TECH 350: DSP

Class IV: Properties of Systems, Signal Flow Diagrams, Filters

Properties of DSP Systems We'll Start With

Deterministic

easily predictable under reasonably simple circumstances

Linear

the sum of their effects is the effect of their sums / the output due to a sum of input signals equals the sum of outputs due to each signal alone

Time Invariant

same output, regardless of when input occurs

LTI Systems = Linear, Time Invariant Systems

Other Important Properties

Causal

output is a function of past and current inputs (not future inputs)

(only matters for real-time applications; offline (e.g. look-ahead), not a problem)

Invertible

can determine input if given output (e.g. deconvolution)

Stable

impulse response is non-infinite (will get to this soon);

FIR = stable, IIR = maybe stable, but not guaranteed

Static

output is only a function of current input (as opposed to dynamic, which requires memory)

Representing DSP Systems

Signal Flow (or Block) Diagrams

$$\begin{tikzcd} x(n) \arrow[r] & \boxed{\quad S \quad} \arrow[r] & y(n) \end{tikzcd}$$

(Difference) Equations

$$y = S(x)$$

Code

```
audioOutput = applyS(audioInput);
#audioInput is a vector corresponding to samples in audio file
```

Signal Flow Diagrams

Components

addition/subtraction

$$\begin{tikzcd} x_t \arrow[r] & \oplus \arrow[r] & x_t + y_t \\ & y_t \arrow[u] & \end{tikzcd}$$

multiplication
(by a constant only!)

$$\begin{tikzcd} x_t \arrow[r, "a"'] & ax_t \end{tikzcd}$$

delay

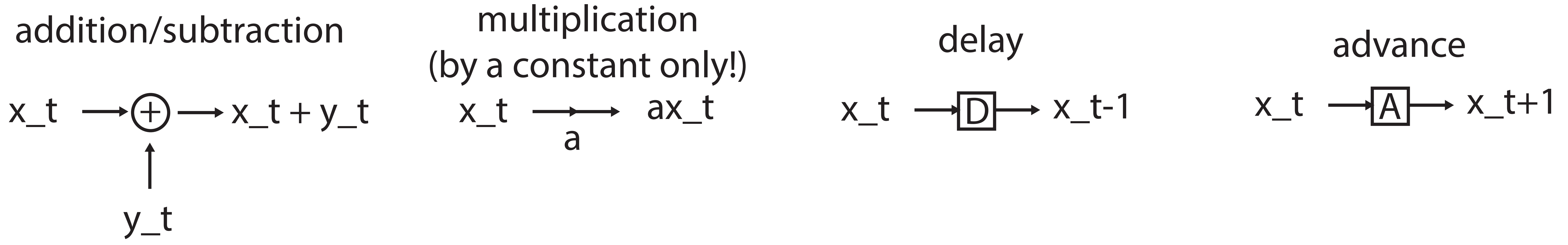
$$\begin{tikzcd} x_t \arrow[r] & \boxed{D} \arrow[r] & x_{t-1} \end{tikzcd}$$

advance

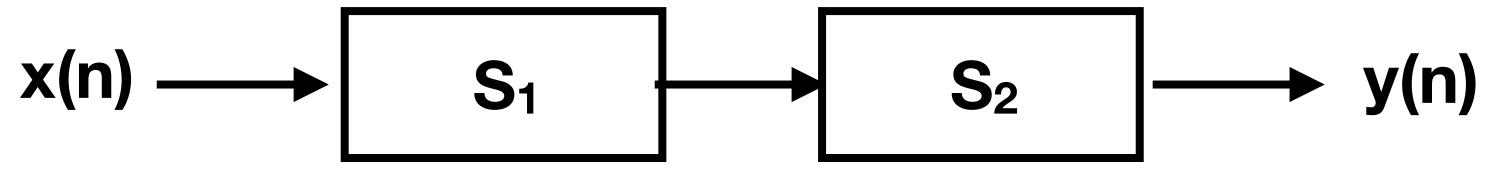
$$\begin{tikzcd} x_t \arrow[r] & \boxed{A} \arrow[r] & x_{t+1} \end{tikzcd}$$

