MUS420/EE367A Lecture 2
Computational Acoustic Modeling with Digital Delay

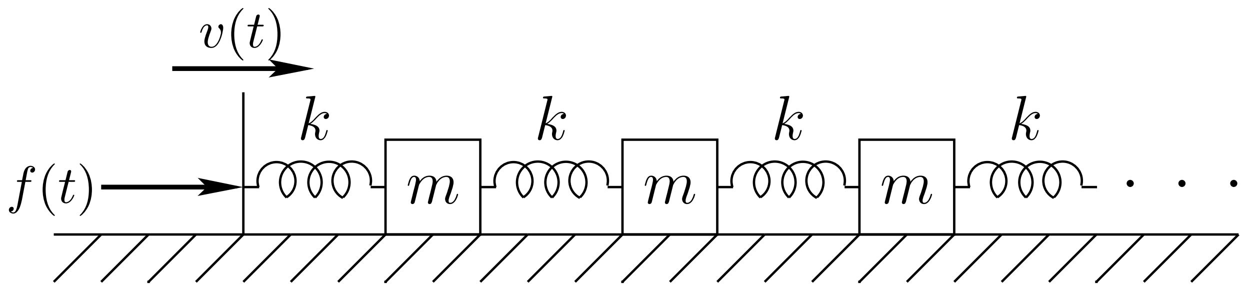
Julius O. Smith III (jos@ccrma.stanford.edu)
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

April 29, 2015

Outline

- Lumped and Distributed Modeling
- Delay lines
- Filtered Delay lines
- Digital Waveguides
- Echo simulation
- Comb filters
- Vector Comb Filters (Feedback Delay Networks)
- Tapped Delay Lines and FIR Filters
- Allpass filters
- Artificial Reverberation

From Lumped to Distributed Modeling



As mass-spring[1] density approaches infinity, we obtain an *ideal string*, governed by "wave equation" PDEs such as

$$\boxed{Y d'' = \rho\, \ddot{d}}$$

where, for longitudinal displacement $d(t,x)$, we have

$$\begin{aligned} Y &\stackrel{\Delta}{=} \text{Young's Modulus} & d &\stackrel{\Delta}{=} d(t,x) \\ \rho &\stackrel{\Delta}{=} \text{mass density} & \dot{d} &\stackrel{\Delta}{=} \frac{\partial}{\partial t} d(t,x) \\ d &\stackrel{\Delta}{=} \text{longitudinal displacement} & d' &\stackrel{\Delta}{=} \frac{\partial}{\partial x} d(t,x) \end{aligned}$$

The wave equation is once again Newton's $f = ma$, but now for each differential string element:

$$\begin{aligned} Y d'' &= \textit{force density} \text{ on the element} \\ \rho \ddot{d} &= \textit{inertial reaction force density} \\ &= \textit{mass-density} \text{ times } \textit{acceleration} \end{aligned}$$

[1] *Transverse waves* demo: `http://phet.colorado.edu/sims/wave-on-a-string/wave-on-a-string_en.html`

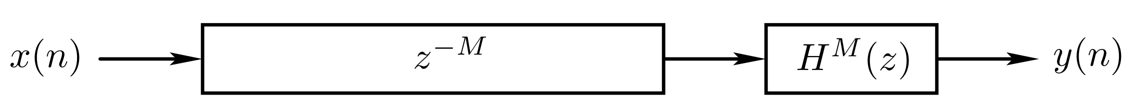
Wave Equation Solution

It is easily checked that the wave equation $Yd'' = \epsilon\ddot{d}$ is obeyed by any pair of *traveling waves*

$$d(t,x) \;=\; d_r\left(t - \frac{x}{c}\right) \;+\; d_l\left(t + \frac{x}{c}\right)$$

where

- $d_l(\cdot)$ and $d_r(\cdot)$ are *arbitrary* twice-differentiable displacement functions
- $c = \sqrt{Y/\rho}$ for longitudinal waves, where Y is Young's modulus = "spring constant" for solids (stress/strain $\stackrel{\Delta}{=}$ force-per-unit-area / relative displacement)
- Published for transverse waves by d'Alembert in 1747 (wave equation itself introduced in same paper)
- We can *sample* these traveling-wave components to obtain the super-efficient *digital waveguide* modeling approach for strings and acoustic tubes (and more)
- Any acoustic "ray" or propagating wave can be implemented digitally using a simple *delay line* followed by *linear filtering* to implement *loss* and/or *dispersion:*



Delay lines

Delay lines are important building blocks for many audio effects and synthesis algorithms, including

- Digital audio effects
  - Phasing
  - Flanging
  - Chorus
  - Leslie
  - Reverb
- Physical modeling synthesis
  - Acoustic propagation delay (echo, multipath)
  - Vibrating strings (guitars, violins, . . .)
  - Woodwind bores
  - Horns
  - Percussion (rods, membranes)

The M-Sample Delay Line

$$x(n) \longrightarrow \boxed{\quad z^{-M} \quad} \longrightarrow y(n)$$

- $y(n) = x(n - M)$, $n = 0, 1, 2, \ldots$
- Must define $x(-1), x(-2), \ldots, x(-M)$ (usually zero)

Delay Line as a Digital Filter

$$x(n) \longrightarrow \boxed{\quad z^{-M} \quad} \longrightarrow y(n)$$

Difference Equation

$$y(n) = x(n-M)$$

Transfer Function

$$H(z) = z^{-M}$$

- M poles at $z = 0$
- M zeros at $z = \infty$

Frequency Response

$$H(e^{j\omega T}) = e^{-jM\omega T}, \quad \omega T \in [-\pi, \pi)$$

- "Allpass" since $\left|H(e^{j\omega T})\right| = 1$
- "Linear Phase" since $\angle H(e^{j\omega T}) = -M\omega T = \alpha\omega$

Delay Line in C

C Code:

```
static double D[M]; /* initialized to zero */
static long ptr=0;  /* read-write offset */

double delayline(double x)
{
  double y = D[ptr]; /* read operation */
  D[ptr++] = x;      /* write operation */
  if (ptr >= M) { ptr -= M; } /* wrap ptr */
  return y;
}
```

- *Circular buffer* in software
- Shared read/write pointer
- Length not easily modified in real time
- Internal state ("instance variables")
  $=$ length M array $+$ read pointer

Ideal Traveling-Wave Simulation

$$x(n) \longrightarrow \boxed{\quad z^{-M} \quad} \longrightarrow y(n)$$

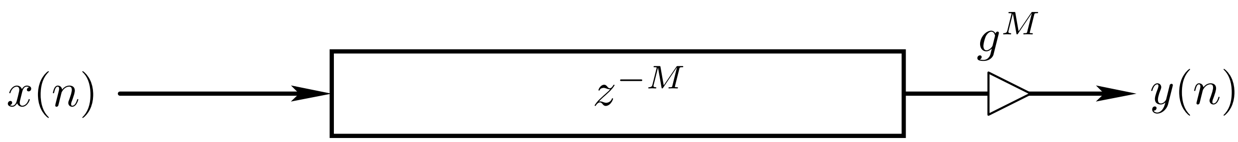
Acoustic Plane Waves in Air

- $x(n) =$ *excess pressure* at time nT, at some fixed point $p_x \in \mathbf{R}^3$ through which a *plane wave* passes
- $y(n) =$ excess pressure at time nT, for a point p_y which is McT meters "downstream" from p_x along the direction of travel for the plane wave, where
  - T denotes the *time sampling interval* in seconds
  - c denotes the *speed of sound* in meters per second
  - In one temporal sampling interval (T seconds), sound travels one spatial sample ($X = cT$ meters)

Transverse Waves on a String

- $x(n) =$ *displacement* at time nT, for some point on the string
- $y(n) =$ transverse displacement at a point McT meters away on the string

Lossy Traveling-Wave Simulator



- Propagation delay $= M$ samples
- Assume (or observe) *exponential decay* in direction of wave travel
- *Distributed attenuation* is *lumped* at one point along the ray: $g^M < 1$
- Input/output simulation is *exact* at the sampling instants
- Only deviation from ideal is that simulation is *bandlimited*

Traveling-Wave Simulation with Frequency-Dependent Losses

In all acoustic systems of interest, propagation losses *vary with frequency*.

- Propagation delay $= M$ samples + filter delay
- Attenuation $= \left|G(e^{j\omega T})\right|^M$
- Filter is linear and time-invariant (LTI)
- Propagation delay and attenuation can now vary with frequency
- For physical passivity, we require

$$\left|G(e^{j\omega T})\right| \leq 1$$

for all ω.

