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
As mass-spring\(^1\) density approaches infinity, we obtain an ideal string, governed by “wave equation” PDEs such as

\[ Y \ddot{d} = \rho \ddot{\dot{d}} \]

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

\[
\begin{align*}
Y & \triangleq \text{Young’s Modulus} & d & \triangleq d(t, x) \\
\rho & \triangleq \text{mass density} & \dot{d} & \triangleq \frac{\partial}{\partial t}d(t, x) \\
d & \triangleq \text{longitudinal displacement} & d' & \triangleq \frac{\partial}{\partial x}d(t, x)
\end{align*}
\]

The wave equation is once again Newton’s \(f = ma\), but now for each differential string element:

\[
Y \ddot{d} = \text{force density on the element} \\
\rho \ddot{\dot{d}} = \text{inertial reaction force density} = \text{mass-density times acceleration}
\]

\(^1\)Transverse waves demo: http://phet.colorado.edu/sims/wave-on-a-string/wave-on-a-string_en.html
Transverse Wave Equation: Ideal String

Wave Equation

\[ Ky'' = \epsilon y \]

\( K \) \( \Delta \) string tension
\( \epsilon \) \( \Delta \) linear mass density
\( y \) \( \Delta \) string displacement

Newton’s second law

[Force = Mass \times Acceleration]

Assumptions

- Lossless
- Linear
- Flexible (no “Stiffness”)
- Slope \( y'(t, x) \ll 1 \)
Derivation of Transverse String Wave Equation

Force diagram for length $dx$ string element

Total upward force on length $dx$ string element:

$$ f(x + dx/2) = K \sin(\theta_1) + K \sin(\theta_2) $$

$$ \approx K [\tan(\theta_1) + \tan(\theta_2)] $$

$$ = K [-y'(x) + y'(x + dx)] $$

$$ \approx K [-y'(x) + y'(x) + y''(x)dx] $$

$$ = Ky''(x)dx $$

Mass of length $dx$ string segment: $m = \epsilon \, dx$.

By Newton’s law, $f = ma = m\ddot{y}$, we have

$$ Ky''(t, x)dx = (\epsilon \, dx)\ddot{y}(t, x) $$

or

$$ Ky''(t, x) = \epsilon \ddot{y}(t, x) $$
Traveling-Wave Solution

One-dimensional lossless wave equation:

\[ Ky'' = \epsilon \ddot{y} \]

Plug in traveling wave to the right:

\[ y(t, x) = y_r(t - x/c) \]

\[ \Rightarrow \quad y'(t, x) = -\frac{1}{c} \dot{y}(t, x) \]

\[ y''(t, x) = \frac{1}{c^2} \ddot{y}(t, x) \]

• Given \( c \triangleq \sqrt{K/\epsilon} \), the wave equation is satisfied for any shape traveling to the right at speed \( c \) (but remember slope \( \ll 1 \))

• Similarly, any left-going traveling wave at speed \( c \), \( y_l(t + x/c) \), satisfies the wave equation (show)
• General solution to lossless, 1D, second-order wave equation:

\[ y(t, x) = y_r(t - x/c) + y_l(t + x/c) \]

• \( y_l(\cdot) \) and \( y_r(\cdot) \) are arbitrary twice-differentiable functions (slope \( \ll 1 \))

• **Important point:** Function of two variables \( y(t, x) \) is replaced by two functions of a single (time) variable \( \Rightarrow \) reduced computational complexity.

• Published by d’Alembert in 1747 (wave equation itself introduced in same paper)
Sampled Waves and Lumped Filters

We have that the wave equation $Y d'' = \epsilon \ddot{d}$ is obeyed by any pair of traveling waves

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

- $d_l(\cdot)$ and $d_r(\cdot)$ are arbitrary twice-differentiable displacement functions

- $c = \sqrt{K/\epsilon}$ for transverse waves, and $c = \sqrt{Y/\rho}$ for longitudinal waves, where $Y$ is Young’s modulus = “spring constant” for solids (stress/strain $\Delta = \frac{\text{force-per-unit-area}}{\text{relative displacement}}$), $\rho$ is mass per unit volume (rods), and $\epsilon$ is mass per unit length (ideal strings)

- 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:

\[ x(n) \xrightarrow{z^{-M}} H^M(z) \xrightarrow{} y(n) \]
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

\[ x(n) \rightarrow z^{-M} \rightarrow y(n) \]

- \( y(n) = x(n - M), \ n = 0, 1, 2, \ldots \)
- Must define \( x(-1), x(-2), \ldots, x(-M) \) (usually zero)
Delay Line as a Digital Filter

\[ x(n) \rightarrow z^{-M} \rightarrow y(n) \]

**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 \( |H(e^{j\omega T})| = 1 \)
- “Linear Phase” since \( \angle H(e^{j\omega T}) = -M\omega T = \alpha \omega \)
Delay Line in C