Signal Flow Diagrams

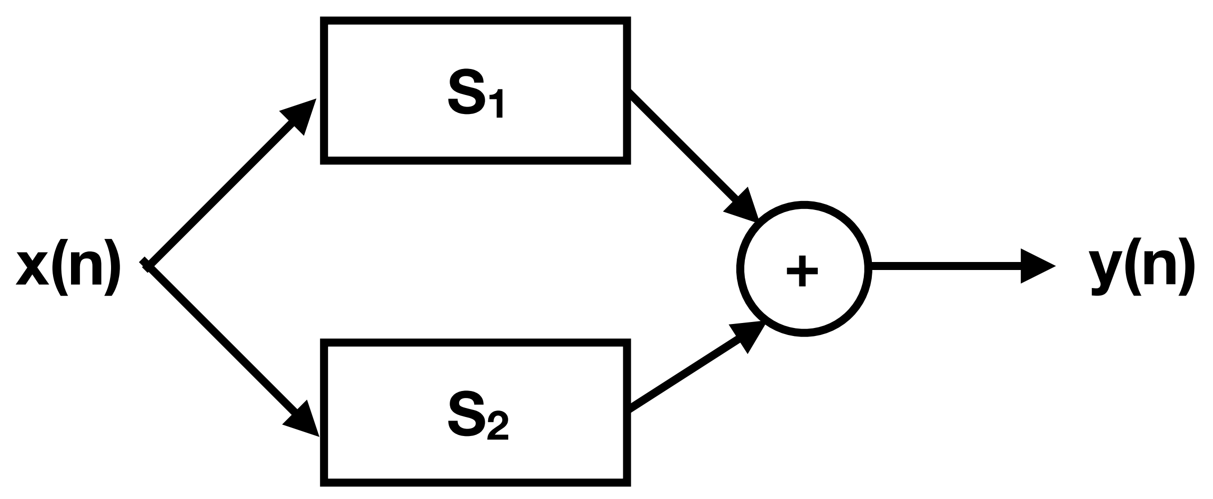
Components



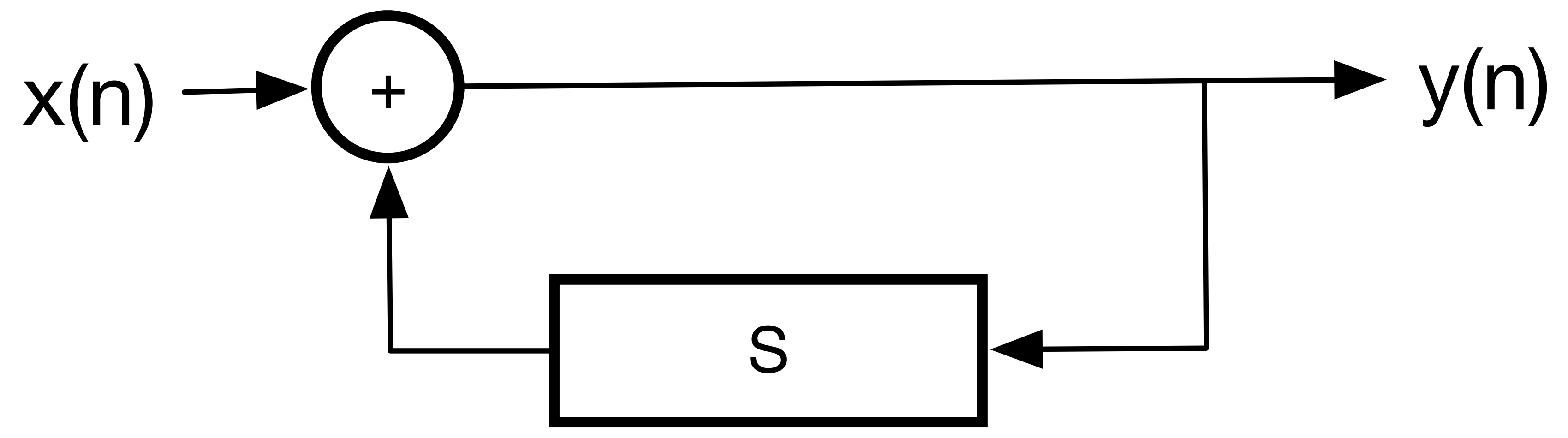
Organizational Schemes



series



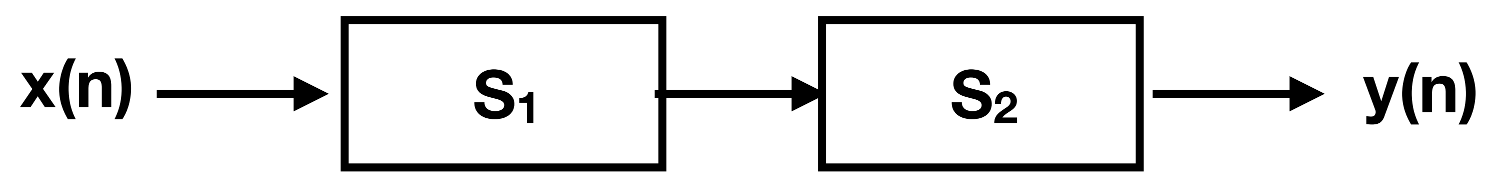
parallel



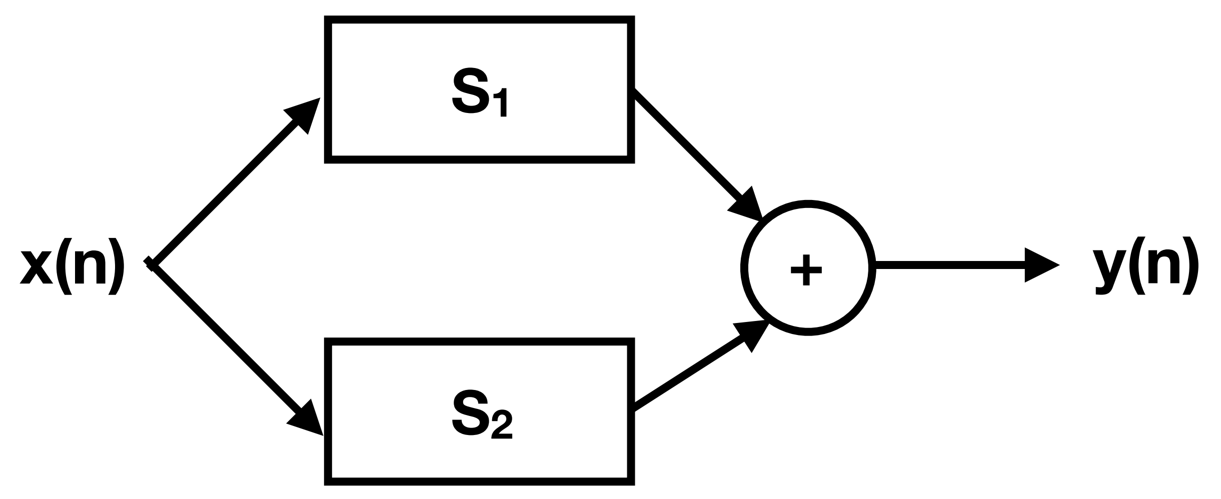
feedback

Signal Flow Diagram <-> Equation

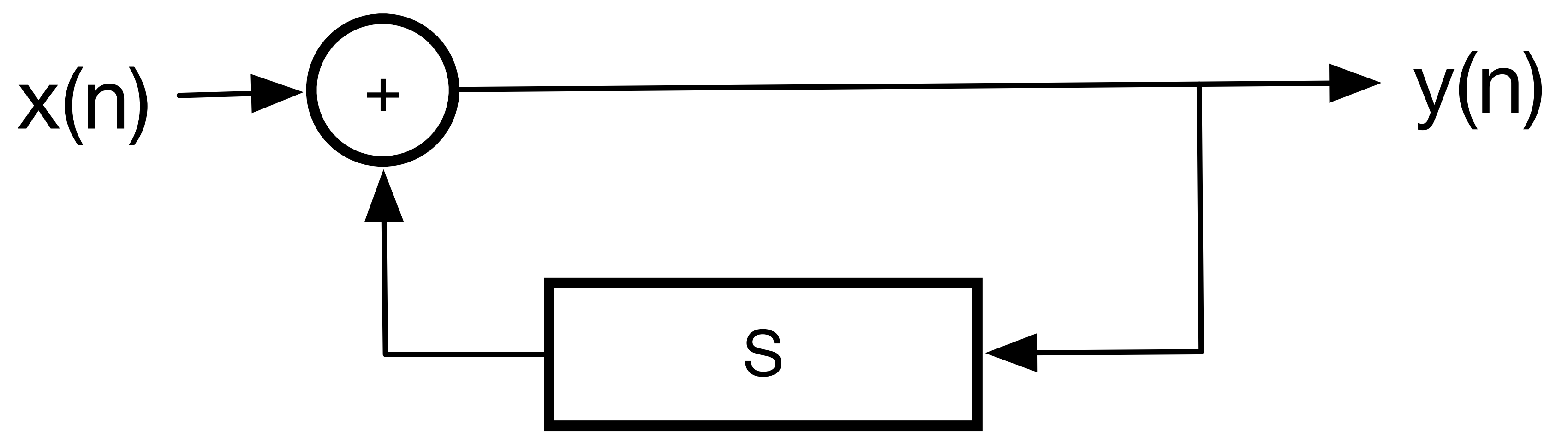
Organizational Schemes



series



parallel

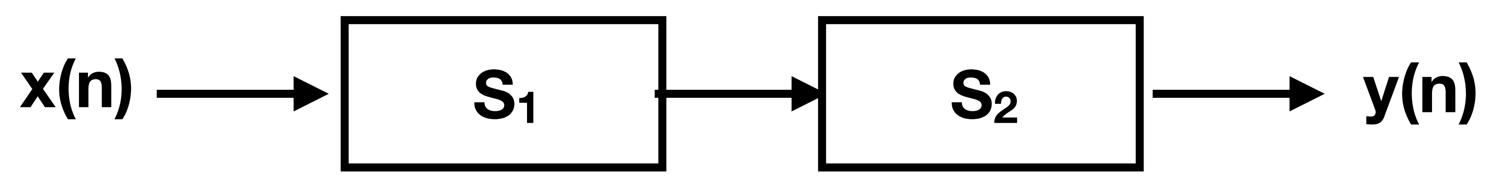


feedback

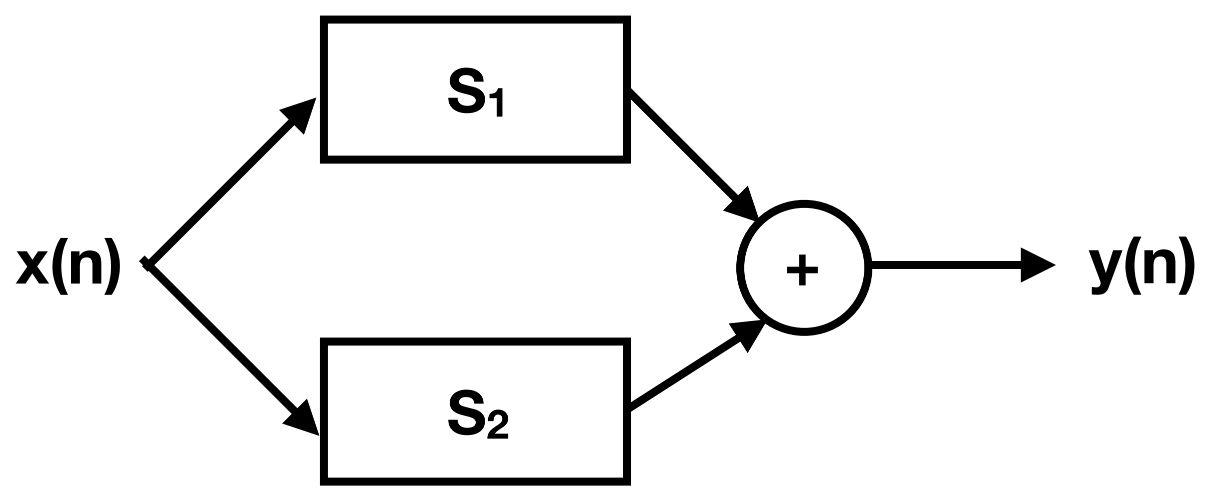