Dispersive Traveling-Wave Simulation

In many acoustic systems, such as piano strings, wave propagation is also *dispersive*

$$x(n) \longrightarrow \boxed{\quad z^{-M} \quad} \longrightarrow \boxed{A^M(z)} \longrightarrow y(n)$$

- This is simulated using an allpass filter $A(z)$ having *nonlinear phase*
- Since dispersive wave propagation is *lossless*, the dispersion filter is "allpass," i.e.,

$$\left|A(e^{j\omega T})\right| \equiv 1, \ \forall \omega$$

- Note that a delay line is also an allpass filter:

$$\left|e^{j\omega MT}\right| \equiv 1, \ \forall \omega$$

Recursive Allpass Filters

In general, (finite-order) allpass filters can be written as

$$H(z) = e^{j\phi} z^{-K} \frac{\tilde{A}(z)}{A(z)}$$

where

$$\begin{aligned}
A(z) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N} \\
\tilde{A}(z) &\triangleq z^{-N} \overline{A}(z^{-1}) \\
&\triangleq \overline{a}_N + \overline{a}_{N-1} z^{-1} + \cdots + \overline{a}_1 z^{-(N-1)} + \cdots + z^{-N}
\end{aligned}$$

- The polynomial $\tilde{A}(z)$ can be obtained by reversing the order of the coefficients in $A(z)$ and conjugating them
- The problem of *dispersion filter design* is typically formulated as an *allpass-filter design* problem

Phase Delay and Group Delay

Phase Response:

$$\boxed{\Theta(\omega) \stackrel{\Delta}{=} \angle H(e^{j\omega T})}$$

Phase Delay:

$$\boxed{P(\omega) \stackrel{\Delta}{=} -\frac{\Theta(\omega)}{\omega}} \quad \text{(Phase Delay)}$$

Group Delay:

$$\boxed{D(\omega) \stackrel{\Delta}{=} -\frac{d}{d\omega}\Theta(\omega)} \quad \text{(Group Delay)}$$

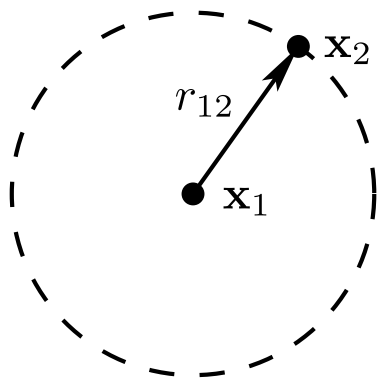
- For a slowly modulated sinusoidal input signal $x(n) = A(nT)\cos(\omega nT + \phi)$, the output signal is

  $$\boxed{y(n) \approx G(\omega)A[nT - D(\omega)] \cdot \cos\{\omega[nT - P(\omega)] + \phi\}}$$

  where $G(\omega) \stackrel{\Delta}{=} |H(e^{j\omega T})|$ is the *amplitude response*.

- *Unwrap* phase response $\Theta(\omega)$ to uniquely define it:
  - $\Theta(0) \stackrel{\Delta}{=} 0$ or $\pm\pi$ for real filters
  - Discontinuities in $\Theta(\omega)$ cannot exceed $\pm\pi$ radians
  - Phase jumps $\pm\pi$ radians are *equivalent*
  - See Matlab function `unwrap`

Acoustic Point Source



- Let $\mathbf{x} = (x, y, z)$ denote the *Cartesian coordinates* of a point in 3D space
- Point source at $\mathbf{x} = \mathbf{x}_1 = (x_1, y_1, z_1)$
- Listening point at $\mathbf{x} = \mathbf{x}_2 = (x_2, y_2, z_2)$
- Propagation distance:

$$r_{12} = \| \mathbf{x}_2 - \mathbf{x}_1 \| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Acoustic pressure peak amplitude (or rms level) at $\mathbf{x} = \mathbf{x}_2$ is given by

$$p(\mathbf{x}_2) = \frac{p_1}{r_{12}}$$

where p_1 is the peak amplitude (or rms level) at $r_{12} = \| \mathbf{x}_2 - \mathbf{x}_1 \| = 1$

Notice that pressure decreases as $1/r$ away from the point source

Inverse Square Law for Acoustics

The *intensity* of a sound is proportional to the *square* of its sound pressure p, where pressure is force per unit area

Therefore, the *average intensity* at distance r_{12} away from a point source of average-intensity

$$I_1 \propto \left\langle |p_1|^2 \right\rangle \qquad \text{is} \qquad \boxed{I(\mathbf{x}_2) = \frac{I_1}{r_{12}^2}}$$

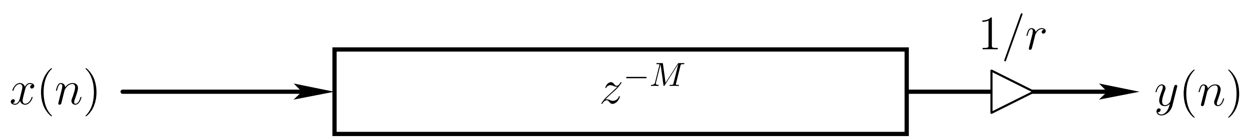
This is a so-called *inverse square law*.

Remember that far away (in wavelengths) from a finite sound source,

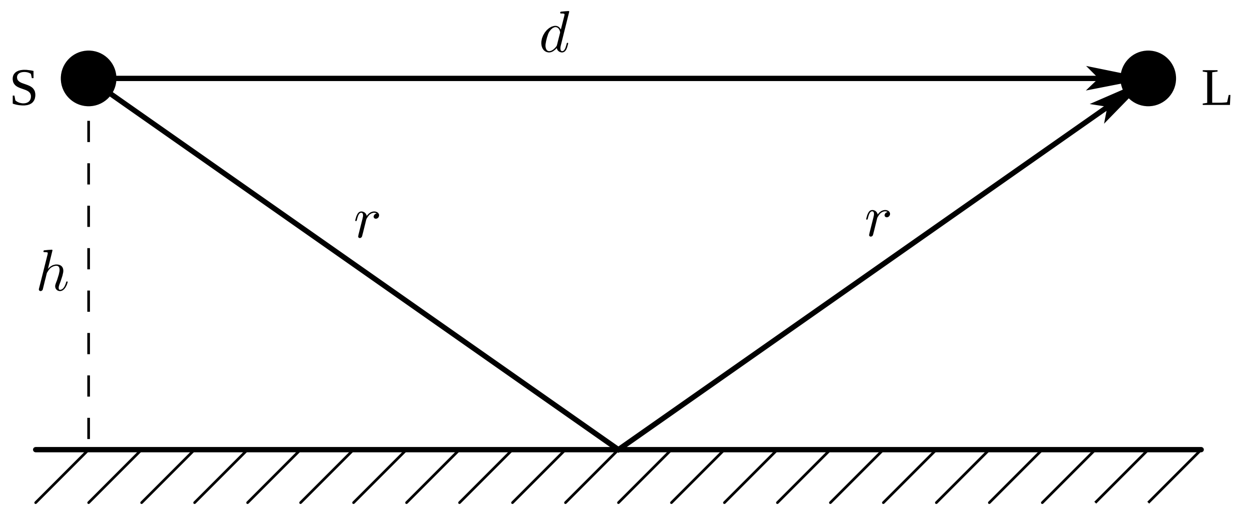
- pressure falls off as $1/r$
- intensity falls off as $1/r^2$

where r is the distance from the source.

Point-to-Point Spherical Pressure-Wave Simulation:



Acoustic Echo



- Source S, Listener L
- Height of S and L above floor is h
- Distance from S to L is d
- Direct sound travels distance d
- Floor-reflected sound travels distance $2r$, where

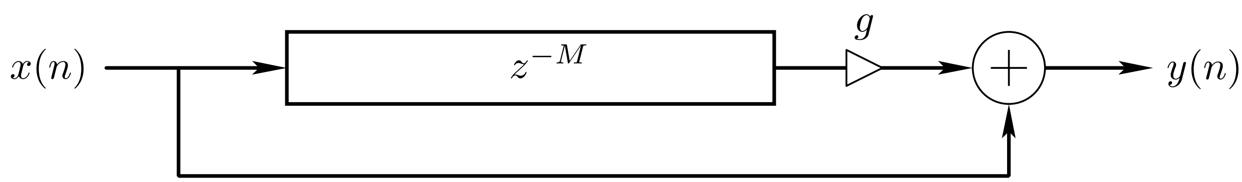
$$r^2 = h^2 + \left(\frac{d}{2}\right)^2$$

- Direct sound and reflection *sum* at listener L

$$p_L(t) \;\propto\; \frac{p_S\left(t - \frac{d}{c}\right)}{d} + \frac{p_S\left(t - \frac{2r}{c}\right)}{2r}$$

- Also called *multipath*

Acoustic Echo Simulator



- Delay line length set to *path-length difference*:

$$M = \frac{2r - d}{cT}$$

where

c = sound speed

T = sampling period

- Gain coefficient g set to *relative attenuation*:

$$g = \frac{1/2r}{1/d} = \frac{d}{2r} = \frac{1}{\sqrt{1 + (2h/d)^2}}$$

- M typically *rounded* to nearest integer
- For non-integer M, delay line must be *interpolated*

STK Program for Digital Echo Simulation

The Synthesis Tool Kit (STK)[2] is an object-oriented C++ tool kit useful for rapid prototyping of real-time computational acoustic models.

```
#include "FileWvIn.h"  /* STK soundfile input support  */
#include "FileWvOut.h" /* STK soundfile output support */
#include "Stk.h"       /* STK global variables, etc.   */

static const int M = 20000;   /* echo delay in samples */
static const StkFloat g = 0.8; /* relative gain factor */

#include "delayline.c" /* defined previously */

int main(int argc, char *argv[])
{
  unsigned long i;
  FileWvIn input(argv[1]); /* read input soundfile */
  FileWvOut output("main"); /* creates main.wav */
  unsigned long nframes = input.getSize();
  for (i=0;i<nframes+M;i++)   {
    StkFloat insamp = input.tick();
    output.tick(insamp + g * delayline(insamp));
  }
}
```

[2] http://ccrma.stanford.edu/CCRMA/Software/STK/

General Loss Simulation

The substitution

$$z^{-1} \leftarrow g z^{-1}$$

in any transfer function *contracts all poles by the factor* g.

Example (delay line):

$$H(z) = z^{-M} \;\rightarrow\; g^M z^{-M}$$

Thus, the contraction factor g can be interpreted as the *per-sample propagation loss factor*.

