MUS420 Lecture 4A
Interpolated Delay Lines, Ideal Bandlimited Interpolation, and Fractional Delay Filter Design

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

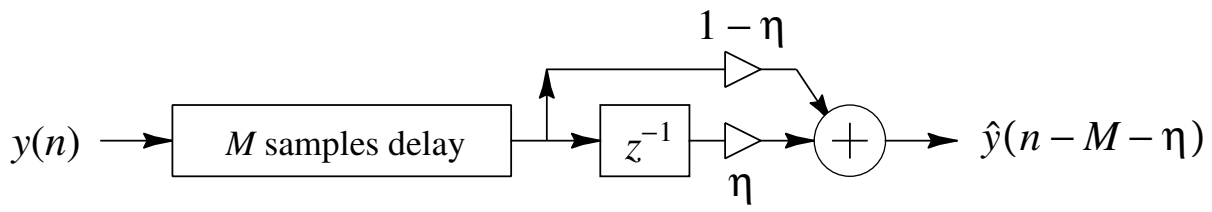
September 5, 2022

Outline

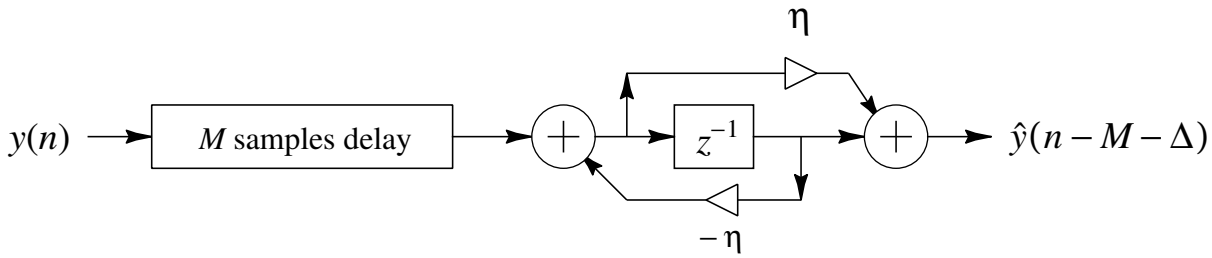
- Low-Order (Fast) Interpolators
  - Linear
  - Allpass
- High-Order Interpolation
  - Ideal Bandlimited Interpolation
  - Windowed-Sinc Interpolation
- High-Order Fractional Delay Filtering
  - Lagrange
  - Farrow Structure
  - Thiran Allpass
- Optimal FIR Filter Design for Interpolation
  - Least Squares
  - Comparison to Lagrange

Simple Interpolators suitable for Real Time Fractional Delay Filtering

Linearly Interpolated Delay Line (1st-Order FIR)



Allpass Interpolated Delay Line (1st-Order)



$$\Delta \approx \frac{1-\eta}{1+\eta}$$

Linear Interpolation

Simplest of all, and the most commonly used:

$$\hat{y}(n-\eta) = (1-\eta)\cdot y(n) + \eta \cdot y(n-1)$$

where $\eta =$ desired fractional delay.

One-multiply form:

$$\hat{y}(n-\eta) = y(n) + \eta \cdot [y(n-1) - y(n)]$$

- Works best with *lowpass* signals
  (Natural spectra tend to roll off rapidly)
- Works well with *over-sampling*
- For faster linear interpolation, prepare a *difference table* containing $y_d(n) = y(n-1) - y(n)$ so that

$$\hat{y}(n-\eta) = y(n) + \eta y_d(n)$$

Deriving Linear Interpolation from Taylor Series

Truncate a *Taylor series expansion* to *first order* and plug in a first-order derivative approximation:

$$\begin{aligned} y(n+\alpha) &= y(n) + \alpha\,\dot{y}(n) + \alpha^2\,\frac{\ddot{y}(n)}{2!} + \alpha^3\,\frac{\dddot{y}(n)}{3!} + \cdots \\ &\approx y(n) + \alpha\,\dot{y}(n) \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad \hat{y}(n+\alpha) &\stackrel{\Delta}{=} y(n) + \alpha\,\frac{y(n+1)-y(n)}{1} \\ &= \alpha\,y(n+1) + (1-\alpha)\,y(n) \end{aligned}$$

where $\alpha = -\eta =$ fractional *advance* desired
(interpolation time between samples n and $n+1$)

- Same approach can be used to define higher-order interpolation filters using various choices of higher-order derivative approximations
- Recall the many finite-difference schemes we have