Conversions to Difference Equations (a bit Pseudo-Code-y)

Signal Flow Diagram <-> Equation

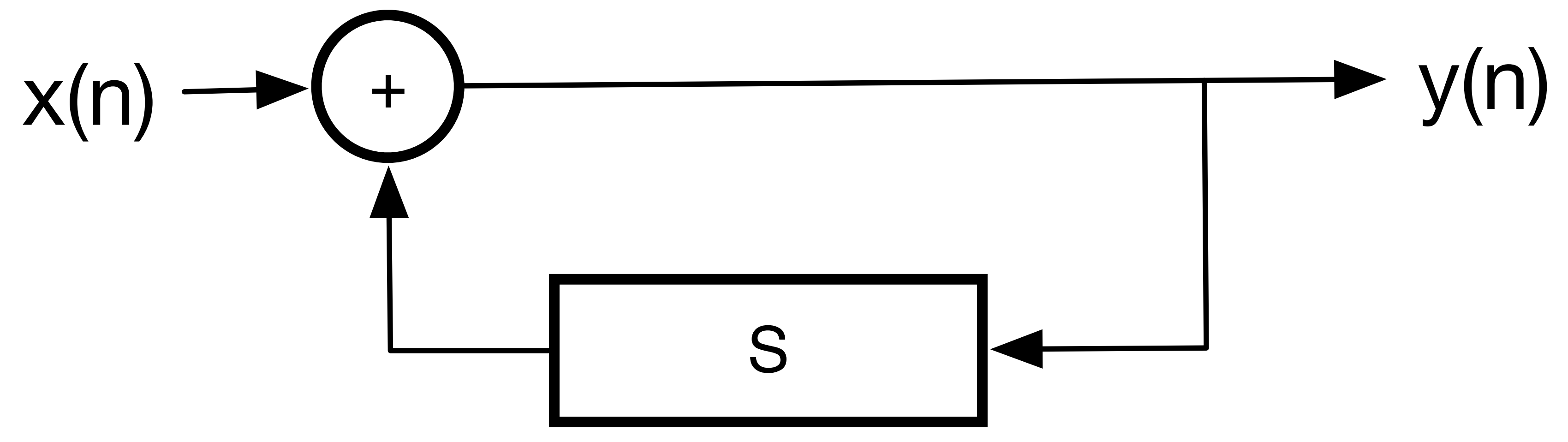
Organizational Schemes



series



parallel



feedback

Conversions to Difference Equations (a bit Pseudo-Code-y)

$y = S_2(S_1(x))$

$y = S_1(x) + S_2(x)$

$y = x + S(y)$

Example 1

System: y(n) = S(x(n)), where S: output = input * 0.5 + 0.5

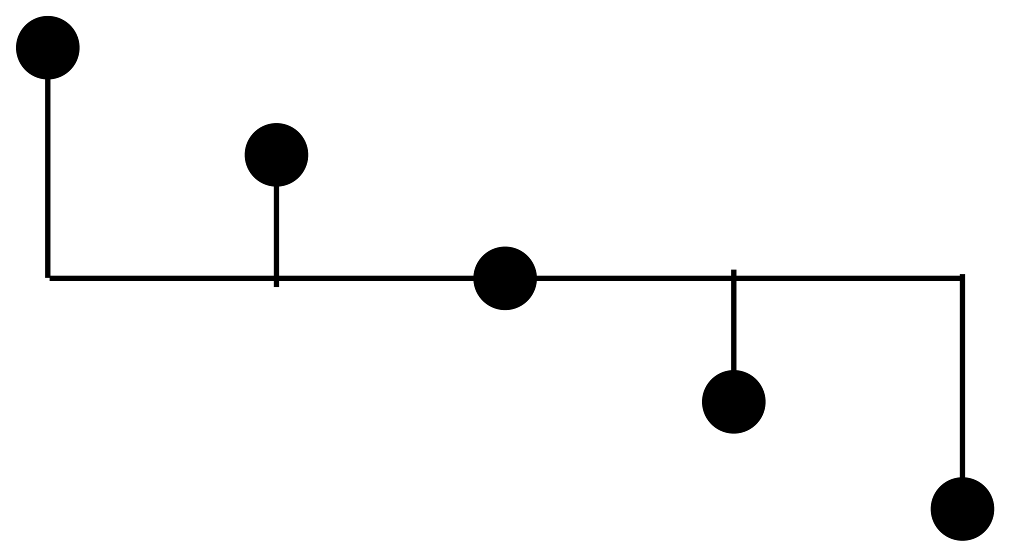
where n = 0, 1, 2, 3, . . . are indices to input X and output Y