Frequency-Dependent Losses:

$$z^{-1} \leftarrow G(z) z^{-1}, \quad \left|G(e^{j\omega T})\right| \leq 1$$

$G(z)$ can be considered the *filtering per sample* in the propagation medium. A lossy delay line is thus described by

$$Y(z) = G^M(z) z^{-M} X(z)$$

in the frequency domain, and *iterated convolution*

$$y(n) = \underbrace{g * g * \ldots * g *}_{M \text{ times}} x(n - M)$$

in the time domain

Air Absorption

The intensity of a *plane wave* is observed to decay *exponentially* according to

$$I(x) = I_0 \, e^{-x/\xi}$$

where

$$\begin{aligned} I_0 &= \text{intensity at the plane source (\textit{e.g.}, a vibrating wall)} \\ I(x) &= \text{intensity } x \text{ meters from the plane-source} \\ \xi &= \text{intensity decay constant } (1/e \text{ distance in meters}) \\ &\quad\ \text{(depends on frequency, temperature, humidity} \\ &\quad\ \text{and pressure)} \end{aligned}$$

Relative Humidity	Frequency in Hz 1000	2000	3000	4000
40	5.6	16	30	105
50	5.6	12	26	90
60	5.6	12	24	73
70	5.6	12	22	63

Attenuation in dB per kilometer at 20°C and standard atmospheric pressure.

Acoustic Intensity

Acoustic Intensity (a real vector) may be defined by

$$\boxed{\underline{I} \stackrel{\Delta}{=} p\underline{v}} \qquad \left(\frac{\text{Energy Flux}}{\text{Area} \cdot \text{Time}} = \frac{\text{Power Flux}}{\text{Area}}\right)$$

where

$$\begin{aligned} p &= \text{ acoustic pressure} \qquad \left(\frac{\text{Force}}{\text{Area}}\right) \\ \underline{v} &= \text{ acoustic particle velocity} \qquad \left(\frac{\text{Length}}{\text{Time}}\right) \end{aligned}$$

For a *traveling plane wave*, we have

$$\boxed{p = Rv}$$

where

$$R \stackrel{\Delta}{=} \rho c$$

is called the *wave impedance* of air, and

$$\begin{aligned} c &= \text{ sound speed} \\ \rho &= \text{ mass density of air} \qquad \left(\frac{\text{Mass}}{\text{Volume}}\right) \\ v &\stackrel{\Delta}{=} |\underline{v}| \end{aligned}$$

Therefore, in a plane wave,

$$\boxed{I \;\stackrel{\Delta}{=}\; pv \;=\; Rv^2 \;=\; \frac{p^2}{R}}$$

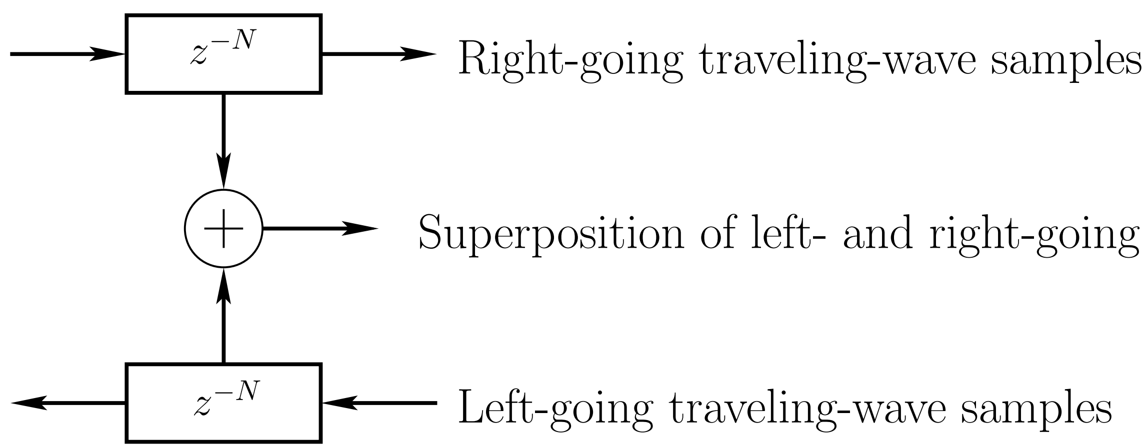
From 1D$+$ to 1D$\pm$

We have been modeling *unidirectional* traveling waves:

$$\text{Attenuation per sample } = |H(e^{j\omega T})|$$

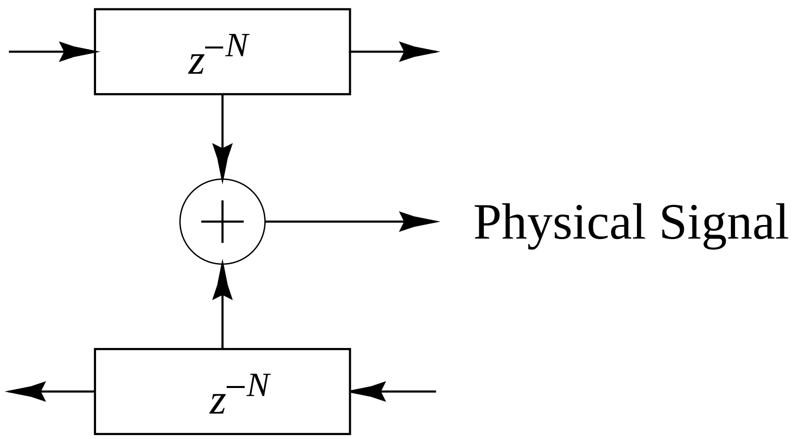
$$\text{Phase-shift per sample } = \angle H(e^{j\omega T})$$

Thanks to *superposition*, we can simulate *both directions of propagation* in a 1D medium *separately* and add them only when needed:



(Lossless, Non-Dispersive Case)

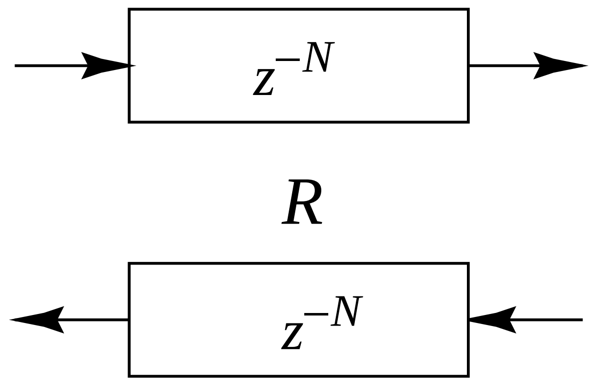
Digital Waveguide Models



There are many musical applications of $1D^{\pm}$ simulations:

- vibrating strings
- woodwind bores
- pipes
- horns
- vocal tracts

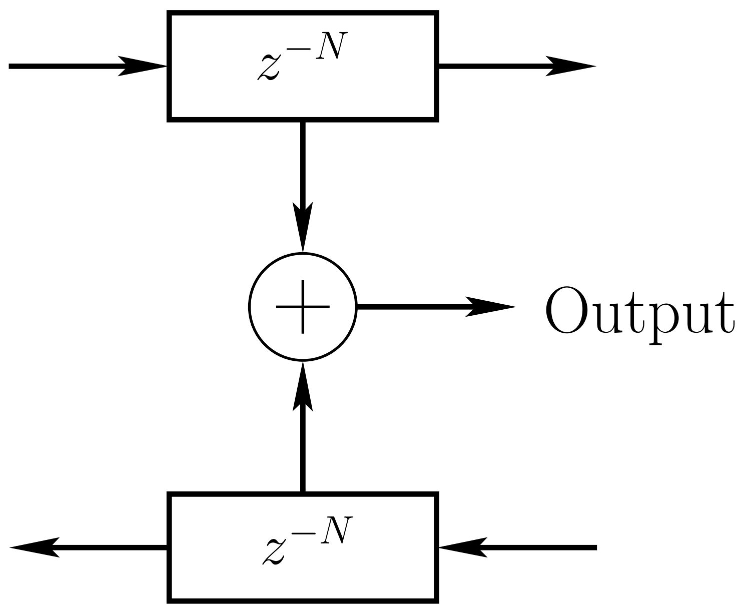
Digital Waveguide Definition



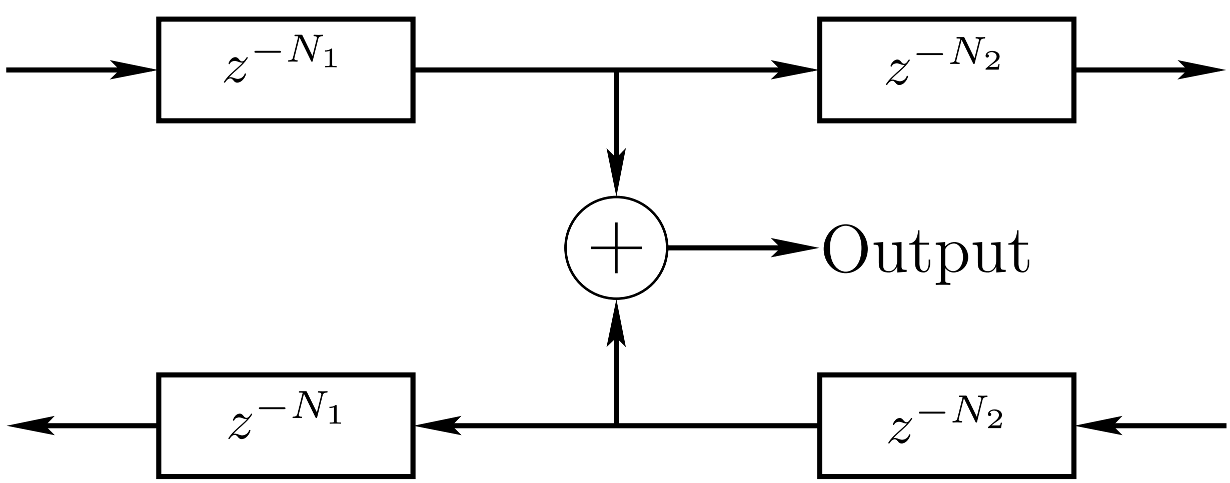
- A *digital waveguide* is defined as a *"bidirectional delay line"* associated with a (real) wave impedance $R > 0$
- A digital waveguide simulates ideal wave propagation (lossless, non-dispersive) *exactly* for frequencies f below the Nyquist limit $f_s/2$
- We'll derive R from first principles later on (for ideal strings)