C Code:

```c
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
Delay Line in Faust

import("stdfaust.lib");
maxDelay = 16;
currentTimeDelay = 5;
process = de.delay(maxDelay, currentTimeDelay);

Generated C++ Code (Optimized!):

class mydsp : public dsp {
    ...
    float fVec0[6];
    ...
    virtual void compute(int count,
                           FAUSTFLOAT** inputs,
                           FAUSTFLOAT** outputs)
    {
        FAUSTFLOAT* input0 = inputs[0];
        FAUSTFLOAT* output0 = outputs[0];
        for (int i = 0; (i < count); i = (i + 1)) {
            fVec0[0] = float(input0[i]);
            output0[i] = FAUSTFLOAT(fVec0[5]);
            for (int j0 = 5; (j0 > 0); j0 = (j0 - 1)) {
                fVec0[j0] = fVec0[(j0 - 1)];
            }
        }
    }
};
import("stdfaust.lib");
maxDelay = 16;
process(x) = de.delay(maxDelay, x);

Generated C++ Code:

class mydsp : public dsp {
    private:
        int IOTA;
        float fVec0[32];
    ...
    virtual void compute( ... 
        ...
        for (int i = 0; (i < count); i = (i + 1)) {
            fVec0[(IOTA & 31)] = float(input1[i]);
            output0[i] = FAUSTFLOAT(fVec0[((IOTA - int(std::min<float>(16.0f,
                std::max<float>(0.0f,
                float(input0[i]))) & 31))));
            IOTA = (IOTA + 1);
        }
    }
};
Ideal Traveling-Wave Simulation

\[ x(n) \rightarrow z^{-M} \rightarrow y(n) \]

Acoustic Plane Waves in Air

- \( x(n) = \text{excess pressure} \) at time \( nT \), at some fixed point \( p_x \in \mathbb{R}^3 \) 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 \textit{time sampling interval} in seconds
  - \( c \) denotes the \textit{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

- \( x(n) = \text{displacement} \) 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

\[ x(n) \xrightarrow{z^{-M}} y(n) \]

- 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 = $|G(e^{j\omega T})|^M$
- Filter is linear and time-invariant (LTI)
- Propagation delay and attenuation can now vary with frequency
- For physical passivity, we require
  $$|G(e^{j\omega T})| \leq 1$$
  for all $\omega$. 
Dispersive Traveling-Wave Simulation

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

\[ x(n) \rightarrow z^{-M} \rightarrow A^M(z) \rightarrow y(n) \]

- 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.,
  \[ |A(e^{j\omega T})| \equiv 1, \forall \omega \]
- Note that a delay line is also an allpass filter:
  \[ |e^{j\omega MT}| \equiv 1, \forall \omega \]
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

\[ A(z) = 1 + a_1z^{-1} + a_2z^{-2} + \cdots + a_Nz^{-N} \]

\[ \tilde{A}(z) \triangleq z^{-N}A(z^{-1}) \]

\[ \triangleq a_N + a_{N-1}z^{-1} + \cdots + a_1z^{-(N-1)} + \cdots + z^{-N} \]

- 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:

\[ \Theta(\omega) \triangleq \angle H(e^{j\omega T}) \]

Phase Delay:

\[ P(\omega) \triangleq -\frac{\Theta(\omega)}{\omega} \quad \text{(Phase Delay)} \]

Group Delay:

\[ D(\omega) \triangleq -\frac{d}{d\omega} \Theta(\omega) \quad \text{(Group Delay)} \]

- For a slowly modulated sinusoidal input signal
  \[ x(n) = A(nT) \cos(\omega n T + \phi) \], the output signal is
  \[ y(n) \approx G(\omega) A[nT - D(\omega)] \cdot \cos\{\omega[nT - P(\omega)] + \phi\} \]
  where \( G(\omega) \triangleq |H(e^{j\omega T})| \) is the amplitude response.

- Unwrap phase response \( \Theta(\omega) \) to uniquely define it:
  - \( \Theta(0) \triangleq 0 \) 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 \texttt{unwrap}
Let $x = (x, y, z)$ denote the Cartesian coordinates of a point in 3D space.

- Point source at $x = x_1 = (x_1, y_1, z_1)$
- Listening point at $x = x_2 = (x_2, y_2, z_2)$
- Propagation distance:
  $$r_{12} = \|x_2 - x_1\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Acoustic pressure peak amplitude (or rms level) at $x = x_2$ is given by

$$p(x_2) = \frac{p_1}{r_{12}}$$

where $p_1$ is the peak amplitude (or rms level) at $r_{12} = \|x_2 - x_1\| = 1$

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

\[
I_1 \propto \langle |p_1|^2 \rangle \quad \text{is} \quad I(x_2) = \frac{I_1}{r_{12}^2}
\]

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:**

\[
x(n) \quad \longrightarrow \quad z^{-M} \quad \longrightarrow \quad \frac{1}{r} \quad \longrightarrow \quad y(n)
\]
• 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 $L$

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

• 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.