Examples

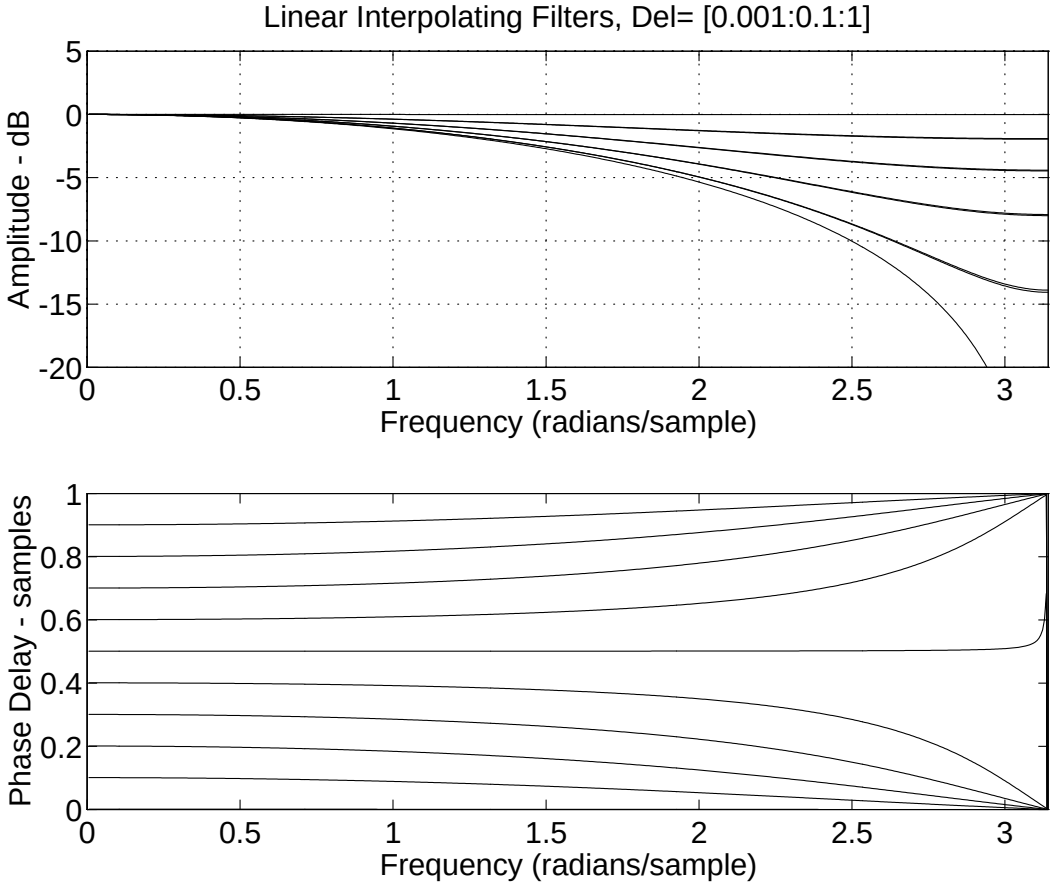
Half-sample delay:

$$\hat{y}\left(n - \frac{1}{2}\right) \;=\; \frac{1}{2} \cdot y(n) + \frac{1}{2} \cdot y(n-1)$$

Quarter-sample delay:

$$\hat{y}\left(n - \frac{1}{4}\right) \;=\; \frac{3}{4} \cdot y(n) + \frac{1}{4} \cdot y(n-1)$$

Frequency Responses of Linear Interpolation for Delays between 0 and 1



Linear Interpolation as a Convolution

- Equivalent to filtering the *continuous-time* weighted impulse train

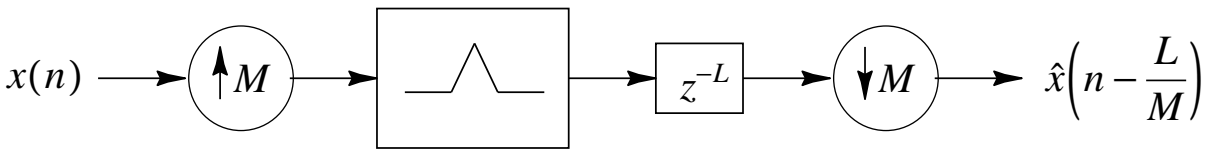
$$\sum_{n=-\infty}^{\infty} y(nT)\delta(t-nT)$$

with the *continuous-time* "triangular pulse" FIR filter

$$h_l(t) = \begin{cases} 1-|t/T|\,, & |t| \le T \\ 0, & \text{otherwise} \end{cases}$$

followed by *sampling* at the desired phase

Discrete-Time Approximation



- Replacing $h_l(t)$ by $h_s(t) \stackrel{\Delta}{=} \text{sinc}\left(\frac{t}{T}\right)$ converts linear interpolation to *ideal bandlimited interpolation* (see "sinc interpolation" below)

First-Order Allpass Interpolation

$$\begin{aligned}\hat{x}(n-\Delta) \stackrel{\Delta}{=} y(n) &= \eta \cdot x(n) + x(n-1) - \eta \cdot y(n-1) \\ &= \eta \cdot [x(n) - y(n-1)] + x(n-1)\end{aligned}$$

$$H(z) = \frac{\eta + z^{-1}}{1 + \eta z^{-1}}$$

- Low frequency delay given by (exact at DC):

$$\Delta \approx \frac{1-\eta}{1+\eta} \quad \Longleftrightarrow \quad \eta \approx \frac{1-\Delta}{1+\Delta}$$

