Note Set #5

- Difference Equations – Implementation of DT Systems
- Reading Assignment: Sect. 2.5 of Proakis & Manolakis
Here we will take a first look at how to implement (i.e., build) DT systems… we’ll look at this in much more detail later but what we see here will be our foundation.

We’ve already seen the following structure for an FIR filter based on interpreting its convolution equation in terms of block diagram elements:

Now we want to look at how to do a similar thing for a DT system described by a general difference equation. We’ll start by motivating this with a simple first-order difference equation.
Block Diagrams for First-Order System

\[ y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1] \]

From this we see that we can use one delay to get \( y[n-1] \) and a second delay to get \( x[n-1] \).

Then we add the various terms together to create \( y[n] \).

This form… that has separate delays for the input and for the output…. is called **Direct-Form I**

It is possible to reduce the number of delays with a “trick”.

Remember… each delay is essentially a memory location so this will reduce the HW needed.
Trick to Get Direct-Form II – which has reduced number of delays

For LTI systems we can interchange their order without changing their overall mathematical result. So…

Direct-Form I

Change order

Same value… so only one delay is needed!

Direct-Form II

Note that nowhere in this do we have the delays of input or output!
We can now deduce the structure for a Direct Form II implementation

\[ y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1] \]
Block Diagrams for Higher-Order System

We can do the exact same process for a higher-order Difference Eq:

\[ y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \]

Note: \( M \) and \( N \) may not be equal… so different # of delays in each side
\[ y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \]
Recursive vs Non-Recursive

Note that the $a_k$ coefficients are responsible for imparting recursion (feeding back past outputs...).

The $b_k$ coefficients do not impart any recursive nature... they constitute the non-recursive part of the system.
Purely Recursive

\[ y[n] = - \sum_{k=1}^{N} a_k y[n-k] + b_0 x[n] \]
Purely Non-Recursive ("Moving Average")

\[ y[n] = \sum_{k=0}^{M} b_k x[n-k] \]

Same thing we already saw for FIR…. So FIR can always be implemented using a purely NON-recursive structure!!

But… any FIR system can be implemented using recursion (though not purely recursive).
Example of FIR Implemented using Recursion

\[ y[n] = \frac{1}{M+1} \sum_{k=0}^{M} x[n-k] \]

i.e., \( b_k = \frac{1}{M+1} \)

\( n, n-1, \ldots, n-M \)

Nonrecursive Form

Can rewrite this system equation as

\[ y[n] = \frac{1}{M+1} \sum_{k=0}^{M} x[n-1-k] + \frac{1}{M+1} \left( x[n] - x[n-(1+M)] \right) \]

\( n-1, \ldots, n-(1+M) \)

Recursive!

This is actually \( y[n-1] \)
$y[n] = \frac{1}{M + 1} \sum_{k=0}^{M} x[n - k]$  

$y[n] = y[n-1] + \frac{1}{M + 1} \left( x[n] - x[n-(1+M)] \right)$

Summarizing:
- **FIR & IIR** describe fundamental characteristic of the system’s impulse response regardless of how it is implemented.
- **Recursive and Non-Recursive** describe the specific structure used to implement the system.
Discrete-Time System Relationships

**Time Domain**

Difference Equation

\[ y[n] = -\sum_{i=1}^{p} a_i y[n-i] + \sum_{i=0}^{q} b_i x[n-i] \]

Impulse Response

\[ h[n] \]

**Z / Freq Domain**

Transfer Function

\[ H^z(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \]

Pole/Zero Diagram

Roots

Unit Circle

\[ z = e^{j\omega} \]

Frequency Response

\[ H^f(\omega) = H^z(z) \bigg|_{z = e^{j\omega}} \]