```c
#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:**
\[ z^{-1} \leftarrow G(z) z^{-1}, \quad |G(e^{j\omega T})| \leq 1 \]

\( G(z) \) 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
\[ y(n) = g * g * \ldots * g * x(n - M) \]
\[ M \text{ times} \]
in the time domain
The intensity of a *plane wave* is observed to decay exponentially according to

\[ I(x) = I_0 e^{-x/\xi} \]

where

- \( I_0 \) = intensity at the plane source (e.g., a vibrating wall)
- \( I(x) \) = intensity \( x \) meters from the plane-source
- \( \xi \) = intensity decay constant (1/e distance in meters) (depends on frequency, temperature, humidity and pressure)

<table>
<thead>
<tr>
<th>Relative Humidity</th>
<th>Frequency in Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>40</td>
<td>5.6</td>
</tr>
<tr>
<td>50</td>
<td>5.6</td>
</tr>
<tr>
<td>60</td>
<td>5.6</td>
</tr>
<tr>
<td>70</td>
<td>5.6</td>
</tr>
</tbody>
</table>

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

*Acoustic Intensity* (a real vector) may be defined by

\[
I \triangleq p v
\]

\[
\left( \frac{\text{Energy Flux}}{\text{Area} \cdot \text{Time}} \right) = \left( \frac{\text{Power Flux}}{\text{Area}} \right)
\]

where

\[ p = \text{acoustic pressure} \quad \left( \frac{\text{Force}}{\text{Area}} \right) \]

\[ v = \text{acoustic particle velocity} \quad \left( \frac{\text{Length}}{\text{Time}} \right) \]

For a *traveling plane wave*, we have

\[ p = R v \]

where

\[ R \triangleq \rho c \]

is called the *wave impedance* of air, and

\[ c = \text{sound speed} \]

\[ \rho = \text{mass density of air} \quad \left( \frac{\text{Mass}}{\text{Volume}} \right) \]

\[ v \triangleq |v| \]

Therefore, in a plane wave,

\[
I \triangleq p v = R v^2 = \frac{p^2}{R}
\]
From 1D+$\mp$ to 1D$\pm$

We have been modeling *unidirectional* traveling waves:

$$x(n) \xrightarrow{} z^{-M} \xrightarrow{} A^M(z) \xrightarrow{} y(n)$$

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

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:

**Right-going traveling-wave samples**

$$z^{-N}$$

**Left-going traveling-wave samples**

$$z^{-N}$$

(Successor, Non-Dispersive Case)
Digital Waveguide Models

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

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

\[ z^{-N} \]

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
• 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 outputs 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, . . . )
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
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

\[ y(n) = b_0 x(n) + b_M x(n - M) \]

Difference Equation

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
- \( H(z) = 1 + gz^{-M} \)
  - \( M = 5 \)
  - \( g = 0.1, 0.5, 0.9 \)
- \( G(\omega) \Delta = |H(e^{j\omega T})| = |1 + ge^{-jM\omega T}| \rightarrow 2\cos(M\omega T/2) \) when \( g = 1 \)
- In flangers, these nulls slowly move with time
Feedback Comb Filter

\[ y(n) = b_0 v(n) \]

\[ y(n) = b_0 x(n) - a_M y(n-M) \]

\[-a_M = \text{Feedback coefficient (need } |a_M| < 1 \text{ for stability)} \]

\[ M = \text{Delay-line length in samples} \]

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

\[ v(n) = x(n) - a_M v(n-M) \]

**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: $b_0 = 1, -a_M = g \Rightarrow$

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

$$H(z) = \frac{1}{1 - g z^{-M}}$$

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

$$H(z) = \frac{1}{1 - g z^{-M}} = 1 + g z^{-M} + g^2 z^{-2M} + \cdots$$

$$\longleftrightarrow \delta(n) + g \delta(n - M) + g^2 \delta(n - 2M) + \cdots$$

$$= [1, 0, \ldots, 0, g, 0, \ldots, 0, g^2, 0, \ldots]$$

- 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

\[ y(n) = x(n) + gy(n - M) \]

\[ H(z) = \frac{1}{1 - gz^{-M}} \]

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
- \( H(z) = \frac{1}{1-gz^{-M}} \)
- \( M = 5, \quad g = 0.1, 0.5, 0.9 \)
- \( G(\omega) \Delta \equiv |H(e^{j\omega T})| = \left| \frac{1}{1-g e^{-jM\omega T}} \right| \xrightarrow{g=1} \frac{1}{2 \sin\left(\frac{M}{2} \omega T\right)} \)
Inverted-Feed-Back Comb-Filter Amplitude Response

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

\[
|H(e^{j\omega T})| = \left| \frac{\overline{a} + e^{-jM\omega T}}{1 + ae^{-jM\omega T}} \right| = \left| \frac{\overline{a} + e^{-jM\omega T}}{e^{jM\omega T} + a} \right|
\]

\[
= \left| \frac{a + e^{jM\omega T}}{a + e^{jM\omega T}} \right| = 1
\]
First-Order Allpass Filter

Transfer function:

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

(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

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

(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