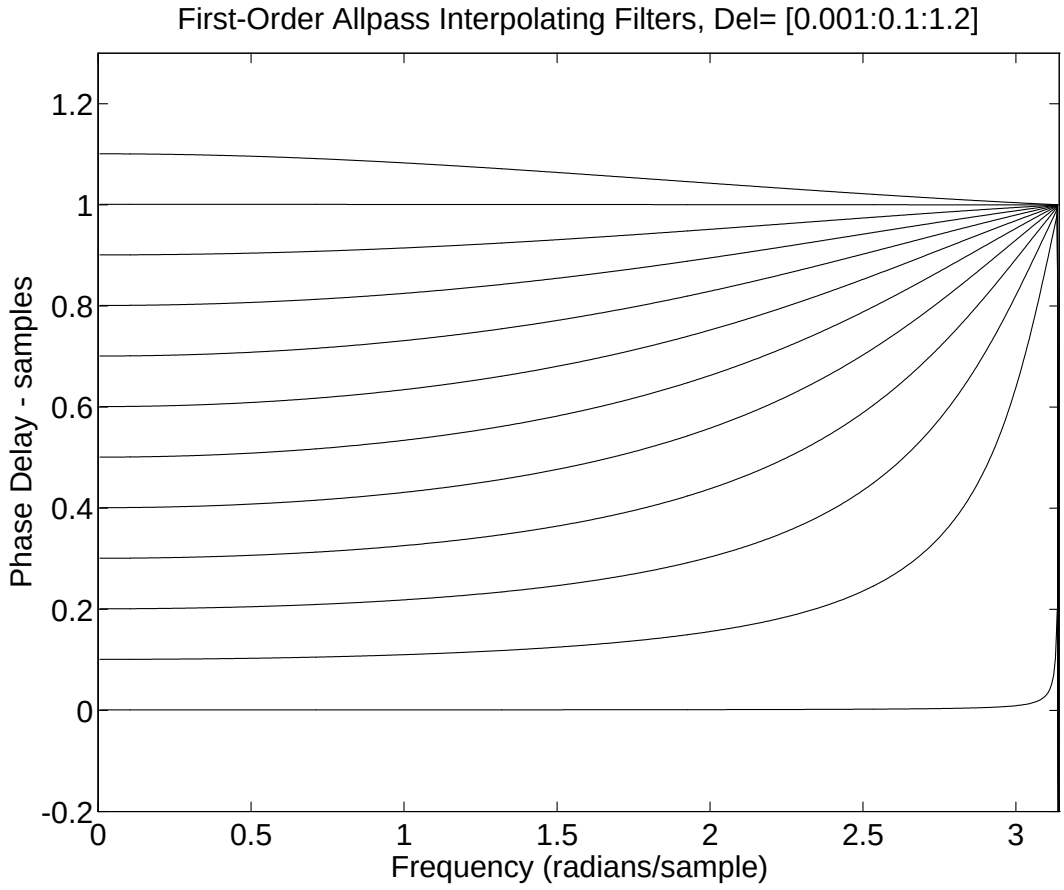
- Same complexity as linear interpolation
- Good for delay-line interpolation, *not* random access
- Best used with *fixed* fractional delay Δ
- To avoid pole near $z = -1$, offset delay range, e.g.,

$$\Delta \in [0.1, 1.1] \;\leftrightarrow\; \eta \in [-0.05, 0.82]$$

- Change delay slowly compared to $\tau \approx T/(1 - \eta_{\mathsf{max}})$

Intuitively, ramping the coefficients of the allpass gradually "grows" or "hides" one sample of delay. This tells us how to handle resets when crossing sample boundaries (sufficiently slowly).

Phase Delays of First-Order Allpass Interpolators for Various Desired Delays



Interpolation Overview

Well Known Closed-Form Solutions

Order

	1	N	Large N	∞
FIR	Linear	Lagrange	Windowed Sinc	
IIR	Allpass_1	Thiran		Sinc

Tabulated Alternative (Order N)

Design a digital filter (FIR or IIR) that approximates

$$H_\Delta\left(e^{j\omega T}\right) = e^{-j\omega\Delta T}, \quad \Delta = \text{Desired delay in samples}$$

optimally in some sense, with coefficients tabulated over a range including Δ samples (and interpolated on lookup).

Itinerary Below

- Ideal Bandlimited (Sinc) Interpolation
- Windowed Sinc Interpolation (still *perceptually* ideal up to some band edge)
- Lagrange FIR (polynomial) Interpolation
- Thiran IIR Interpolation

Ideal Bandlimited (Sinc) Interpolation

Ideal interpolation for digital audio is *bandlimited interpolation*, i.e., samples are *uniquely* interpolated based on the assumption of zero spectral energy for $|f| \geq f_s/2$.

Ideal *bandlimited interpolation* is *sinc interpolation*:

$$y(t) = (y * h_s)(t) = \sum_{n=-\infty}^{\infty} y(nT) h_s(t - nT)$$

where

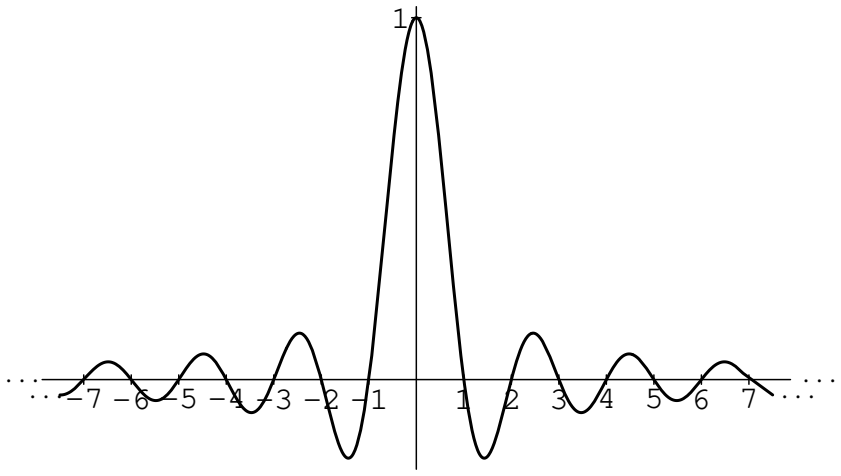
$$h_s(t) \stackrel{\Delta}{=} \mathrm{sinc}(f_s t) \stackrel{\Delta}{=} \mathrm{sinc}\left(\frac{t}{T}\right)$$

$$\mathrm{sinc}(x) \stackrel{\Delta}{=} \frac{\sin(\pi x)}{\pi x} \qquad \text{(sinc function)}$$

(Proof: sampling theorem)

