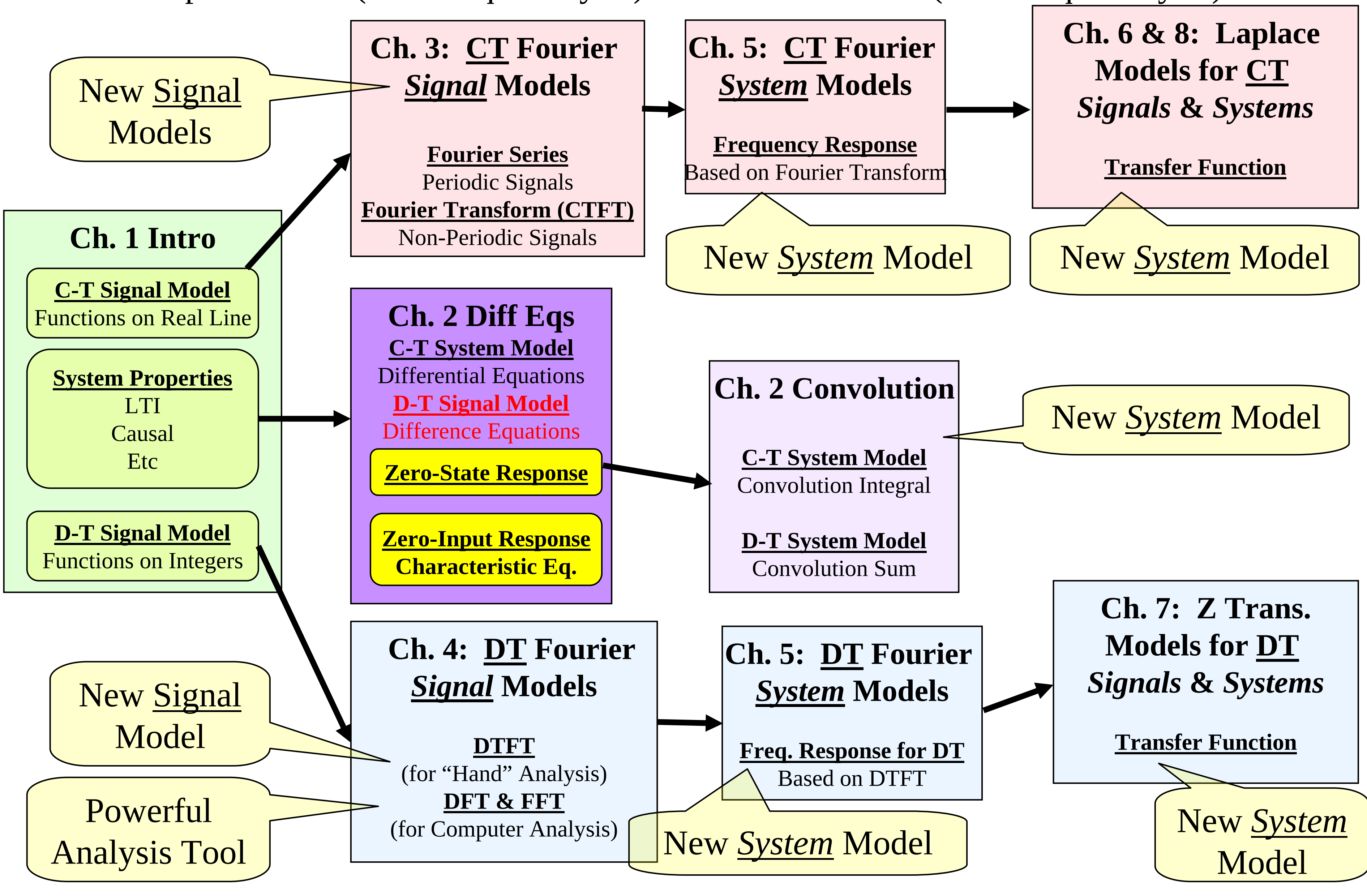
EECE 301
Signals & Systems
Prof. Mark Fowler

Note Set #7

- D-T Systems: Recursive Solution of Difference Equations
- Reading Assignment: Section 2.3 of Kamen and Heck

Course Flow Diagram

The arrows here show conceptual flow between ideas. Note the parallel structure between the pink blocks (C-T Freq. Analysis) and the blue blocks (D-T Freq. Analysis).



D-T System Models

We saw that Differential Equations model C-T systems…

D-T systems are "modeled" by Difference Equations.

The quotes are used here because we aren't really modeling some existing system with difference equations but rather building a desired system with difference equations. So in that sense, difference equations aren't just models they *are* the system.

A general N^{th} order Difference Equations looks like this:

$$y[n] + a_1 y[n-1] + \ldots + a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \ldots + b_M x[n-M]$$

Most "Advanced" Output Sample

Least "Advanced" Output Sample

The difference between these two index values is the "order" of the difference eq.
Here we have: $n - (n - N) = N$

Solving Difference Equations

Although Difference Equations are quite different from Differential Equations, the methods for solving them are remarkably similar. We'll study such analytic methods later.

Here we'll look at a numerical way to solve Difference Equations. This method is called Recursion… and it is actually used to implement (or build) many D-T systems, which is the main advantage of the recursive method.