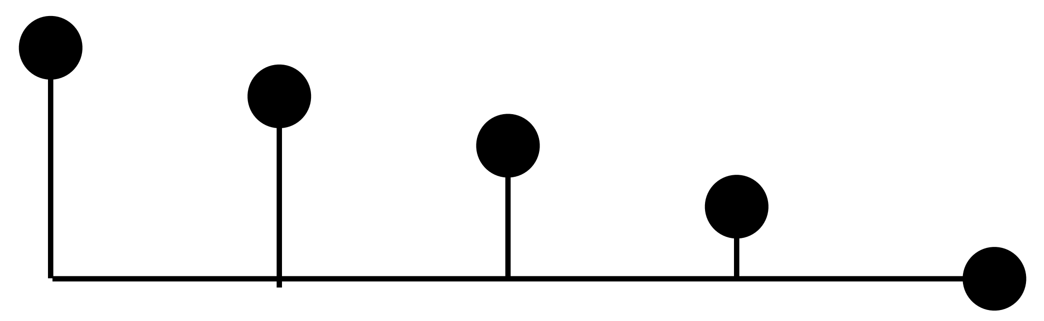
$$\begin{tikzcd} x(n) \arrow[r] & \boxed{S} \arrow[r] & y(n) \end{tikzcd}$$

$$\begin{tikzcd} x(n) \arrow[r, "0.5"'] & \oplus \arrow[r] & y(n) \\ & 0.5 \arrow[u] & \end{tikzcd}$$

Input: 1 0.5 0 -0.5 -1

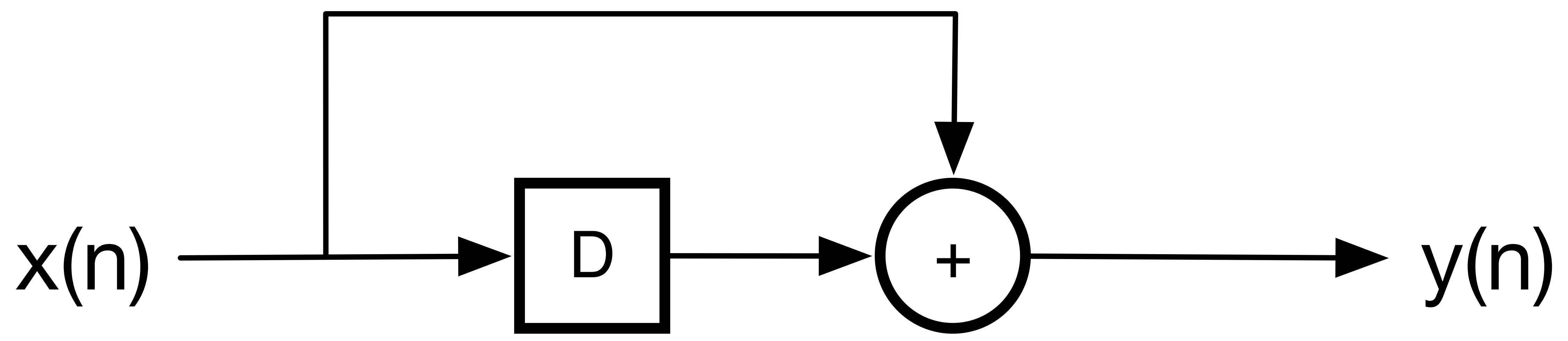
Output: 1 0.75 0.5 0.25 0





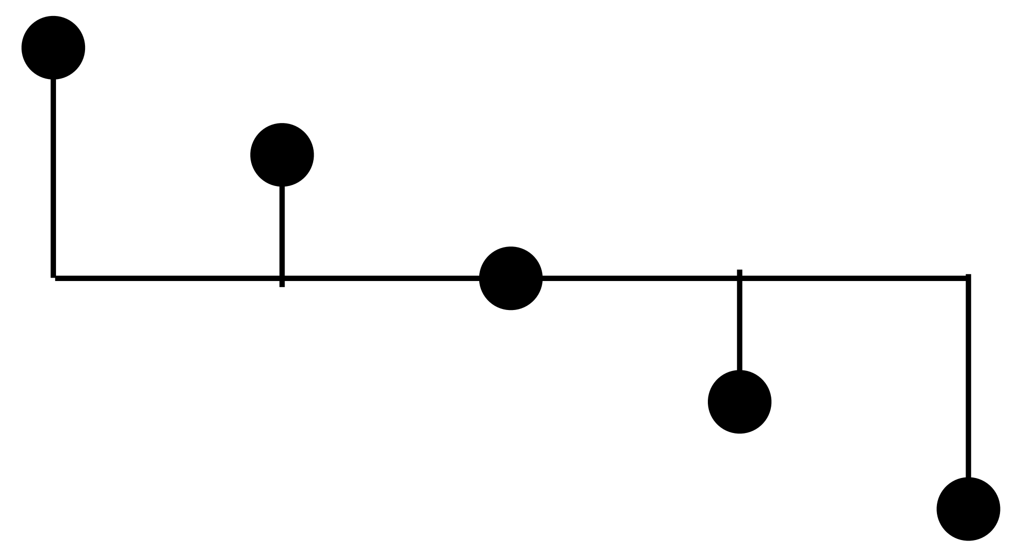
Example 2

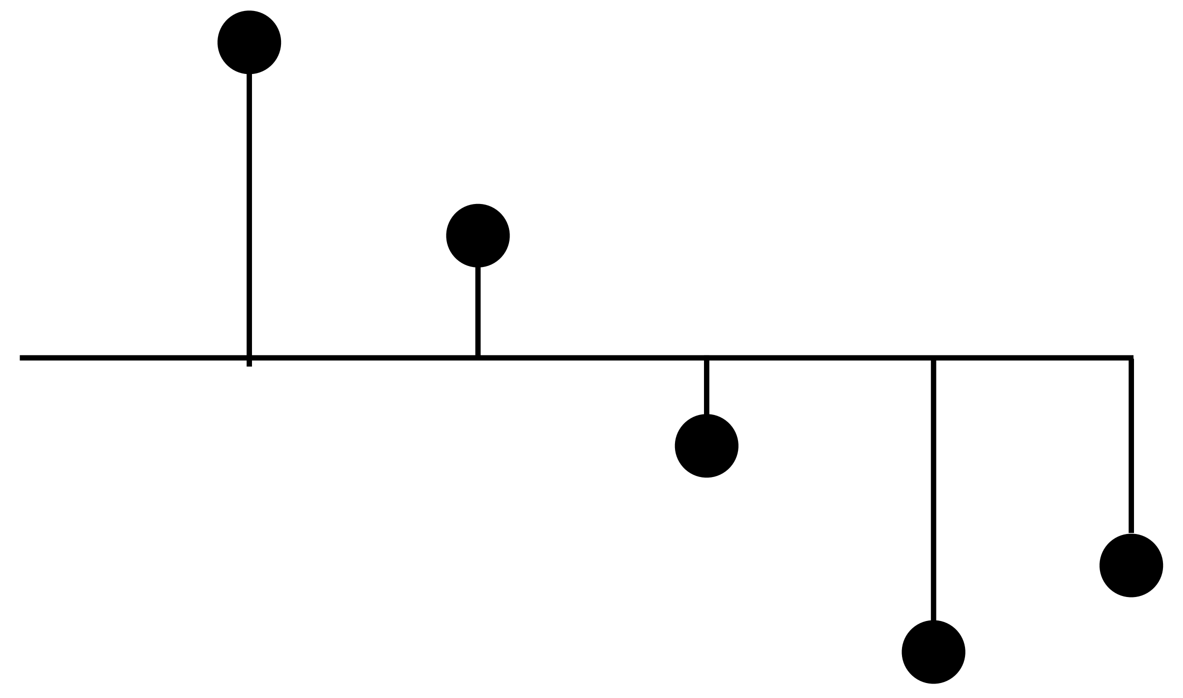
System: y(n) = x(n) + x(n-1), where n = 0, 1, 2, 3, . . . are indices to input X and output Y