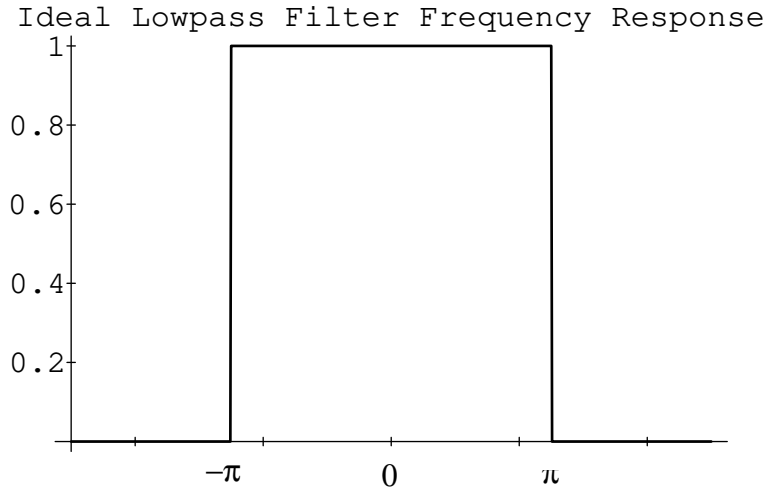
The Sinc Function ("Cardinal Sine")

$$\mathrm{sinc}(t) \stackrel{\Delta}{=} \begin{cases} \frac{\sin(\pi t)}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases}$$



Sinc Function

The sinc function is the impulse response of the ideal lowpass filter which cuts off at half the sampling rate



Applications of Bandlimited Interpolation

Bandlimited Interpolation is used in (e.g.)

- Sampling-rate conversion
- Oversampling D/A converters
- Wavetable/sampling synthesis
- Virtual analog synthesis
- Fractional delay filtering

Fractional delay filtering is a *special case* of bandlimited interpolation:

- Fractional delay filters only need *sequential access* $\Rightarrow$ *IIR filters* can be used
- General bandlimited-interpolation requires *random access* $\Rightarrow$ *FIR filters* normally used

Fractional Delay Filters are used for (among other things)

- Time-varying delay lines (flanging, chorus, leslie)
- Tuning digital waveguide models to correct pitch
- Exact tonehole placement in woodwind models
- Beam steering of microphone / speaker arrays

Ideal D/A Conversion

Each sample in the time domain scales and locates one *sinc function* in the unique, continuous, bandlimited interpolation of the sampled signal.

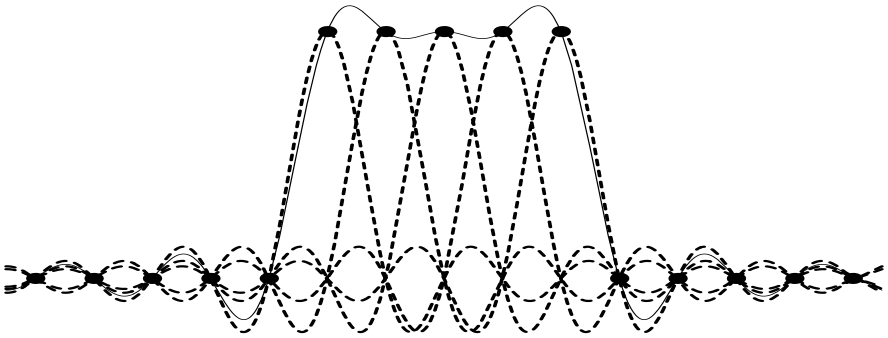
Convolving a sampled signal $y(n)$ with $\mathsf{sinc}(n-\eta)$ "evaluates" the signal at an arbitrary continuous time $\eta \in \mathbb{R}$:

$$y(\eta) \;=\; \sum_{n=-\infty}^{\infty} y(n)\mathsf{sinc}(\eta - n)$$

Proof: Sampling Theorem

Ideal D/A Example

Reconstruction of a bandlimited rectangular pulse $x(t)$ from its samples $x = [\ldots, 0, 1, 1, 1, 1, 1, 0, \ldots]$:



Bandlimited Rectangular Pulse Reconstruction from its Samples

Catch

- Sinc function is infinitely long and noncausal
- Must be available in *continuous* form

Supplementary: Optimal Least Squares Bandlimited Interpolation Formulated as a Fractional Delay Filter

Consider a filter which delays its input by Δ samples:

- Ideal impulse response = *bandlimited delayed impulse* = *delayed sinc*

$$h_\Delta(t) \;=\; \mathrm{sinc}(t-\Delta) \;\stackrel{\Delta}{=}\; \frac{\sin[\pi(t-\Delta)]}{\pi(t-\Delta)}$$

- Ideal frequency response = *"brick wall" lowpass* response, cutting off at $f_s/2$ and having *linear phase* $e^{-j\omega\Delta T}$

$$H_\Delta(\omega) \;\stackrel{\Delta}{=}\; \mathrm{FT}(h_\Delta) \;=\; \begin{cases} e^{-j\omega\Delta}, & |\omega| < \pi f_s \\ 0, & |\omega| \geq \pi f_s \end{cases}$$

$$\rightarrow \quad H_\Delta(e^{j\omega T}) \;=\; e^{-j\omega\Delta T}, \quad -\pi \leq \omega T < \pi$$

$$\leftrightarrow\; \mathrm{sinc}(n-\Delta), \quad n = 0, \pm 1, \pm 2, \ldots$$

  after critically sampling in the time domain.

The sinc function is an infinite-impulse-response (IIR) digital filter with no recursive form $\Rightarrow$ *non-realizable*.