Physical Outputs

The diagram



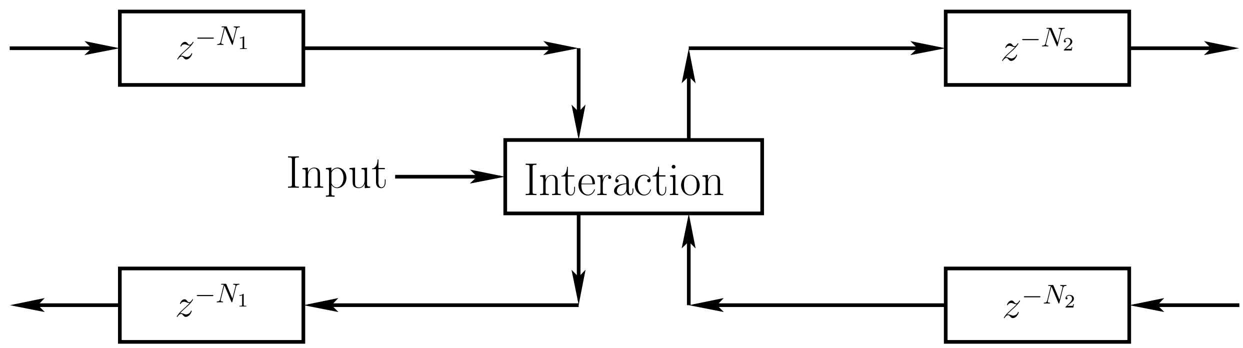
means summing opposite samples using *delay taps:*



Physical Inputs

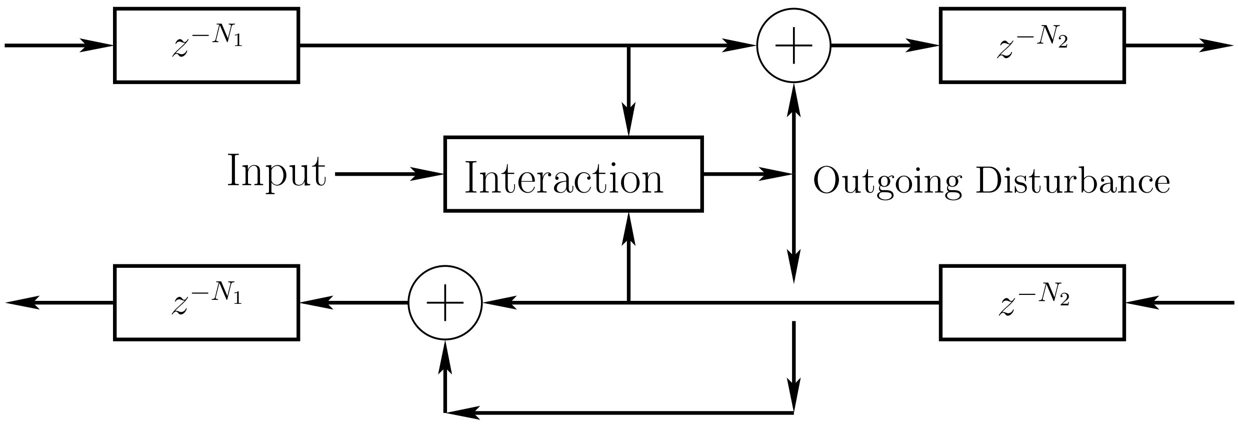
input signal = *disturbance* of the propagation medium

General Case



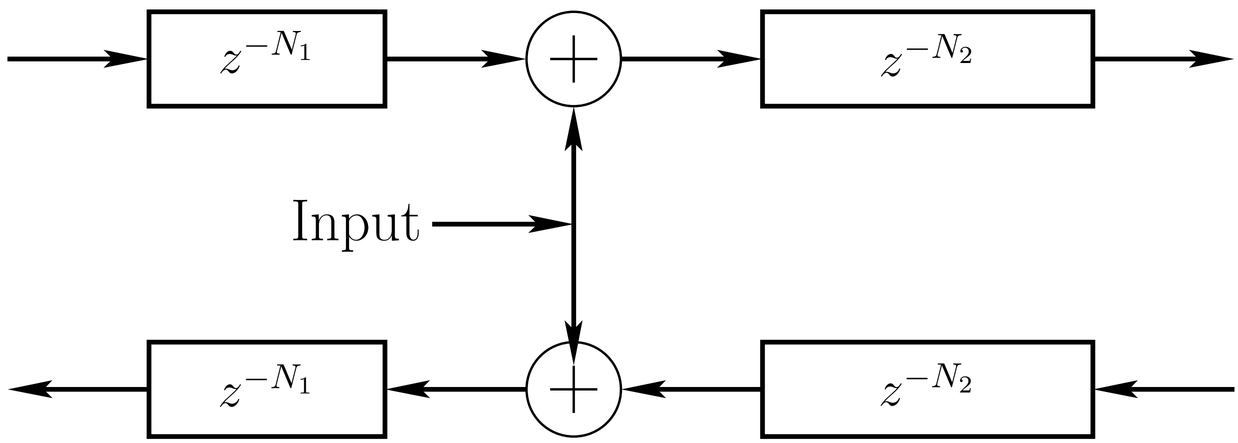
- Interaction can only depend on the "incoming state" (traveling-wave components) and driving input signal
- Interaction is at *one spatial point* in this example
- Delay-line inputs from interaction are usually equal in magnitude (by physical symmetry)

Symmetric Superimposing Outgoing Disturbance



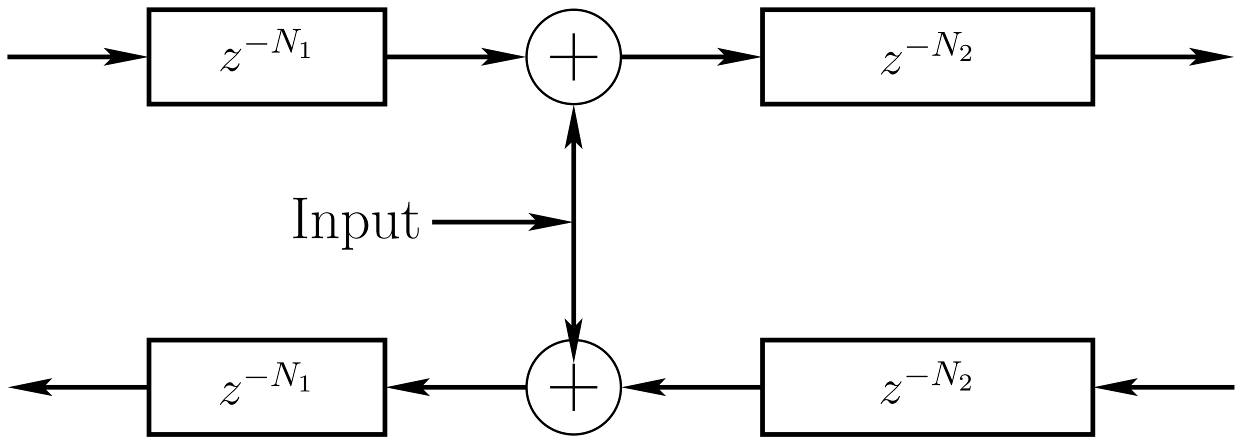
- Less general but typical
- Outgoing disturbance equal to left and right (signs may differ)
- Disturbance sums with the incoming waves
  - Output superimposes on unperturbed state
  - No loss of generality in choosing this formulation (can always include a canceling term in the output)

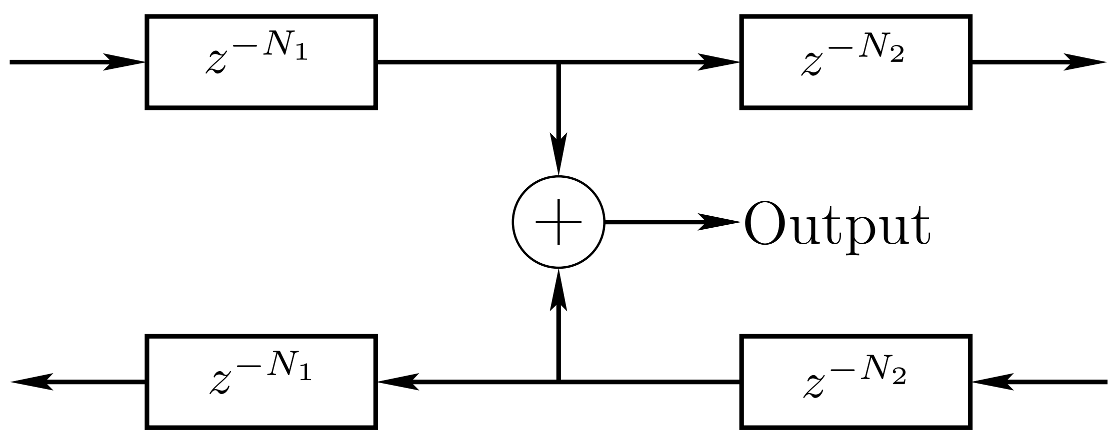
Pure Superimposing Input



- Original state unaffected
- Input *sums* with existing state
- Often hard to realize physically

