as the output produced by the demo program.

Python code: canonical_06.py

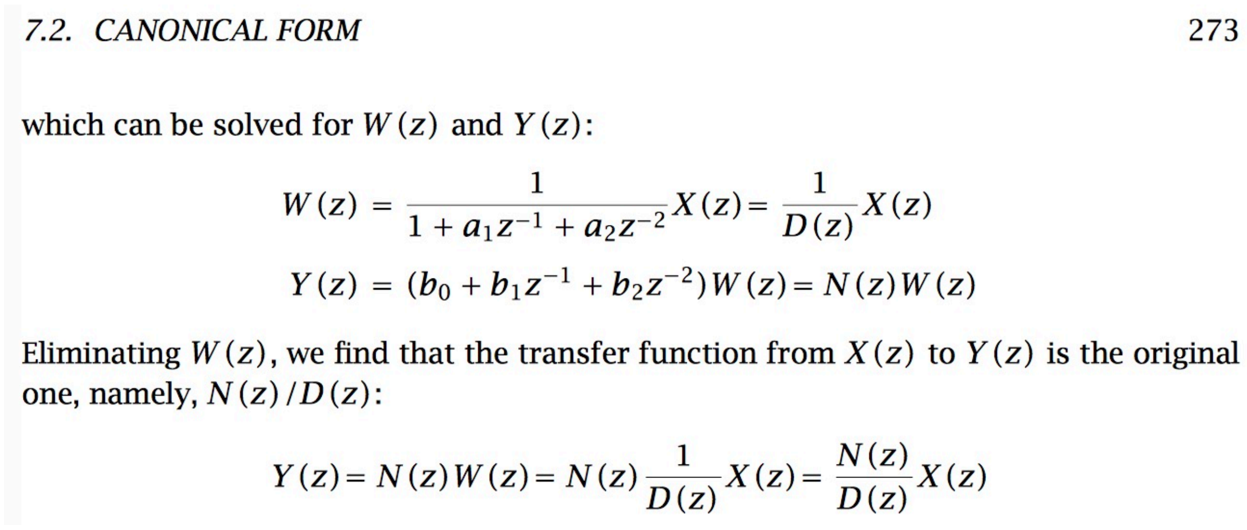
```
import pyaudio
import wave
import struct
import math

from myfunctions import clip16

wavfile = 'author.wav'
# wavfile = 'sin01_mono.wav'

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

# Open wave file (should be mono channel)
wf = wave.open( wavfile, '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
w1 = 0.0
w2 = 0.0
w3 = 0.0
w4 = 0.0
p = pyaudio.PyAudio()

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

# Get first frame from wave file
input_string = wf.readframes(1)

nwf = wave.open('wav_canonical.wav','w')
nwf.setnchannels(num_channels)
nwf.setsampwidth(width)
nwf.setframerate(RATE)

while len(input_string) > 0:

    # Convert string to number
    input_tuple = struct.unpack('h', input_string)  # One-element tuple
    input_value = input_tuple[0]                    # Number

    # Set input to difference equation
    x0 = input_value

    # Difference equation
    w0 = x0 - a1*w1 - a2*w2 - a3*w3 - a4*w4
    y0 = b0*w0 + b2*w2 + b4*w4
    # Delays
    w4 = w3
    w3 = w2
    w2 = w1
    w1 = w0

    # Compute output value
    output_value = int(clip16(y0))    # Integer in allowed range

    # Convert output value to binary string
    output_string = struct.pack('h', output_value)

    # Write binary string to audio stream
    stream.write(output_string)
    nwf.writeframes(output_string)

    # Get next frame from wave file
    input_string = wf.readframes(1)

print('* Finished')

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