To obtain a *finite* impulse response (FIR) interpolating filter, let's formulate a *least-squares filter-design problem:*

Desired Interpolator Frequency Response

$$H_\Delta\left(e^{j\omega T}\right) \;=\; e^{-j\omega\Delta T}, \quad \Delta \;=\; \text{Desired delay in samples}$$

FIR Frequency Response, Zero-Phase Alignment

$$\hat{H}_\Delta\left(e^{j\omega T}\right) \;=\; \sum_{n=-\frac{L-1}{2}}^{\frac{L-1}{2}} \hat{h}_\Delta(n) e^{-j\omega n T}$$

Error to Minimize

$$E\left(e^{j\omega T}\right) = H_\Delta\left(e^{j\omega T}\right) - \hat{H}_\Delta\left(e^{j\omega T}\right)$$

L_2 Error Norm

$$\begin{aligned} J(\underline{h}) \stackrel{\Delta}{=} \| E \|_2^2 \;&=\; \frac{T}{2\pi}\int_{-\pi/T}^{\pi/T} \left|E\left(e^{j\omega T}\right)\right|^2 d\omega \\ &=\; \frac{T}{2\pi}\int_{-\pi/T}^{\pi/T} \left|H_\Delta\left(e^{j\omega T}\right) - \hat{H}_\Delta\left(e^{j\omega T}\right)\right|^2 d\omega \end{aligned}$$

By Parseval's Theorem

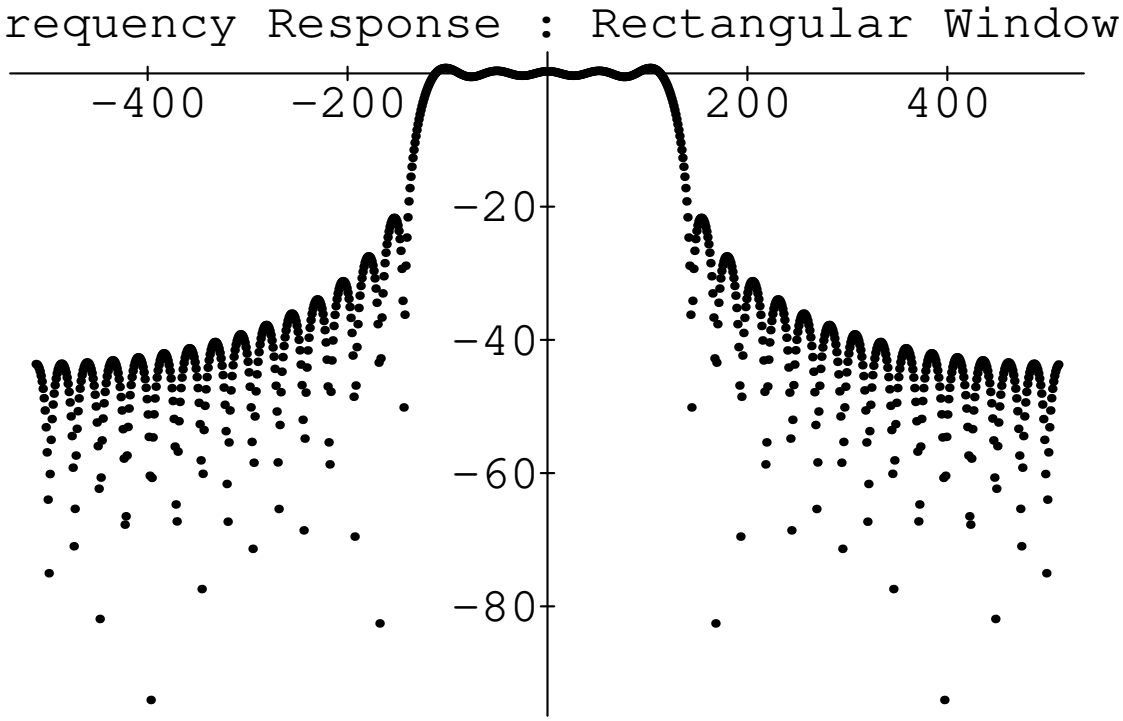
$$J(\underline{h}) = \sum_{n=0}^{\infty} \left|h_\Delta(n) - \hat{h}_\Delta(n)\right|^2$$

Optimal Least-Squares FIR Interpolator

$$\hat{h}_\Delta(n) = \begin{cases} \text{sinc}(n-\Delta), & \frac{L-1}{2} \le n \le \frac{L-1}{2} \\ 0, & \text{otherwise} \end{cases}$$

Truncated-Sinc Interpolation

Truncate sinc(t) at 5th zero-crossing to left and right of time 0 to get the following amplitude response:



Truncated-Sinc Transform

- Vertical axis in dB, horizontal axis in spectral samples
- Optimal in least-squares sense
- Poor stop-band rejection (≈ 20 dB)
- "Gibbs Phenomenon" gives too much "ripple"
- Ripple can be reduced by *tapering* the sinc function to zero instead of simply truncating it

