Filters_and_Noble_Identities_2

March 17, 2017

0.1 Example: (with speech signal using polyphase matrices)

```
In [1]: from sound import *
        import scipy.signal
        x, fs = wavread('speech8kHz.wav')
```

```
('Number of channels: ', 1)
('Number of bytes per sample:', 2)
('Sampling rate: ', 8000)
('Number of samples:', 60246)
```

Listen to it as a comparison:

```
In [2]: sound(x,fs)
```

```
* done
```

Take a low pass FIR filter with impulse response

```
In [3]: h = [0.5, 1, 1.1, 0.6]
```

and a down-sampling factor N=2. Hence we get the z-transform or the impulse response as, $H(z) = 0.5 + 1z^1 + 1.1 \cdot z^2 + 0.6 \cdot z^3$ and its polyphase components as $H_0(z) = 0.5 + 1.1 \cdot z^1$, $H_1(z) = 1 + 0.6 \cdot z^1$ The polyphase components in in the tme domain (in Python) are hence:

```
In [4]: h0 = h[0::2]
        h1 = h[1::2]
```

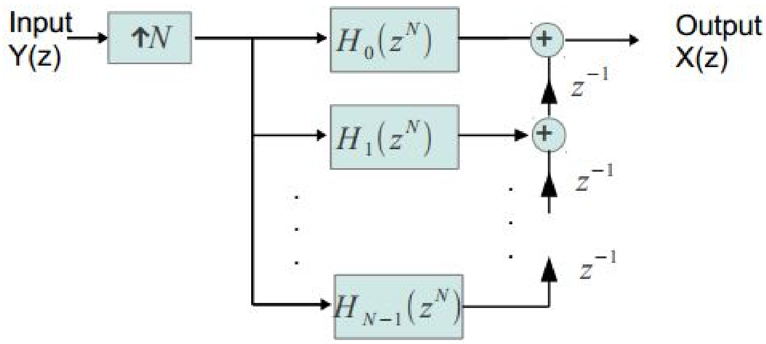
Produce the 2 phases of a down-sampled input signal x:

```
In [5]: x0 = x[0::2]
        x1 = x[1::2]
```

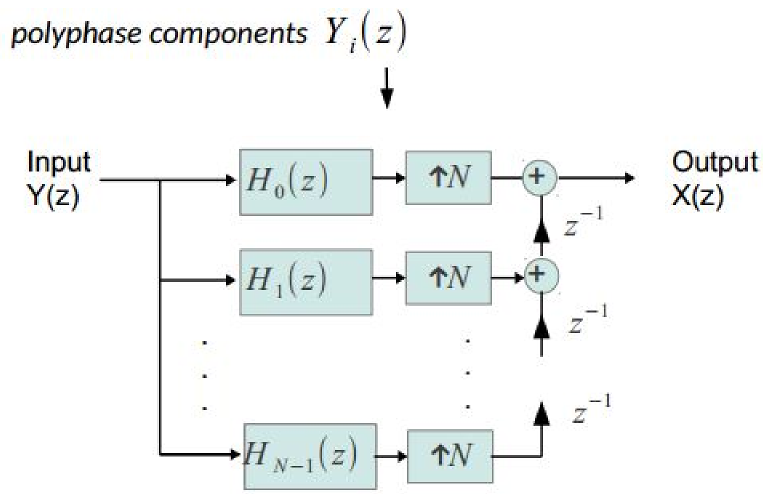
then the filtered and down-sampled output y is

```
In [6]: y=scipy.signal.lfilter(h0,1,x0)+scipy.signal.lfilter(h1,1,x1)
```

Observe that each of these 2 filters now works on a downsampled signal, but the result is identcal to frst filtering and then down-sampling. Now listen to the resultng down-sampled signal:



polyphase



Polyohase2

```
In [7]: sound(y,fs/2)
```

```
* done
```

Correspondingly, **up-samplers** can be obtained with flters operatng on the **lower sampling rate**. Since $H_i(z^N)$ and z^i are linear tme-invariant systems, we can exchange their ordering, $H_i(z^N) \cdot z^i = z^i \cdot H_i(z^N)$ Hence we can redraw the polyphase decompositon for an up-sampler followed by a (e.g. low pass) flter (at the high sampling rate) as follows,

Using the Noble Identtes, we can now shif the upsampler to the right, behind the polyphase flters (with changing their arguments from z^N to z) and before the delay chain,

Again, this leads to a parallel processing, with N flters working in **parallel** at the **lower sampling rate**. The structure on the right with the up-sampler and the delay chain can be seen as a **de-blocking** operaton. Each tme the up-sampler let a complete block through, it is given to the delay chain. In the next time-steps the up-samplers stop letting through, and the block is shifed through the delay chain as a sequence of samples. This can also be seen as a **parallel to serial conversion.**

With the polyphase elements $Y_i(z)$ the processing at the lower sampling rate can also be writen in terms of **polyphase vectors**

0.1.1 $Y(z) \cdot [H_0(z),, H_{N1}(z)] = [Y_0(z),, Y_{N1}(z)]$

Observe: If we have more than 1 flter, we can collect their polyphase vectors into **polyphase matrices.**

```
In [8]: import scipy.signal
        import numpy as np
        from sound import *
```

up-sample the signal y by a factor of N=2 and low-pass flter it with the flter

```
In [9]: h = np.array([0.5, 1, 1, 0.5])
```

as in the previous example. Again we obtain the flters polyphase components as,

```
In [10]: h0 = h[0::2]
         h1 = h[1::2]
```

Now we can use these polyphase components to flter at the lower sampling rate to obtain the polyphase components of the fltered and upsampled signal y0 and y1,

```
In [11]: y0 = scipy.signal.lfilter(h0,1,y)
         y1 = scipy.signal.lfilter(h1,1,y)
```

The complete up-sampling the signal is then obtained from its 2 polyphase components, performing our deblocking,

```
In [12]: L = np.max([len(y0), len(y1)])
         yu = np.zeros(2*L)
         yu[0::2] = y0
         yu[1::2] = y1
```

Where now the signal yu is the same as if we had frst upsampled and then fltered the signal!

Now listen to the up-sampled signal:

```
In [13]: sound(yu/2,fs)

* done
```