Matlab code : make_and_use_filter.m

Comments :

From the link above is :



7.2. CANONICAL FORM 273

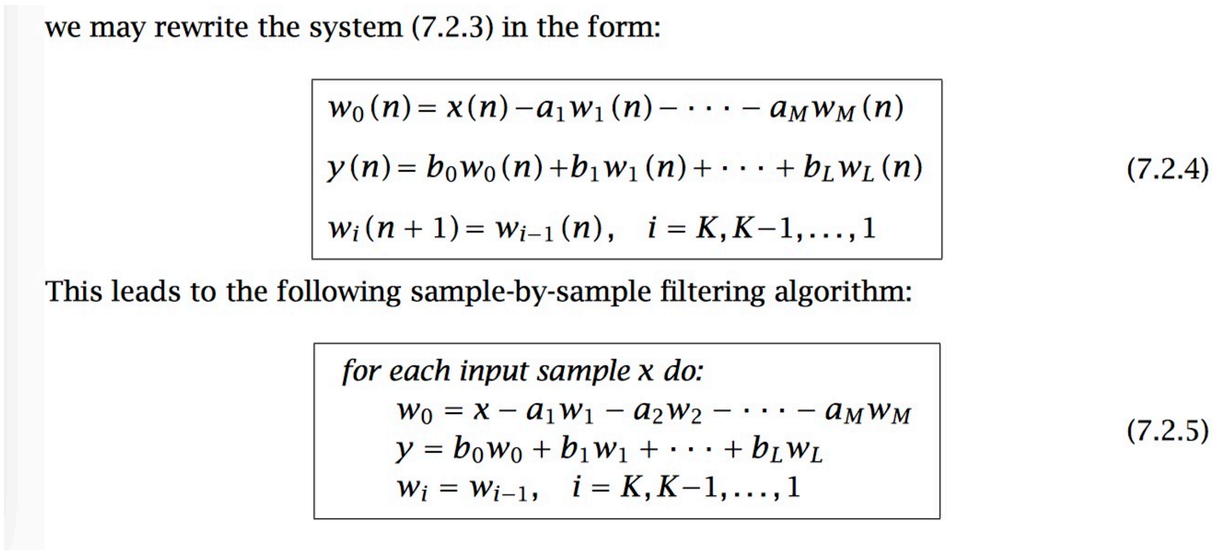
which can be solved for $W(z)$ and $Y(z)$:

$$W(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) = \frac{1}{D(z)} X(z)$$

$$Y(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2}) W(z) = N(z) W(z)$$

Eliminating $W(z)$, we find that the transfer function from $X(z)$ to $Y(z)$ is the original one, namely, $N(z)/D(z)$:

$$Y(z) = N(z)W(z) = N(z)\frac{1}{D(z)}X(z) = \frac{N(z)}{D(z)}X(z)$$



we may rewrite the system (7.2.3) in the form:

$$\boxed{\begin{aligned} &w_0(n) = x(n) - a_1 w_1(n) - \cdots - a_M w_M(n) \\ &y(n) = b_0 w_0(n) + b_1 w_1(n) + \cdots + b_L w_L(n) \\ &w_i(n+1) = w_{i-1}(n), \quad i = K, K-1, \dots, 1 \end{aligned}} \qquad (7.2.4)$$

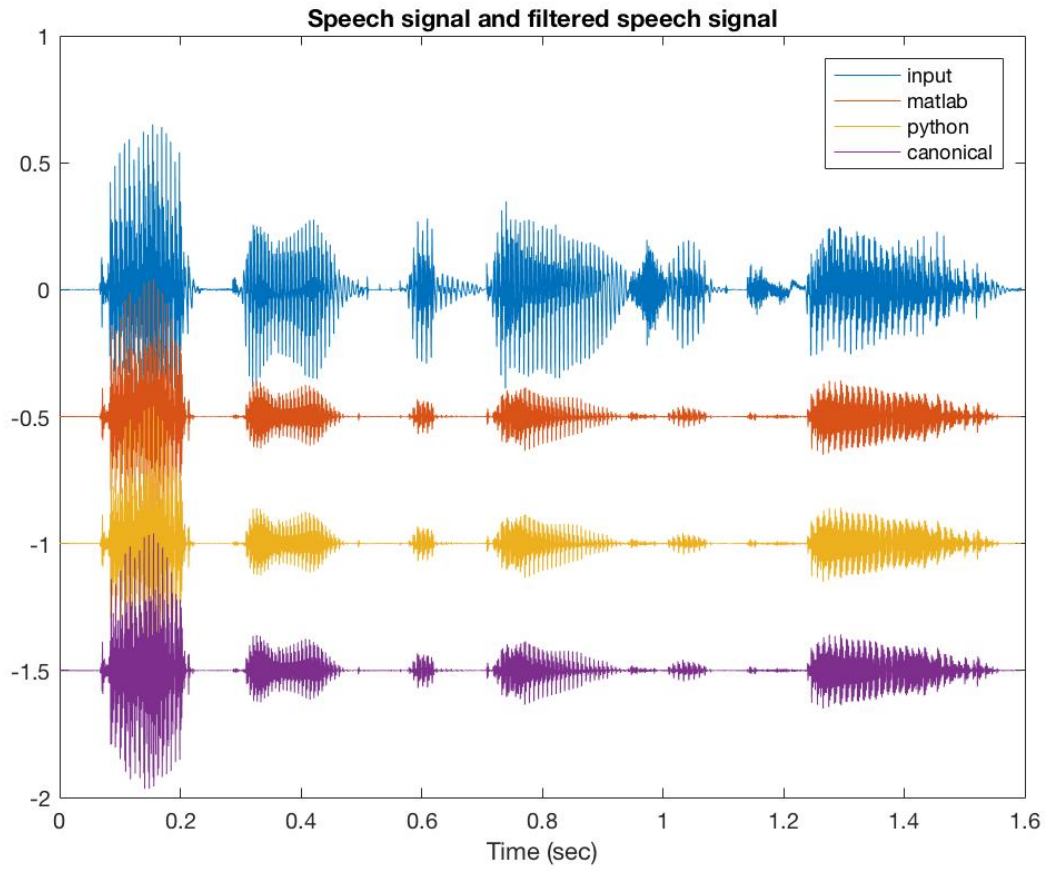
This leads to the following sample-by-sample filtering algorithm:

```
for each input sample x do:
    $w_0 = x - a_1 w_1 - a_2 w_2 - \cdots - a_M w_M$
    $y = b_0 w_0 + b_1 w_1 + \cdots + b_L w_L$
    $w_i = w_{i-1}, \quad i = K, K-1, \dots, 1$
```

(7.2.5)

It shows a canonical form that use w0 to make an intermediary. So I just apply these equations in python and use only 4 delays of w0 to make the filter.

Plus, I modify the maltab code so that it could not only plot the direct form wave but also this canonical form wave in the same figure, it shows the code work well. (The figure are shown below)


Speech signal and filtered speech signal
input
matlab
python
canonical
1
0.5
0
-0.5
-1
-1.5
-2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time (sec)