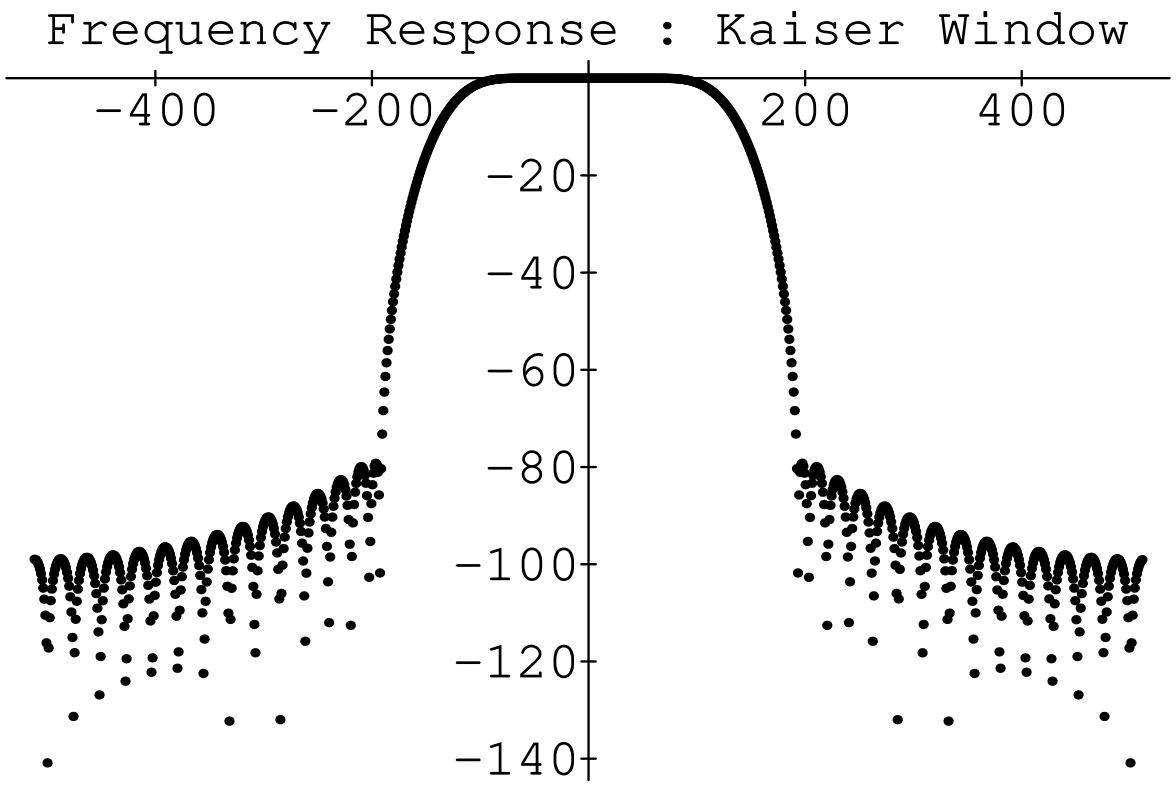
Windowed Sinc Interpolation

- Sinc function can be *windowed* more generally to yield

$$\hat{h}_\Delta(n) = \begin{cases} w(n-\Delta)\mathrm{sinc}[\alpha(n-\Delta)], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

- Example of *window method* for FIR lowpass filter design applied to sinc functions (ideal lowpass filters) sampled at various phases (corresponding to desired delay between samples)
- For best results, $\Delta \approx L/2$ (causal length L window)
- $w(n)$ is any real symmetric window (e.g., Hamming, Blackman, Kaiser)
- Non-rectangular windows *taper* truncation which *reduces* Gibbs phenomenon, as in FFT analysis
- $\alpha < 1$ provides for a nonzero *transition band*

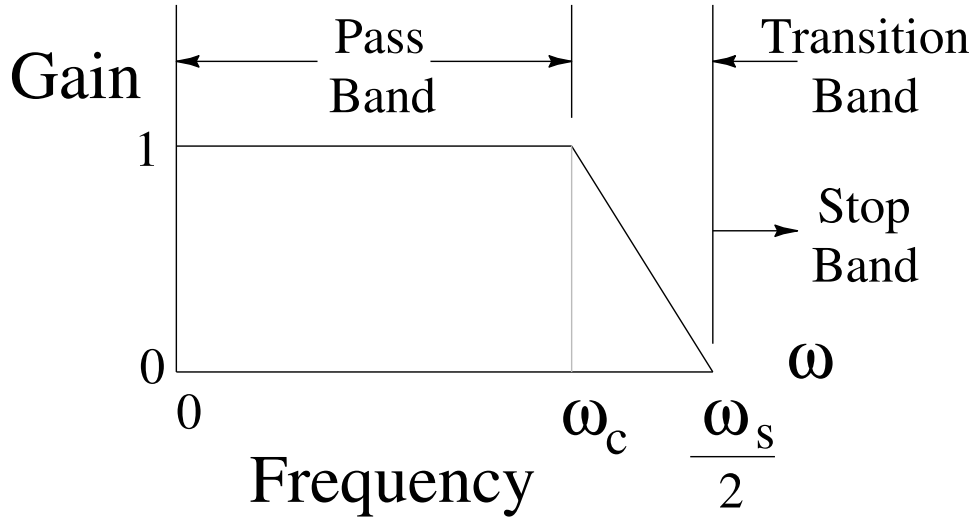
Spectrum of Kaiser-windowed Sinc



Kaiser-Windowed Sinc Transform

- Stopband now starts out close to -80 dB
- Kaiser window has a single parameter which trades off stop-band attenuation versus transition-bandwidth from pass-band to stop-band

Lowpass Filter Design



Lowpass Filter Design Parameters

- In the *transition band*, frequency response "rolls off" from 1 at $\omega_c = \alpha\omega_s/2$ to zero (or ≈ 0.5) at $\omega_s/2$
- Interpolation can be "perfect" in pass-band

Online references (FIR interpolator design)

- Music 421 Lecture 2 on Windows[1]
- Music 421 Lecture 3 on FIR Digital Filter Design[2]
- Optimal FIR Interpolator Design[3]

[1] http://ccrma.stanford.edu/~jos/Windows/
[2] http://ccrma.stanford.edu/~jos/WinFlt/
[3] http://ccrma.stanford.edu/~jos/resample/optfir.pdf