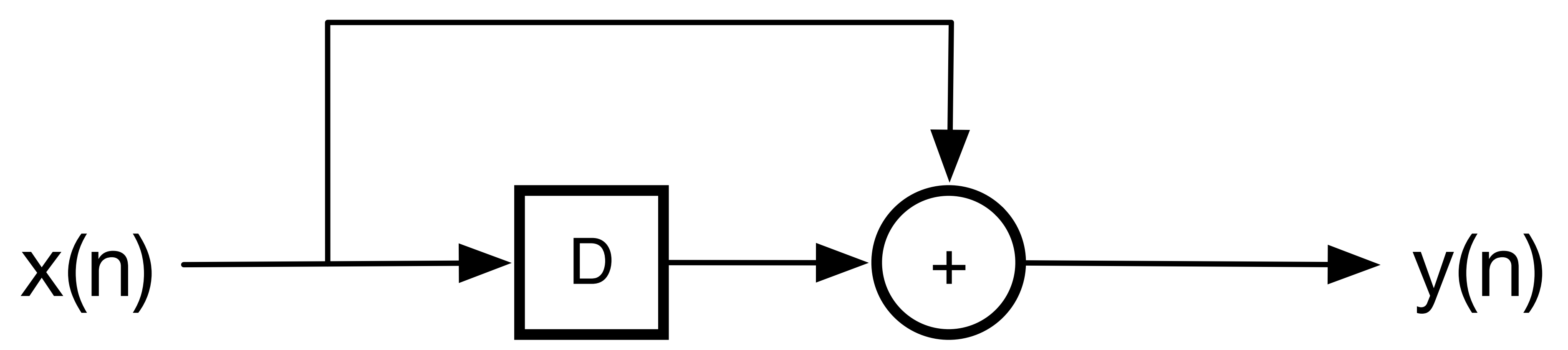
Input: 1 0.5 0 -0.5 -1

Output: _ 1.5 0.5 -0.5 -1.5 (-1)





Example 2 = Simplest Lowpass Filter



$y(n) = x(n) + x(n-1)$

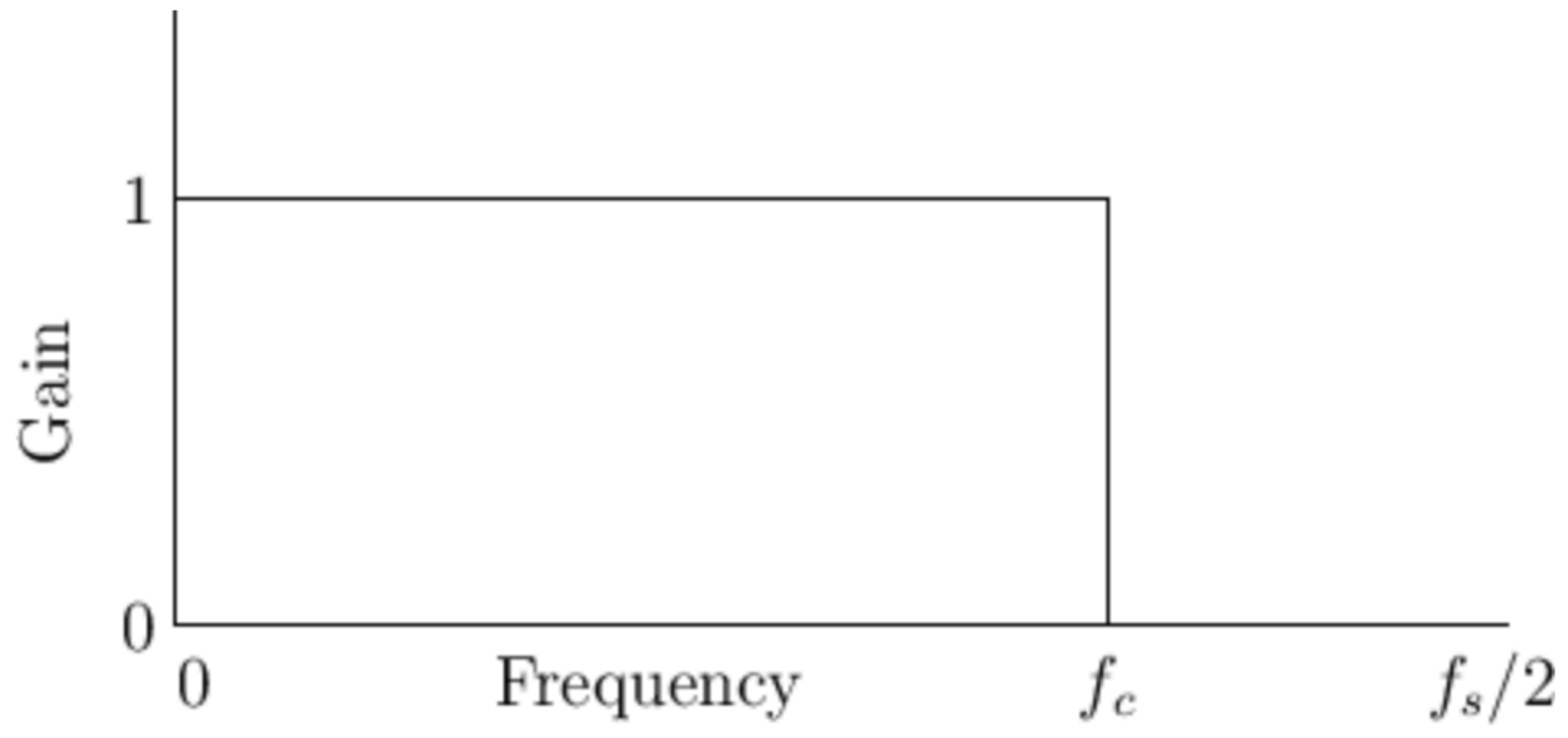


Figure 1.1: Amplitude response (gain versus frequency) specification for the ideal low-pass filter.