The disadvantage of the recursive method is that it doesn't provide a so-called "closed-form" solution… in other words, you don't get an equation that describes the output (you get a finite-duration sequence of numbers that shows part of the output).

Later we'll see how to get "closed-form" solutions… such solutions give engineers keen insight needed to perform design and analysis tasks.

Solution by Recursion

But, for computer processing it is possible to recursively solve (i.e. compute) a numerical solution. In fact, this is how D-T systems are implemented (i.e. built!)

We can re-write any linear, constant-coefficient difference equation in "recursive form". Here is the form we've already seen for an N^{th} order difference:

$$y[n] + a_1 y[n-1] + \ldots + a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \ldots + b_M x[n-M]$$

Re-Write As:

$$y[n] + \sum_{i=1}^{N} a_i y[n-i] = \sum_{i=0}^{M} b_i x[n-i]$$

Now… isolating the $y[n]$ term gives the "Recursive Form":

$$y[n] = -\sum_{i=1}^{N} a_i y[n-i] + \sum_{i=0}^{M} b_i x[n-i]$$

"current" output value to be computed

Some "past" output values, with values already known

current & past input values already "received"

The key to Recursive Form is that you have the current output $y[n]$ in terms of *past* outputs $y[n - i]$

Note: sometimes it is necessary to re-index a difference equation using $n+k \to n$ to get this form… as shown below.

Here is a slightly different form… but it is still a difference equation:

$$y[n+2] - 1.5y[n+1] + y[n] = 2x[n]$$

If you isolate $y[n]$ here you will get the current output value in terms of future output values (Try It!)… We don't want that!

So… in general we start with the "Most Advanced" output sample… here it is $y[n+2]$… and re-index it to get only n (of course we also have to re-index everything else in the equation to maintain an equation):

So here we need to subtract 2 from each sample argument:

$$y[n] - 1.5y[n-1] + y[n-2] = 2x[n-2]$$

Now we can put this into recursive form as before.

Ex: Solve this difference equation recursively

$$y[n]-1.5y[n-1]+y[n-2]=2x[n-2]$$

For $x[n]=u[n]$ unit step

And ICs of: $\begin{cases} y[-2]=2 \\ y[-1]=1 \end{cases}$

Note: You need N "past" values as IC's to solve an N^{th} order Difference Equation

Recursive Form: $y[n]=1.5y[n-1]-y[n-2]+2x[n-2]$

n	$x[n]=u[n]$	$y[n]$
-2	0	2
-1	0	1
0	1	$1.5\cdot 1-2+2\cdot 0=-0.5$
1	1	$1.5\cdot(-0.5)-1+2\cdot 0=-1.75$
2	1	-.0125
3	1	3.563

1st: Fill in Input (Unit Step here)

2nd: Put IC's Here

3rd: Compute n=0 Output
$y[0]=1.5y[-1]-y[-2]+2x[-2]$

4th: Compute n=1 Output
$y[1]=1.5y[0]-y[-1]+2x[-1]$

Etc.

We can write a simple matlab routine to implement this difference equation $y[n] = 1.5y[n-1] - y[n-2] + 2x[n-2]$

There is a more general version of this code on the Book's web page.

```
function y = recur_2(x,y_ics);

y(1) = y_ics(1);
y(2) = y_ics(2);

for k=3:(length(x)+2)
        y(k)=1.5*y(k-1)-y(k-2)+2*x(k-2);
end
```

x is a vector of input samples
(from our table-based solution we see that we need the vector x to start at *n* = -2)
y_ics is 1x2 vector holding the 2 ICs

y will be the returned vector holding the output samples

Write the ICs into the output vector's first two positions

Each time through the for-loop we compute the output value according to the recursive form of the difference equation

```
x = [0 0 ones(1,20)];
```

```
stem(-2:(length(y)-3),y)
```

The trickiest part of getting this code right is getting the indexing right!!!

Mathematical indexing used in difference equations is “zero-origin” and allows negative indices.

Matlab indexing is “one-origin” and does NOT allow negative indexing.

The “k” in the code is related to the math index *n* according to: k = n+3

Thus, when we first enter the loop we are computing for k=3 or *n* = 0

```
function y = recur_2(x,y_ics);

y(1) = y_ics(1);
y(2) = y_ics(2);

for k=3:(length(x)+2)
        y(k)=1.5*y(k-1)-y(k-2)+2*x(k-2)
end
```

Store *y*[-2] in k=1 position of vector
Store *y*[-1] in k=2 position of vector

We must continue the loop until the last input value is used… since we use x(k-2) in the recursion we need to stop our for-loop at length(x)+2.

We already have filled the first two elements of the output vector so we put y[0] into the 3^{rd} position, etc.

That way… when we go through the last loop (i.e., k = length(x)+2) we’ll index x using k-2 = length(x)… which grabs the <u>last</u> element in the input vector x

We could use these ideas to implement this D-T system on a computer… although for real-time operation we would not use matlab, we likely would write the code using C or assembly language.

Also… we probably wouldn't implement this on a general microprocessor like those used in desktop or laptop computers. We would implement it in a microcontroller for simple applications but for high-performance signal processing applications (like for radar and sonar, etc.) we would use a special DSP microprocessor.

Web Link to Extra Info on DSP Processors

This is a S/W implementation of the D-T system…. It is also possible to build dedicated digital H/W to implement it.

Web Link to Example of Dedicated H/W D-T System