Oversampling Reduces Filter Length

- Example 1:
  - $f_s = 44.1$ kHz (CD quality)
  - Audio upper limit $= 20$ kHz
  - *Transition band* $= 2.05$ kHz
  - *FIR filter length* $\stackrel{\Delta}{=} L_1$
- Example 2:
  - $f_s = 48$ kHz (e.g., DAT)
  - Audio upper limit $= 20$ kHz
  - *Transition band* $= 4$ kHz
  - *FIR filter length* $\approx L_1/2$
- Required FIR filter length varies inversely with transition bandwidth
  $\Rightarrow$ Required filter length in example 1 is almost *double* ($\approx 4/2.1$) the required filter length for example 2
- Increasing the sampling rate by less than ten percent reduces the filter expense by almost fifty percent in this example

The Digital Audio Resampling Home Page

- C++ software for windowed-sinc interpolation
- C++ software for FIR filter design by window method
- Fixed-point data and filter coefficients
- Can be adapted to time-varying resampling
- Open source, free
- First written in 1983 in SAIL
- URL:
  `http://ccrma.stanford.edu/~jos/resample/`
- Interesting comparisons of various audio interpolators out there:
  `http://src.infinitewave.ca/`
- *Most needed upgrade*:
  - Design and install a set of *optimal* FIR interpolating filters.[4]

[4] `http://ccrma.stanford.edu/~jos/resample/optfir.pdf`

Interpolator Types

	Order			
	1	N	Large N	∞
FIR	Linear	Lagrange	Windowed Sinc	
IIR	Allpass_1	Thiran		Sinc

Lagrange Interpolation

- Lagrange interpolation is just *polynomial interpolation*
- Nth-order polynomial interpolates $N+1$ points
- First-order case $=$ *linear interpolation*

Problem Formulation

Given a set of $N+1$ known samples $f(x_k)$,
$k = 0, 1, 2, \ldots, N$, find the *unique* order N *polynomial* $y(x)$ which *interpolates* the samples

Solution (Waring, Lagrange):

$$y(x) = \sum_{k=0}^{N} l_k(x) f(x_k)$$

where $l_k(x)$ is the *Lagrange polynomial* corresponding to sample x_k:

$$l_k(x) \stackrel{\Delta}{=} \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_N)}$$

- Numerator gives a *zero* at all samples but the kth
- Denominator simply *normalizes* $l_k(x)$ to 1 at $x = x_k$

- As a result,

$$l_k(x_j) = \delta_{kj} \stackrel{\Delta}{=} \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$$

- Generalized bandlimited impulse = generalized sinc function:
  Each $l_k(x)$ goes through 1 at $x = x_k$ and zero at all other sample points
  I.e., $l_k(x)$ is analogous to sinc$(x - x_k)$
- For uniformly spaced samples, Lagrange interpolaton converges to *sinc* interpolation as $N \to \infty$
- For uniformly spaced samples and *finite* N, Lagrange interpolaton is equivalent to *windowed sinc* interpolation using a *binomial window*
  (see text for refs)
- Nonuniformly spaced sample locations, such as along the zeros of a Chebyshev polynomial, generally do better than uniform spacing, when applicable

Lagrange Interpolation Optimality

In the uniformly sampled case, Lagrange interpolation can be viewed as an FIR filter with coefficients that depend on the fractional part of the "interpolation time" (sample times on the integers)

- Lagrange interpolation filters are *maximally flat* in the frequency domain about dc:

  $$\left.\frac{d^m E(e^{j\omega})}{d\omega^m}\right|_{\omega=0} = 0, \quad m = 0, 1, 2, \ldots, N,$$

  (1st $N+1$ terms of Taylor expansion about $\omega = 0$ zeroed), where

  $$E(e^{j\omega}) \stackrel{\Delta}{=} e^{-j\omega\Delta} - \sum_{n=0}^{N} h_\Delta(n) e^{-j\omega n}$$

  and Δ is the desired delay in samples
  (see text for proof)