Code on next page…

Reminder: what a low-pass filter is (from JOS)

The Simplest Lowpass Filter

Matlab Implementation 1:

```
N = 10; # length of text input
x = 1:N; # integer ramp
y = zeros(1,N) # our output
for n=2:N # for loop from second index to end
    y(n) = x(n) + x(n-1); # our equation
end # end for loop
```

The Simplest Lowpass Filter

Matlab Implementation 2:

```
N = 10; # length of text input

x = 1:N; # integer ramp

x_shifted = [0 x(1:N-1)]; # offset to simulate delay

y = x + x_shifted; # add x and “delayed” version of x to one another
```

The Simplest Lowpass Filter

Matlab Implementation 3:

```
N = 10;

x = 1:N;

y = filter([1,1],1,x);

# these define feedback and
# feedforward filter coefficients,
# which we’ll get to soon!
```

The Simplest Lowpass Filter

Some Questions:

How far away from ideal (‘perfect’) is this filter?

What we want to know is this filter's ***frequency response***

How can we test this?

First, by testing it at **each frequency**, what is called ***sine-wave analysis***

Sine-Wave Analysis

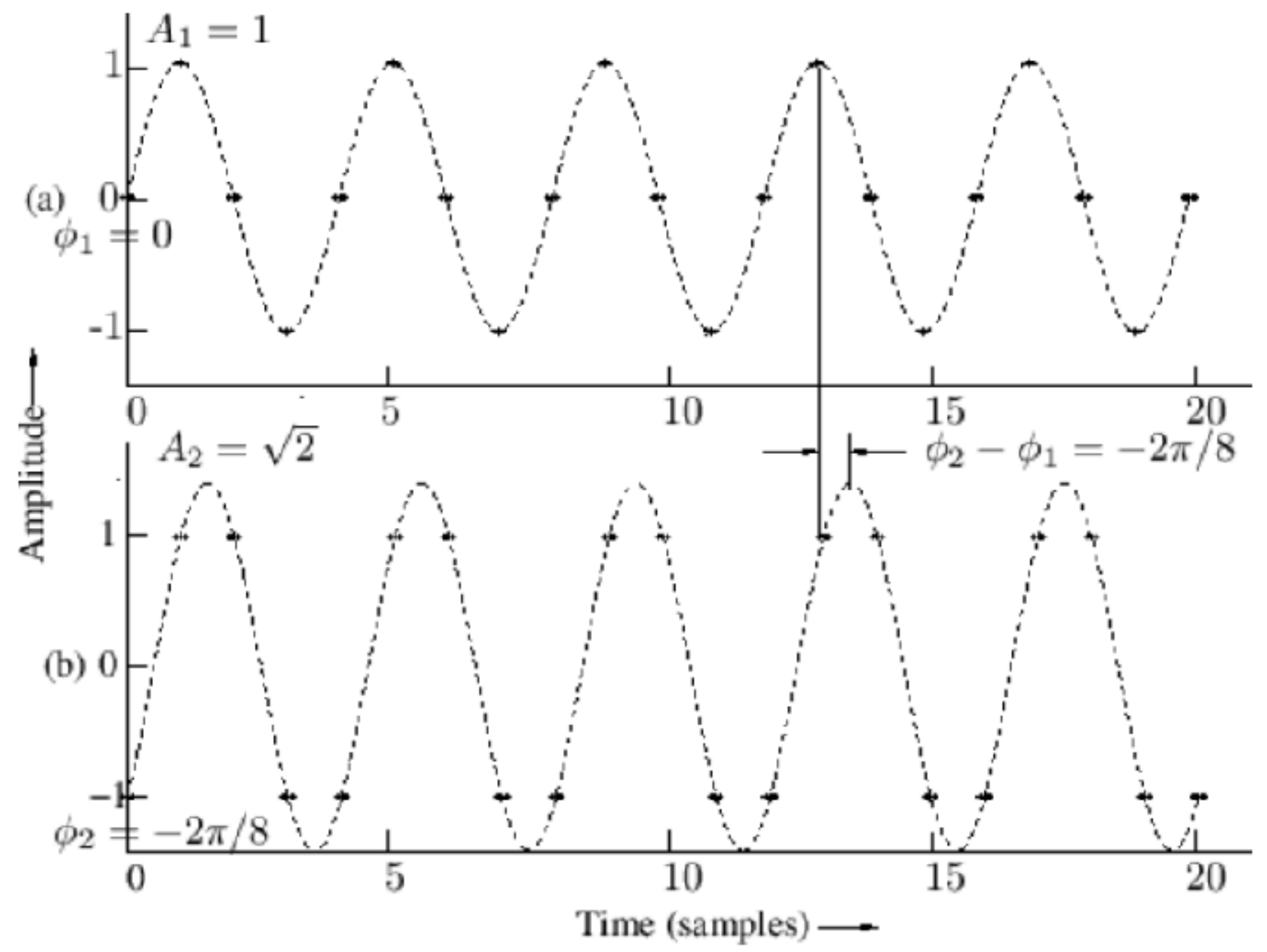


Figure 1.6: Input and output signals for the filter $y(n) = x(n) + x(n-1)$. (a) Input sinusoid $x(n) = A_1 \sin(2\pi f n T + \phi_1)$ at amplitude $A_1 = 1$, frequency $f = f_s/4$, and phase $\phi_1 = 0$. (b) Output sinusoid $y(n) = A_2 \sin(2\pi f n T + \phi_2)$ at amplitude $A_2 = 1.414$, frequency $f = f_s/4$, and phase $\phi_2 = -\pi/4$.

From JOS

What do we learn from this?