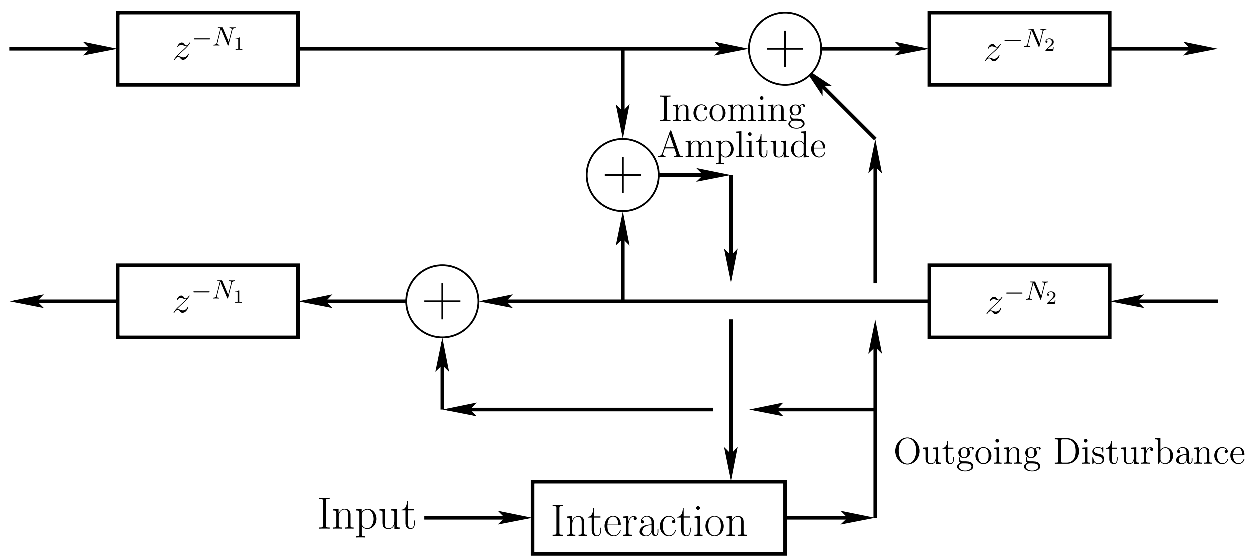
Idealized Inputs and Outputs





- *Superimposing inputs* and *non-loading outputs* can only be *approximated* in real-world systems
- Superimposing input is the graph-theoretic *transpose* of an ideal output — two *"transposed taps"*
  - Physical *inputs* usually *disturb* the system state non-additively
  - Physical *ouputs* always present some *load* on the system (energy must be extracted)

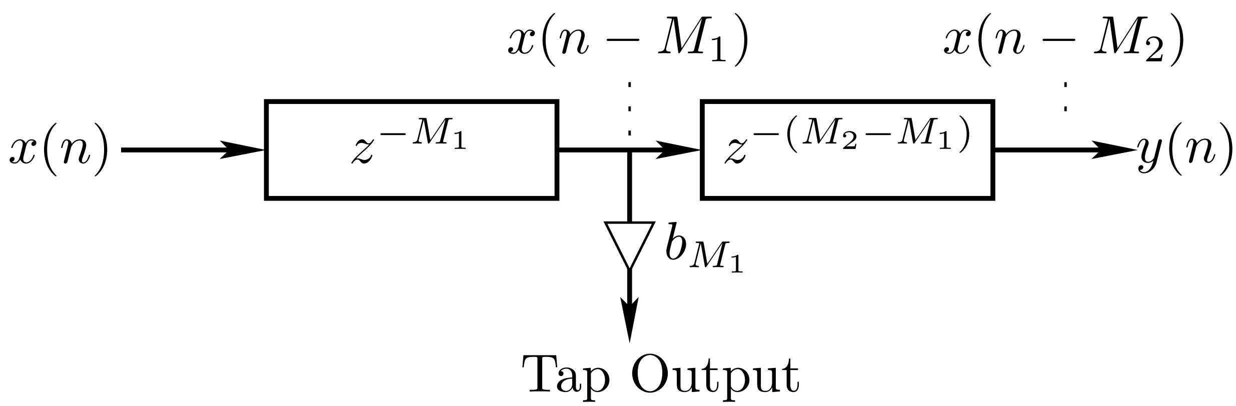
Amplitude-Determined Superimposing Symmetric Outgoing Disturbance



- Interaction depends only upon *incoming amplitude* (sum of incoming traveling waves)
- Used in many practical waveguide models
  - guitar plectra
  - violin bows
  - woodwind reeds
  - flue-pipe air-jets (flute, organ, . . .)

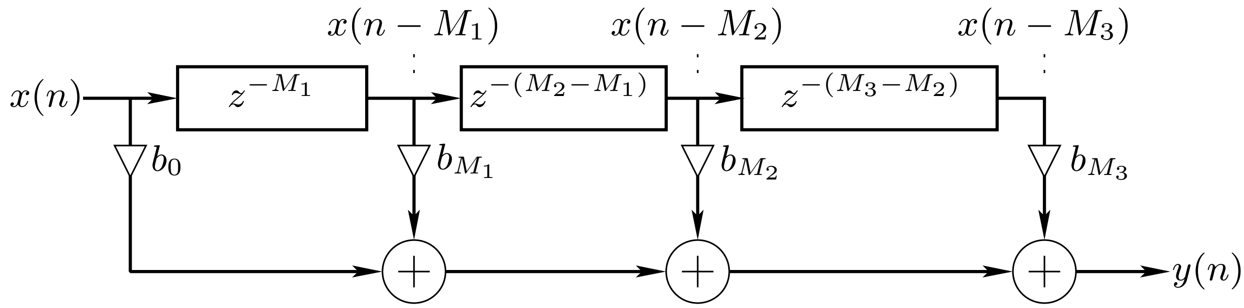
Tapped Delay Lines (TDL)

- A *tapped delay line* (TDL) is a delay line with at least one "tap"
- A *tap* brings out and scales a signal inside the delay line
- A tap may be *interpolating* or *non-interpolating*

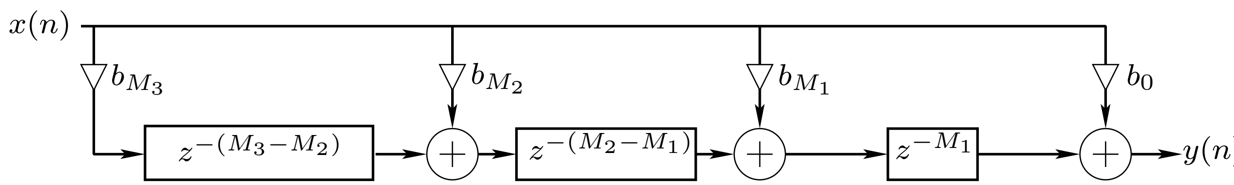


- TDLs efficiently simulate *multiple echoes* from the *same source*
- Extensively used in *artificial reverberation*

Transposed Tapped Delay Line (TTDL)



Tapped Delay Line (TDL)



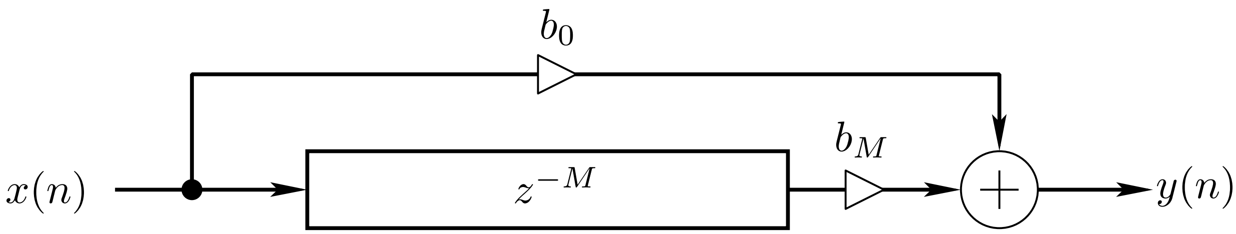
Transposed Tapped Delay Line (TTDL)

A flow-graph is *transposed* (or "reversed") by reversing all signal paths:

- Branchpoints become sums
- Sums become branchpoints
- Input/output exchanged
- Transfer function *identical* for SISO systems
  - Derives from *Mason's gain formula*
- Transposition converts direct-form I & II digital filters to two more direct forms

Comb Filters

Feedforward Comb Filter



$$\begin{aligned} b_0 &= \text{Feedforward coefficient} \\ b_M &= \text{Delay output coefficient} \\ M &= \text{Delay-line length in samples} \end{aligned}$$

Difference Equation

$$y(n) = b_0\, x(n) + b_M\, x(n - M)$$

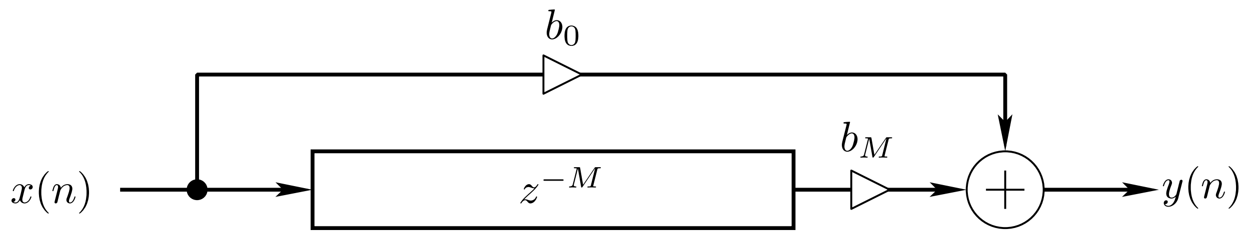
Transfer Function

$$H(z) = b_0 + b_M z^{-M}$$

Frequency Response

$$H(e^{j\omega T}) = b_0 + b_M e^{-jM\omega T}$$

Gain Range for Feedforward Comb Filter



$$y(n) = b_0\, x(n) + b_M\, x(n-M)$$

For a sinewave input, with $b_0, b_M > 0$:

- Gain is maximum $(b_0 + b_M)$ when a *whole number of periods* fits in M samples:

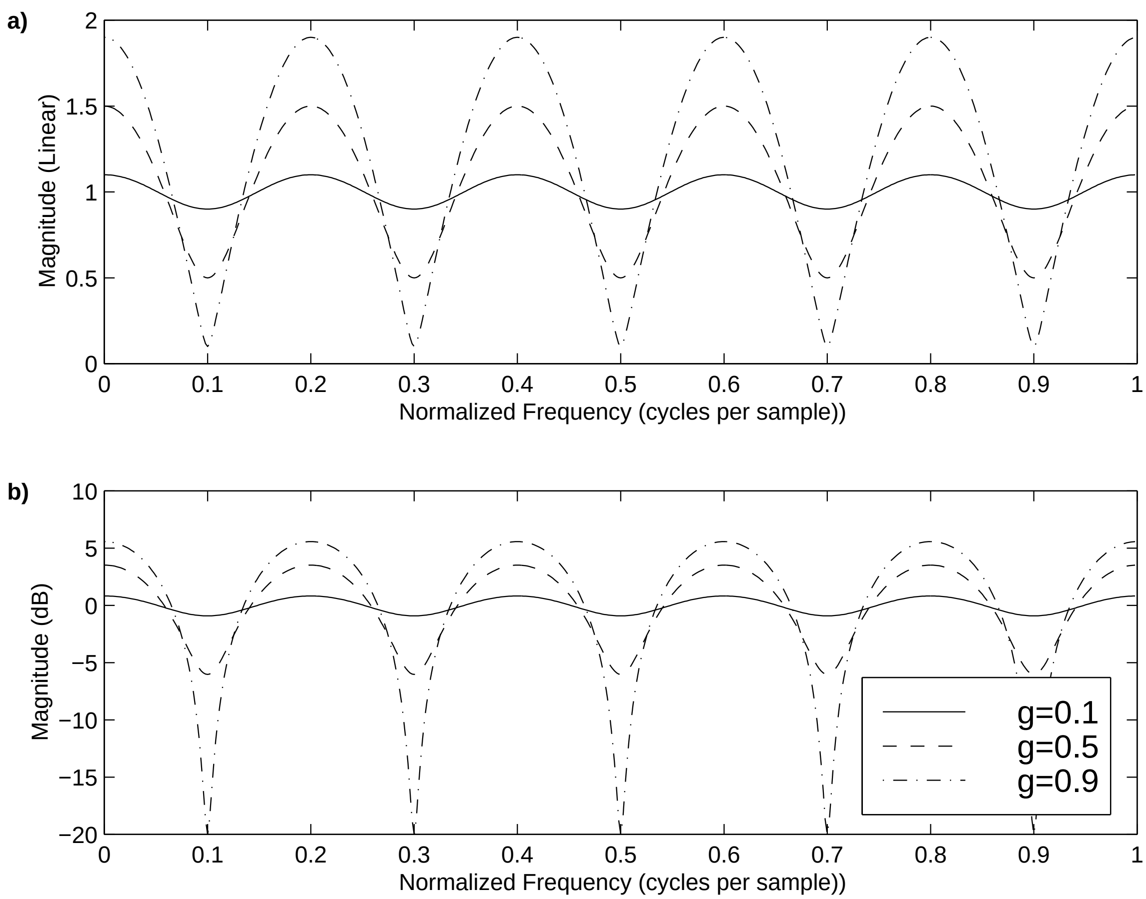
  $$\omega_k T = k\frac{2\pi}{M}, \quad k = 0, 1, 2, \ldots$$

  (the *DFT basis frequencies* for length M DFTs)

- Gain is minimum $(|b_0 - b_M|)$ when an *odd number of half-periods* fits in M samples:

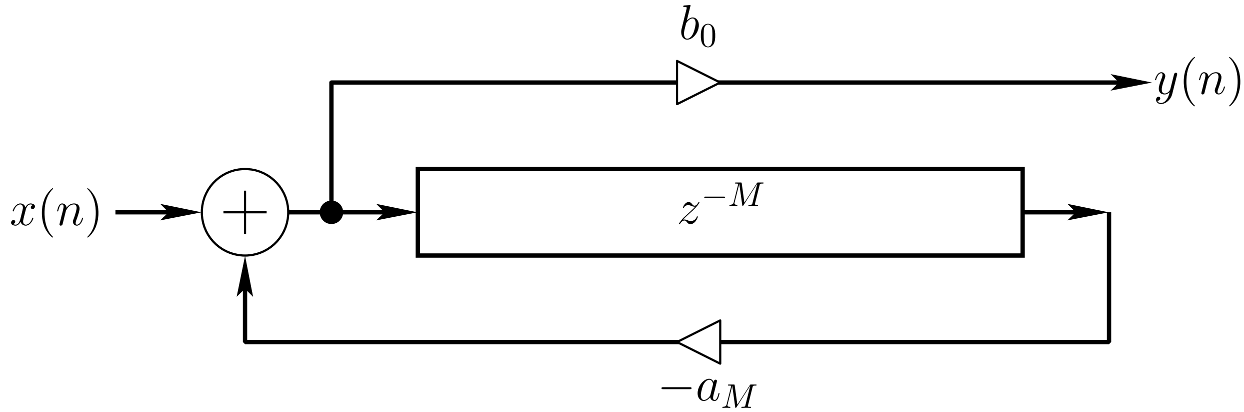
  $$\omega_k T = (2k+1)\frac{\pi}{M}, \quad k = 0, 1, 2, \ldots$$

Feed-Forward Comb-Filter Amplitude Response



- Linear (top) and decibel (bottom) amplitude scales
- $H(z) = 1 + gz^{-M}$
  - $M = 5$
  - $g = 0.1, 0.5, 0.9$
- $G(\omega) \stackrel{\Delta}{=} \left|H(e^{j\omega T})\right| = \left|1 + ge^{-jM\omega T}\right| \to 2\cos(M\omega T/2)$ when $g = 1$
- In *flangers*, these nulls slowly move with time

Feedback Comb Filter



$-a_M$ = Feedback coefficient (need $|a_M| < 1$ for stability)
M = Delay-line length in samples

Direct-Form-II Difference Equation (see figure):

$$\begin{aligned} v(n) &= x(n) - a_M\, v(n-M) \\ y(n) &= b_0\, v(n) \end{aligned}$$

Direct-Form-I Difference Equation

(commute gain b_0 to the input):

$$y(n) = b_0\, x(n) - a_M\, y(n-M)$$

Transfer Function

$$H(z) = \frac{b_0}{1 + a_M z^{-M}}$$

Frequency Response

$$H(e^{j\omega T}) = \frac{b_0}{1 + a_M e^{-jM\omega T}}$$

Simplified Feedback Comb Filter

Special case: $b_0 = 1$, $-a_M = g \;\Rightarrow$

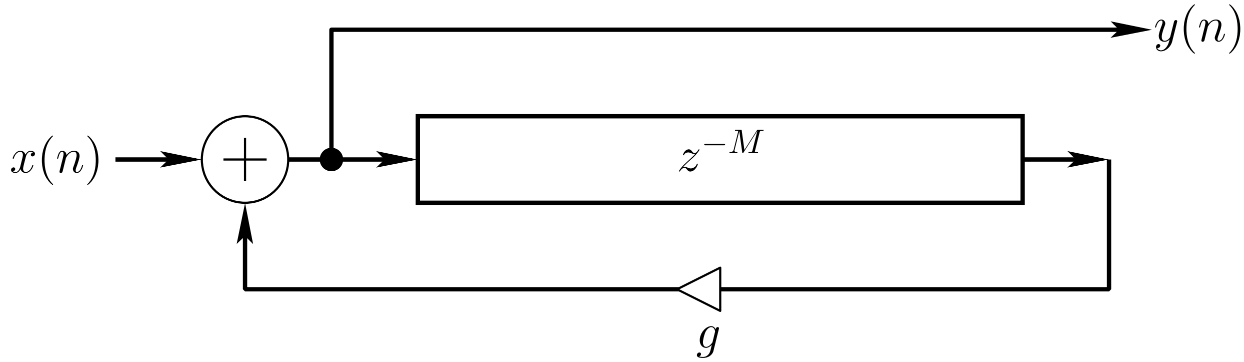
$$\begin{aligned} y(n) &= x(n) + g\, y(n-M) \\ H(z) &= \frac{1}{1 - g\, z^{-M}} \end{aligned}$$

- Impulse response is a *series* of echoes, exponentially decaying and uniformly spaced in time:

$$\begin{aligned} H(z) &= \frac{1}{1 - g\, z^{-M}} = 1 + g\, z^{-M} + g^2\, z^{-2M} + \cdots \\ &\longleftrightarrow \delta(n) + g\, \delta(n-M) + g^2\, \delta(n-2M) + \cdots \\ &= [1, \underbrace{0, \ldots, 0}_{M-1}, g, \underbrace{0, \ldots, 0}_{M-1}, g^2, 0, \ldots] \end{aligned}$$

- Models a *plane wave between parallel walls*
- Models *wave propagation on a guitar string*
- $g =$ *round-trip gain coefficient:*
  - two wall-to-wall traversals (two wall reflections)
  - two string traversals (two endpoint reflections)

Simplified Feedback Comb Filter, Cont'd



$$
\begin{aligned}
y(n) &= x(n) + g\,y(n-M) \\
H(z) &= \frac{1}{1 - g\,z^{-M}}
\end{aligned}
$$

For a sinewave input and $0 < g < 1$:

- Gain is maximum $[1/(1-g)]$ when a whole number of periods fits in M samples:

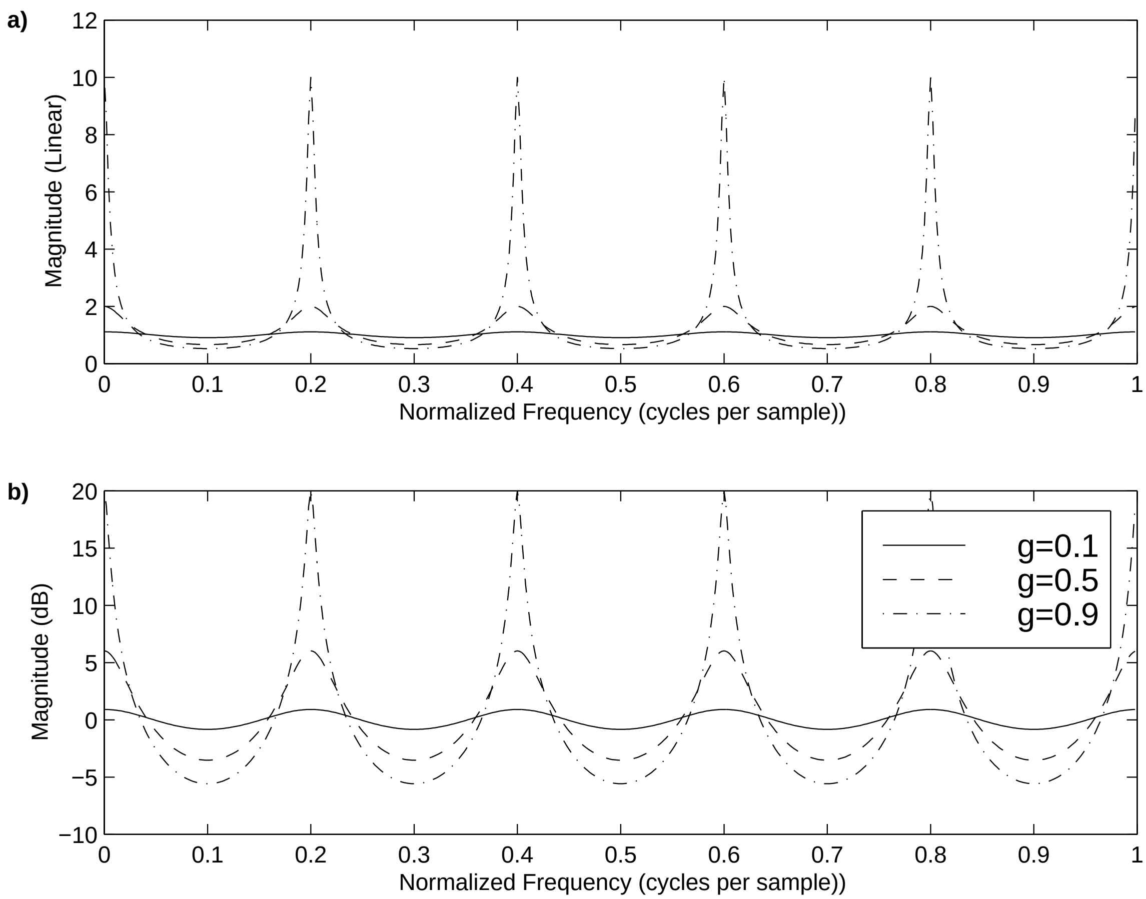
  $$\omega_k T = k\frac{2\pi}{M}, \quad k = 0, 1, 2, \ldots$$

  These are again the *DFT$_M$ basis frequencies*

- Gain is minimum $[1/(1+g)]$ when an odd number of half-periods fits in M samples:

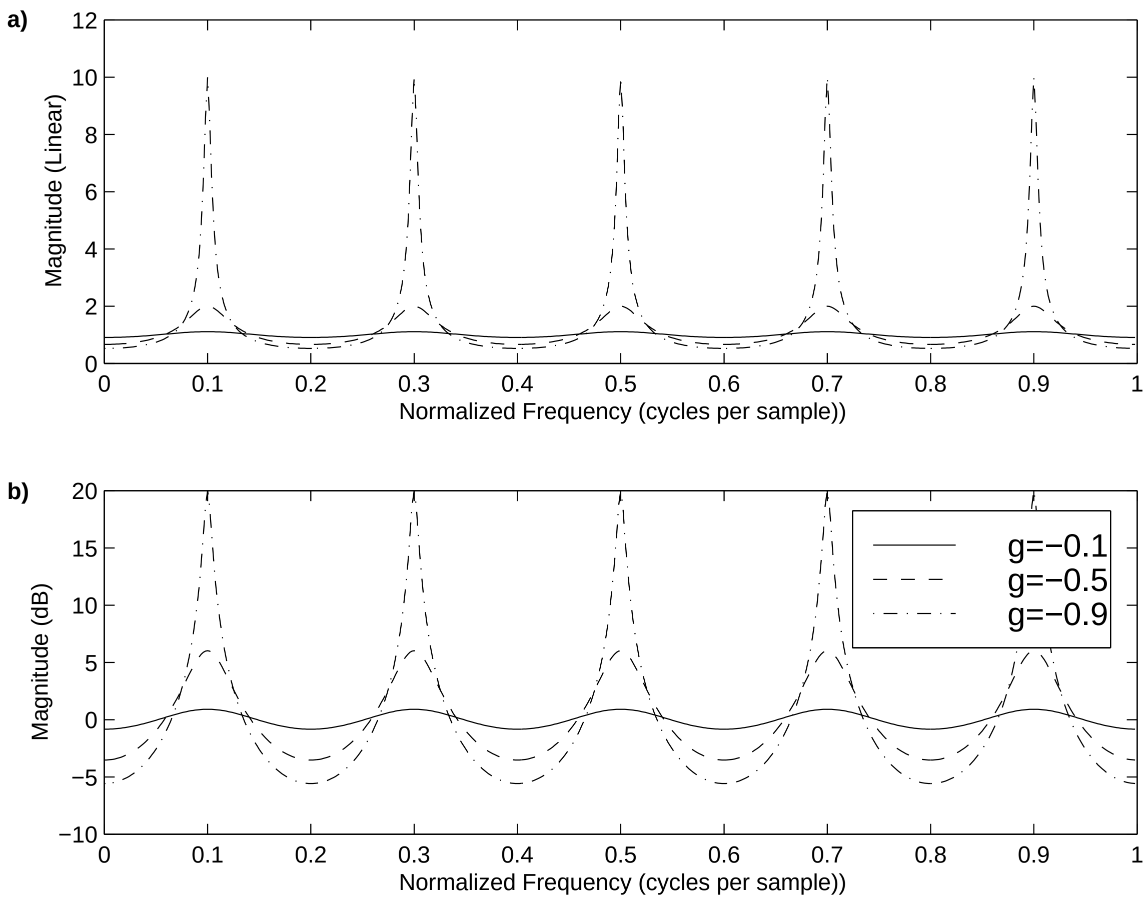
  $$\omega_k T = (2k+1)\frac{\pi}{M}, \quad k = 0, 1, 2, \ldots$$

Feed-Back Comb-Filter Amplitude Response



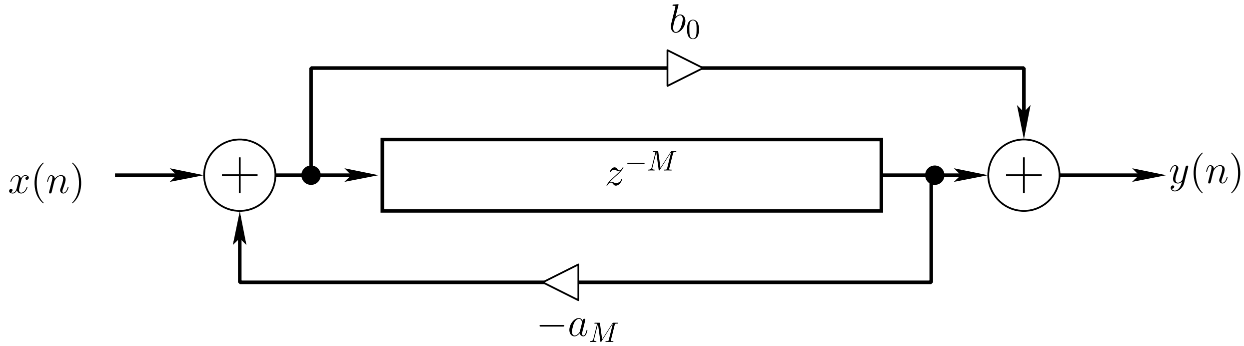
- Linear (top) and decibel (bottom) amplitude scales
- $H(z) = \frac{1}{1-gz^{-M}}$
- $M = 5, \quad g = 0.1, 0.5, 0.9$
- $G(\omega) \stackrel{\Delta}{=} \left|H(e^{j\omega T})\right| = \left|\frac{1}{1-ge^{-jM\omega T}}\right| \underset{g=1}{\rightarrow} \frac{1}{2\sin\left(\frac{M}{2}\omega T\right)}$

Inverted-Feed-Back Comb-Filter Amplitude Response



- Linear (top) and decibel (bottom) amplitude scales
- $H(z) = \frac{1}{1-gz^{-M}}$
- $M = 5, \quad g = -0.1, -0.5, -0.9$
- $G(\omega) \stackrel{\Delta}{=} \left|H(e^{j\omega T})\right| = \left|\frac{1}{1-ge^{-jM\omega T}}\right| \underset{g=-1}{\rightarrow} \frac{1}{2\cos\left(\frac{M}{2}\omega T\right)}$

