1 Lecture Outline

Reading: Chapter 4 FIR Filtering and Convolution

- Sample processing methods (4.2)
- Delay lines (4.2.1)
- FIR filtering in direct form (4.2.2)

2 Sample Processing Methods (4.2)

We now discuss formulations of FIR filters that operate on a sample-by-sample basis. As we mentioned earlier, such methods are convenient for real-time applications that require the continuous processing of the incoming input. Sample processing algorithms are closely related to block diagram realizations of the I/O filtering equations. A block diagram is a mechanization of the I/O equation in terms of the three basic building blocks: adders, multipliers, and delays, shown in Fig. 4.2.1.

Figure 1: Orfanidis p. 147 Fig. 4.2.1

3 Delay Lines

The single delay has the I/O relation

$$y(n) = x(n-1). \tag{1}$$

It can be thought of as a *register* or *memory* holding the *previous* input sample $x(n-1)$. At each time instant n, two steps must be carried out: (a) the current content $x(n-1)$ is clocked out to the output and (b) the current input $x(n)$ gets stored in memory, where it will be held for one sampling instant and become the output at the *next* time $n+1$.

We can think of the contents of this memory as the *internal state* of the filter at sample index n

$$w_1(n) = x(n-1), \quad \text{internal state at time } n. \tag{2}$$

The internal state is updated at the next sample index

$$w_1(n+1) = x(n), \quad \text{internal state at time } n+1. \tag{3}$$

The output and state update equations for the delay are then

$$y(n) = w_1(n) \tag{4}$$

Figure 2: Orfanidis p. 147 Fig. 4.2.2

$$w_1(n+1) = x(n) \tag{5}$$

and we initialize the internal state as

$$w_1(0) = 0. \tag{6}$$

The effect of course is a right shift by one sample of the input sequence

$$[x_0,\ x_1,\ x_2,\ \ldots] \quad \rightarrow \quad [0,\ x_0,\ x_1,\ \ldots]\,. \tag{7}$$

In pseudo-code, we have

```
for each input sample x do:
   y := w1
   w1 := x
```

Example: In the case of a delay of two samples we have

$$y(n) = x(n-2). \tag{8}$$

Figure 3: Orfanidis p. 148 Fig. 4.2.3

For this delay we require two words of memory to store the previous two samples (two states)

$$y(n) = w_2(n) \tag{9}$$

$$w_2(n+1) = w_1(n) \tag{10}$$

$$w_1(n+1) = x(n). \tag{11}$$

In pseudo-code, we have

```
for each input sample x do:
   y := w2
   w2 := w1
   w1 := x
```

Example: For the general case of a delay by D samples we have (introducing $w_0(n)$ for convenience)

$$y(n) = w_D(n) \tag{12}$$

$$w_0(n) = x(n) \tag{13}$$

$$w_i(n+1) = w_{i-1}(n), \quad i = D,\ D-1,\ ,\ldots,\ 2,\ 1. \tag{14}$$

```
for each input sample x do:
   y := wD
   w0 := x
   for i = D, D-1, ..., 1 do:
      wi := w_i-1
```

Figure 4: Orfanidis p. 151 Fig. 4.2.5

4 FIR Filtering in Direct Form (4.2.2)

Example: The direct form realization of a 3rd order FIR filter is shown in Fig. 4.2.6 and Fig. 4.2.7.

Figure 5: Orfanidis p. 153 Fig. 4.2.6 and Fig. 4.2.7.

In pseudo-code, we have

```
for each input sample x do:
   w0 := x
   y := h0w0 + h1w1 + h2w2 + h3w3
   w3 = w2
   w2 = w1
   w1 = w0
```

Example: The direct form realization of an Mth order FIR filter is shown below.

In pseudo-code, we have

```
for each input sample x do:
   w0 := x
   y := h0w0 + h1w1 + ... + hMwM
   for i = M, M-1, ..., 1 do:
      wi := w_i-1
```

Figure 6: Mth order FIR filter