Amplitude Response =

how much quieter or louder the output sine is

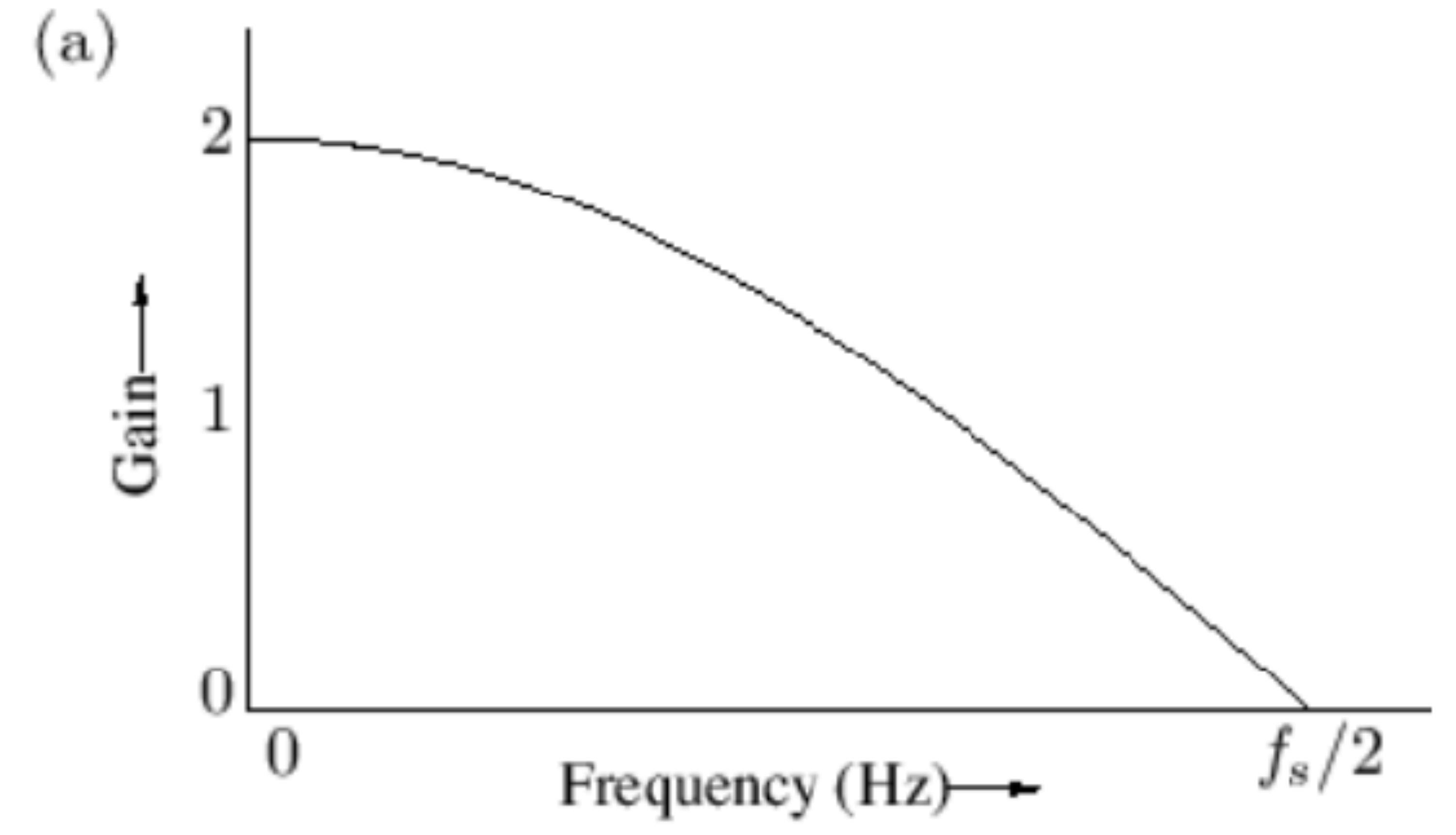
Phase Response =

how much out of phase the output sine is

Amplitude Response and Phase Response together =

Frequency Response

More Frequency Response Analysis



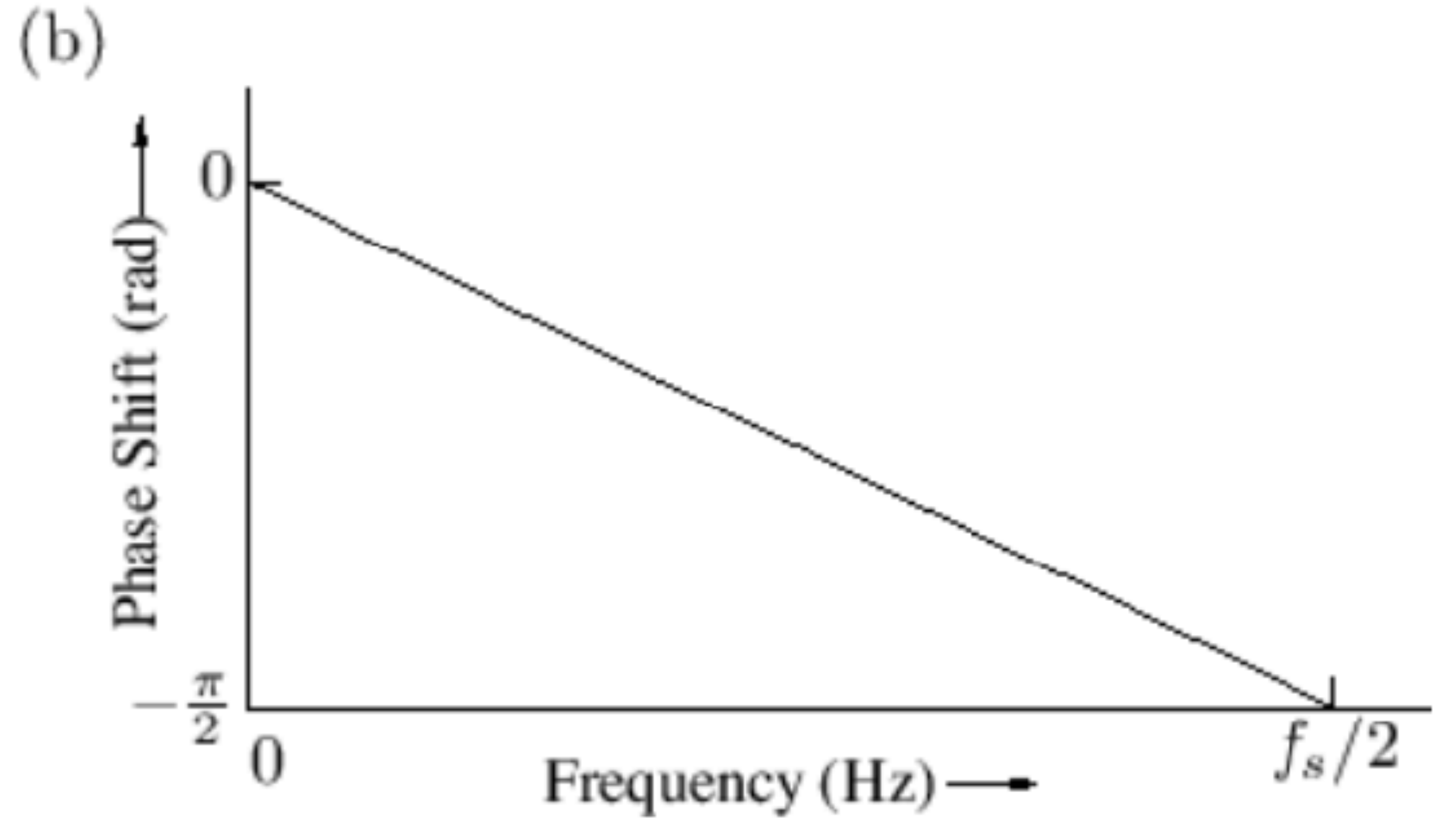


Figure 1.7: Frequency response for the filter $y(n) = x(n) + x(n-1)$. (a) Amplitude response. (b) Phase response.