Schroeder Allpass Filters



- Used extensively in artificial reverberation
- Transfer function:

$$H(z) = \frac{b_0 + z^{-M}}{1 + a_M z^{-M}}$$

- To obtain an allpass filter, set $b_0 = \overline{a_M}$

Proof:

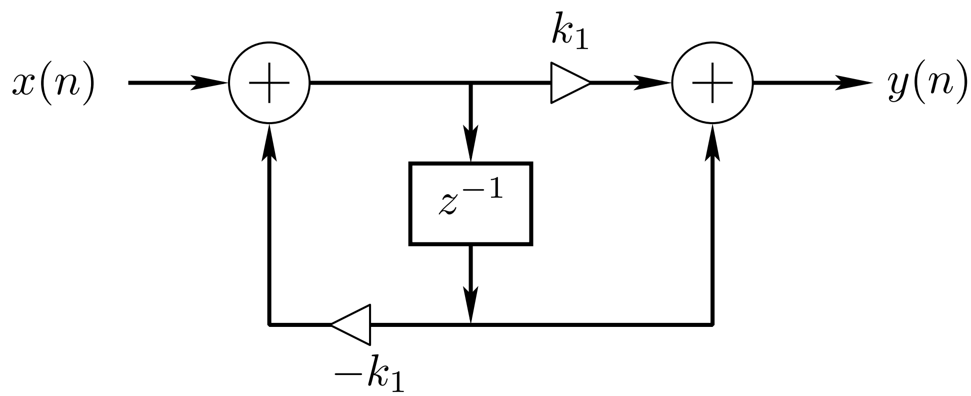
$$\begin{aligned} \left|H(e^{j\omega T})\right| &= \left|\frac{\overline{a} + e^{-jM\omega T}}{1 + ae^{-jM\omega T}}\right| = \left|\frac{\overline{a} + e^{-jM\omega T}}{e^{jM\omega T} + a}\right| \\ &= \left|\frac{\overline{a + e^{jM\omega T}}}{a + e^{jM\omega T}}\right| = 1 \end{aligned}$$

First-Order Allpass Filter

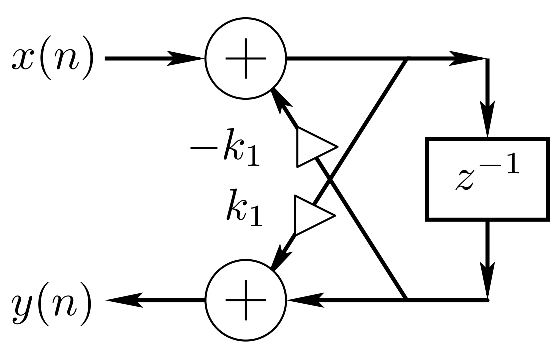
Transfer function:

$$H_1(z) = S_1(z) \triangleq \frac{k_1 + z^{-1}}{1 + k_1 z^{-1}}$$

(a)



(b)



(a) Direct form II filter structure

(b) Two-multiply lattice-filter structure

Nested Allpass Filter Design

Any delay-element or delay-line inside a stable allpass-filter can be replaced by any stable allpass-filter to obtain a new stable allpass filter:

$$z^{-1} \leftarrow H_a(z)\, z^{-1}$$

(The pure delay on the right-hand-side guarantees no delay-free loops are introduced, so that the original structure can be used)

Proof:

1. Allpass Property: Note that the above substitution is a *conformal map* taking the unit circle of the z plane to itself. Therefore, unity gain for $|z| = 1$ is preserved under the mapping.
2. Stability: Expand the transfer function in series form:

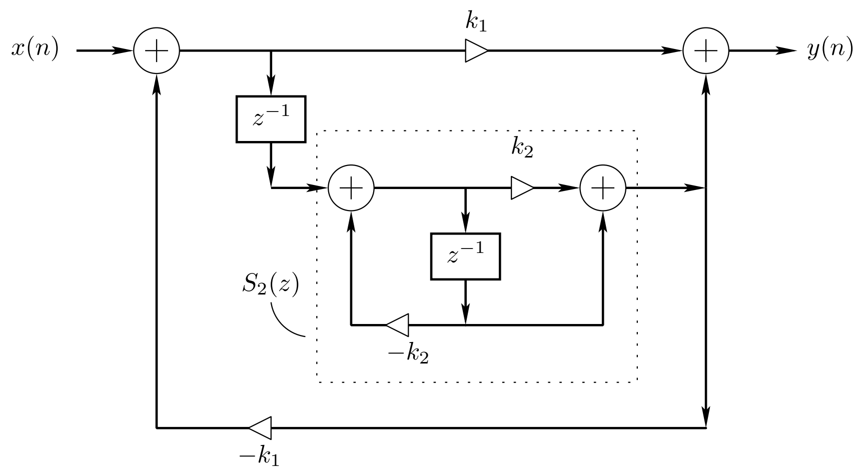
   $$S\left([H_a(z)z^{-1}]^{-1}\right) \;=\; s_0 + s_1 H_a(z) z^{-1} + s_2 H_a^2(z) z^{-2} + \cdots$$

   where $s_n =$ original impulse response. In this form, it is clear that stability is preserved if $H_a(z)$ is stable.

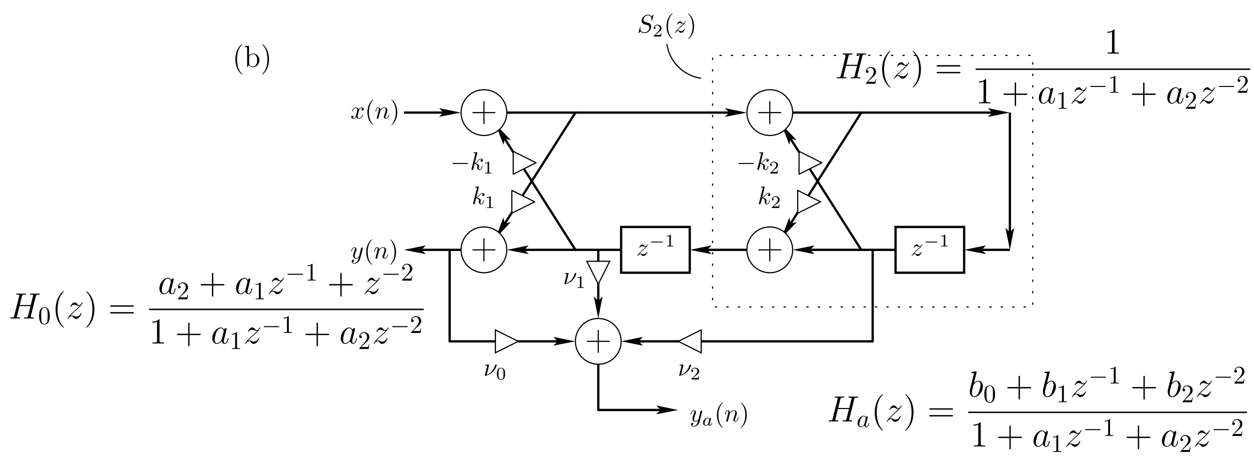
Nested Allpass Filters

$$H_2(z) = S_1\left(\left[z^{-1}S_2(z)\right]^{-1}\right) \stackrel{\Delta}{=} \frac{k_1 + z^{-1}S_2(z)}{1 + k_1 z^{-1} S_2(z)}$$

(a)



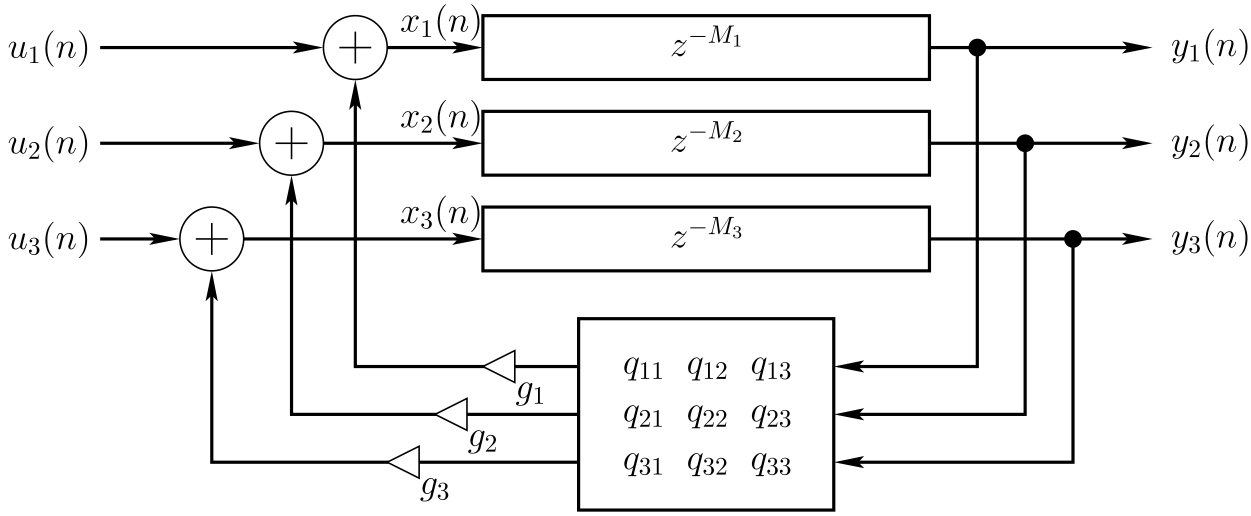
(b)



(a) *Nested* direct-form-II structures

(b) Two-multiply lattice-filter structure (equivalent)

Feedback Delay Network (FDN)



Order 3 MIMO FDN

- "Vectorized Feedback Comb Filter"
- Closely related to *state-space representations of LTI systems* ("vectorized one-pole filter")
- Transfer function, stability analysis, etc., essentially identical to corresponding state-space methods