Sine-Wave Analysis Works…

…but isn't particularly useful

Alternatives include:

using trigonometry and delightful simplifications like this…

$$
\begin{aligned}
y(n) &= \cos(\omega nT) + \cos[\omega(n-1)T] \\
&= \cos(\omega nT) + \cos(\omega nT)\cos(-\omega T) - \sin(\omega nT)\sin(-\omega T) \\
&= \cos(\omega nT) + \cos(\omega nT)\cos(\omega T) + \sin(\omega nT)\sin(\omega T) \\
&= [1 + \cos(\omega T)]\cos(\omega nT) + \sin(\omega T)\sin(\omega nT) \\
&= a(\omega)\cos(\omega nT) + b(\omega)\sin(\omega nT)
\end{aligned}
$$

An Easier (?) Way

…or using complex sinusoids (AKA phasors), which are more advanced (requiring an understanding of complex numbers)

Who has encountered complex numbers (a + bi) before?

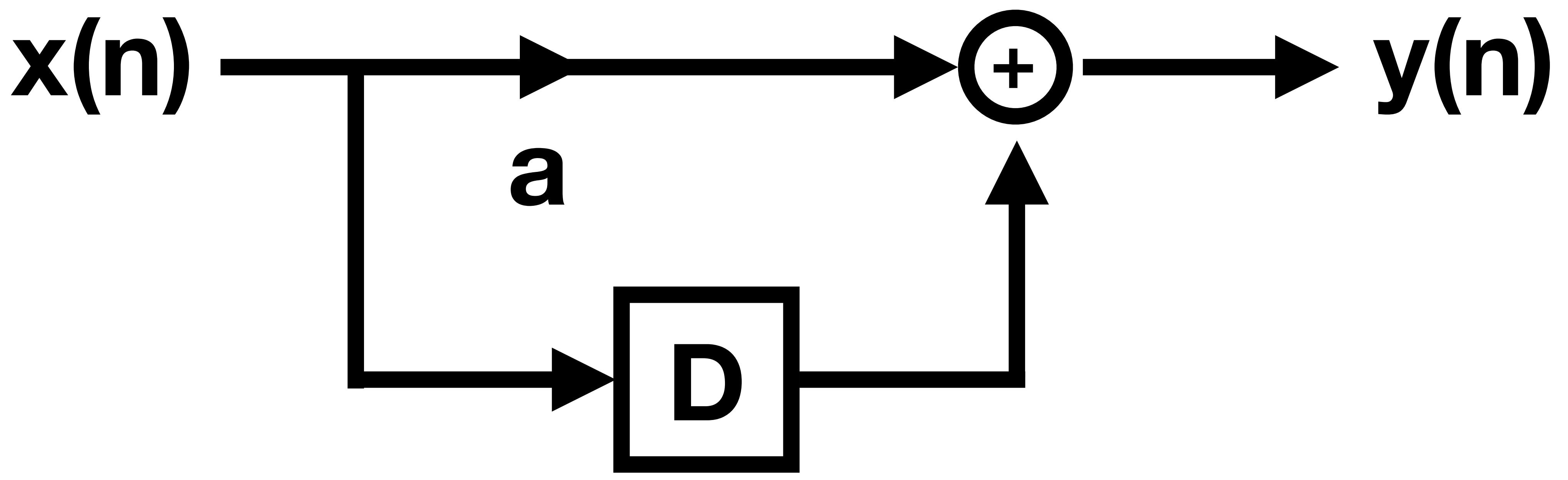
Next class we'll be discussing…

Other Filters (comb, allpass, biquad, chebyshev, butterworth), convolution, impulse responses

Practice: Diff. Equation -> Diagram

$$y(n) = a\,x(n) + b\,x(n-1)$$

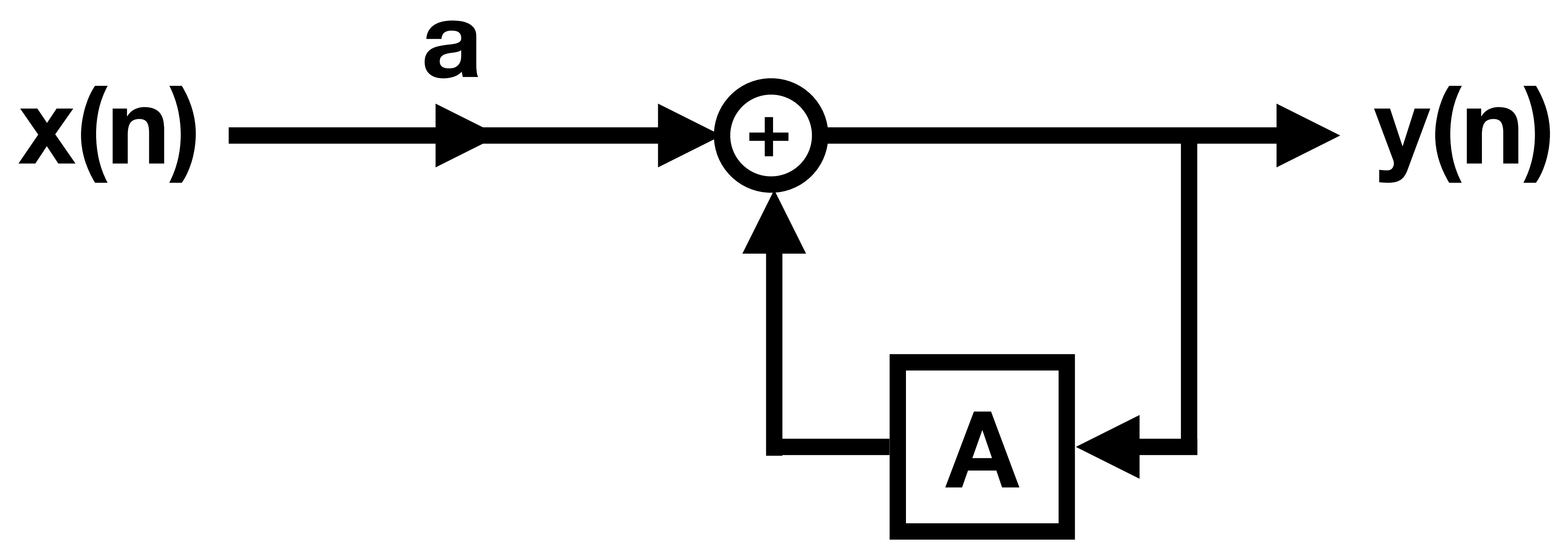
Practice: Diagram -> Diff. Equation



Practice: Diff. Equation -> Diagram

$$y(n) = a\,x(n) + b\,x(n-1) + c\,y(n-1)$$

Practice: Diagram -> Diff. Equation



Practice: Filtering Audio Files in Octave

```
fs = 44100; # define sample rate
[x, fs] = audioread(‘audio/suzanne.wav’); #read into a vector, x
sound(x,fs); # playback vector

### Do something here, getting vector y

audiowrite(‘myoutput.wav’,y,fs); # output to ‘myoutput.wav’
```

Mini-Assignment 1:

Conversion from signal flow diagrams to equations and vice versa (2 of each)

Coding some filters in Matlab (2 filters, can do whatever you'd like)

Processing audio with filters (2 examples) (include write-up of what kind of filter you used and what you did)

Submit to Box Folder (which I sometimes call the server)