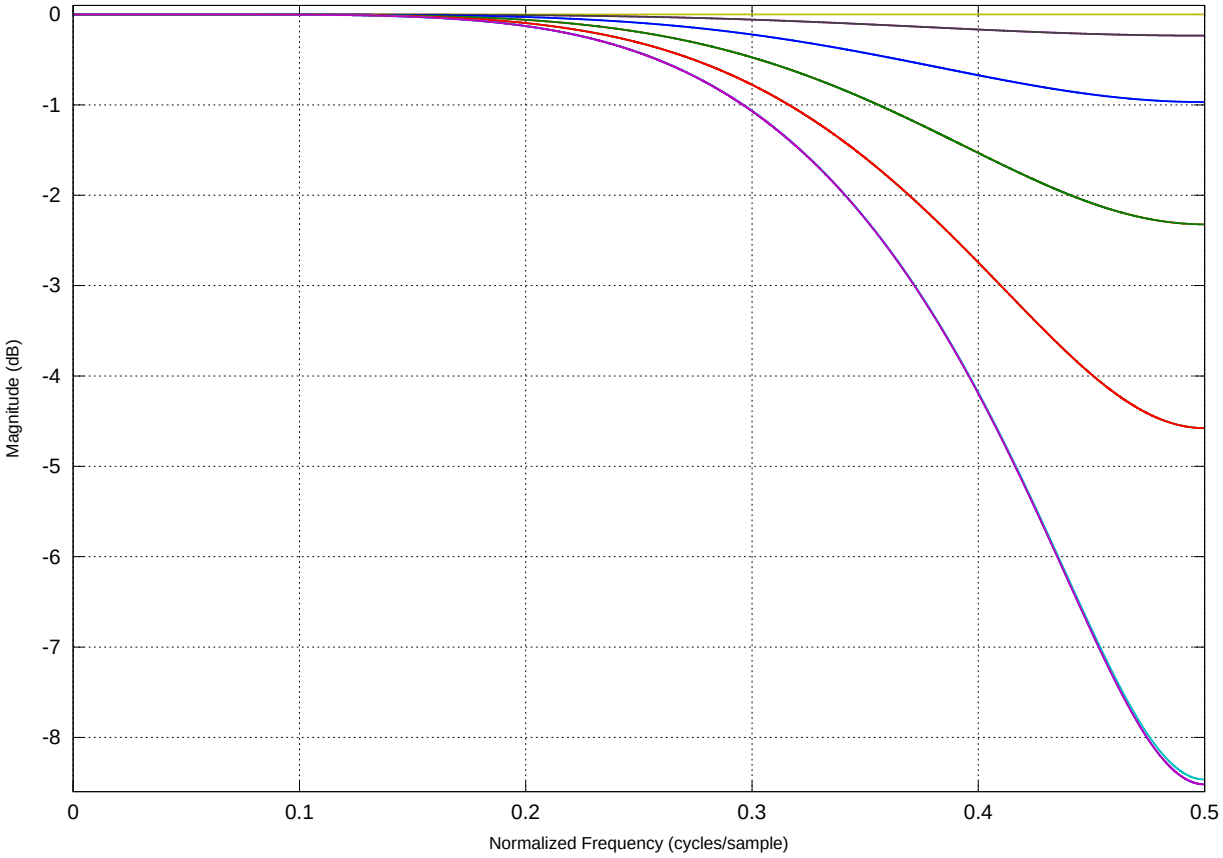
- Same optimality criterion as *Butterworth filters* in classical analog filter design

- Can also be viewed as "Padé approximation" to a constant frequency response

- Also *bounded by 1* in the frequency domain:

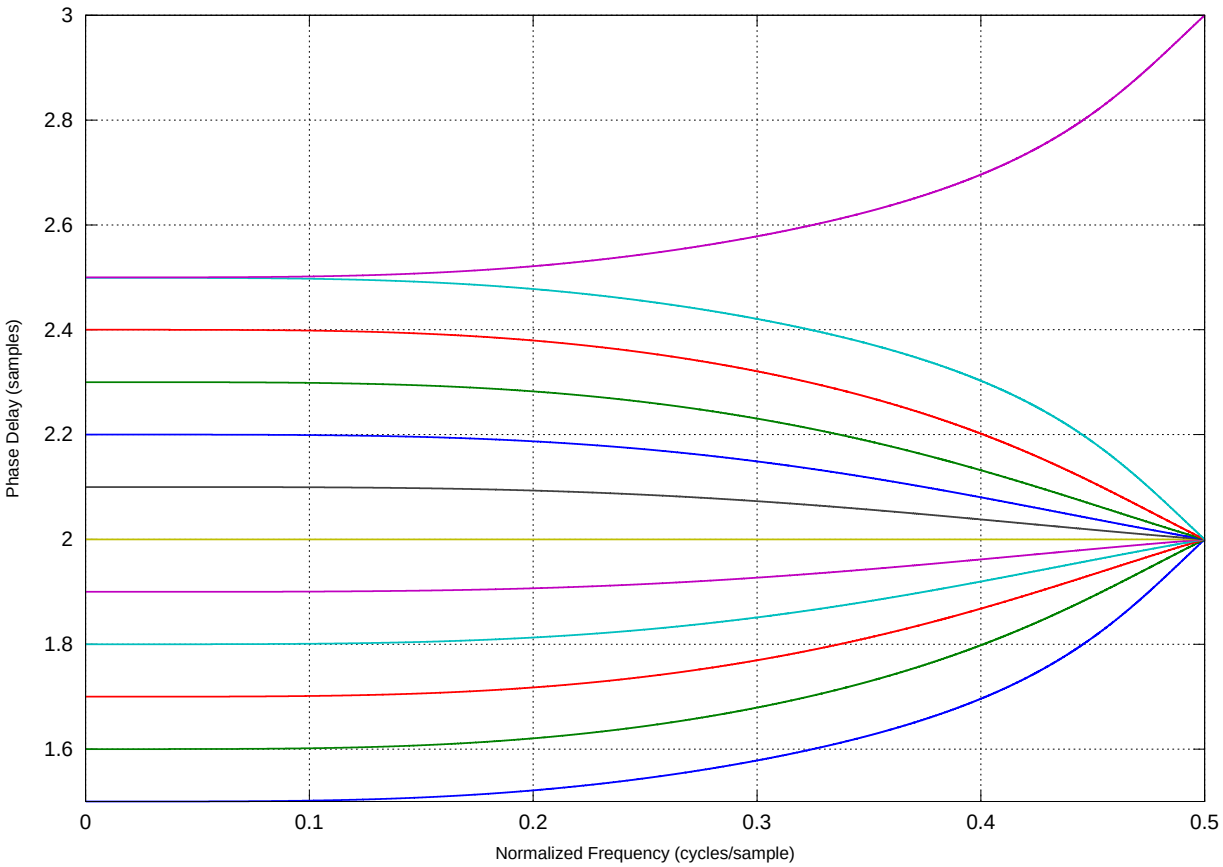
  $$|H(e^{j\omega T})| \leq 1, \quad \forall\, \omega T \in [-\pi, \pi]$$

Order 4 Amplitude Response Over a Range of Fractional Delays



$$\Delta = 1.5{:}0.1{:}2.5$$

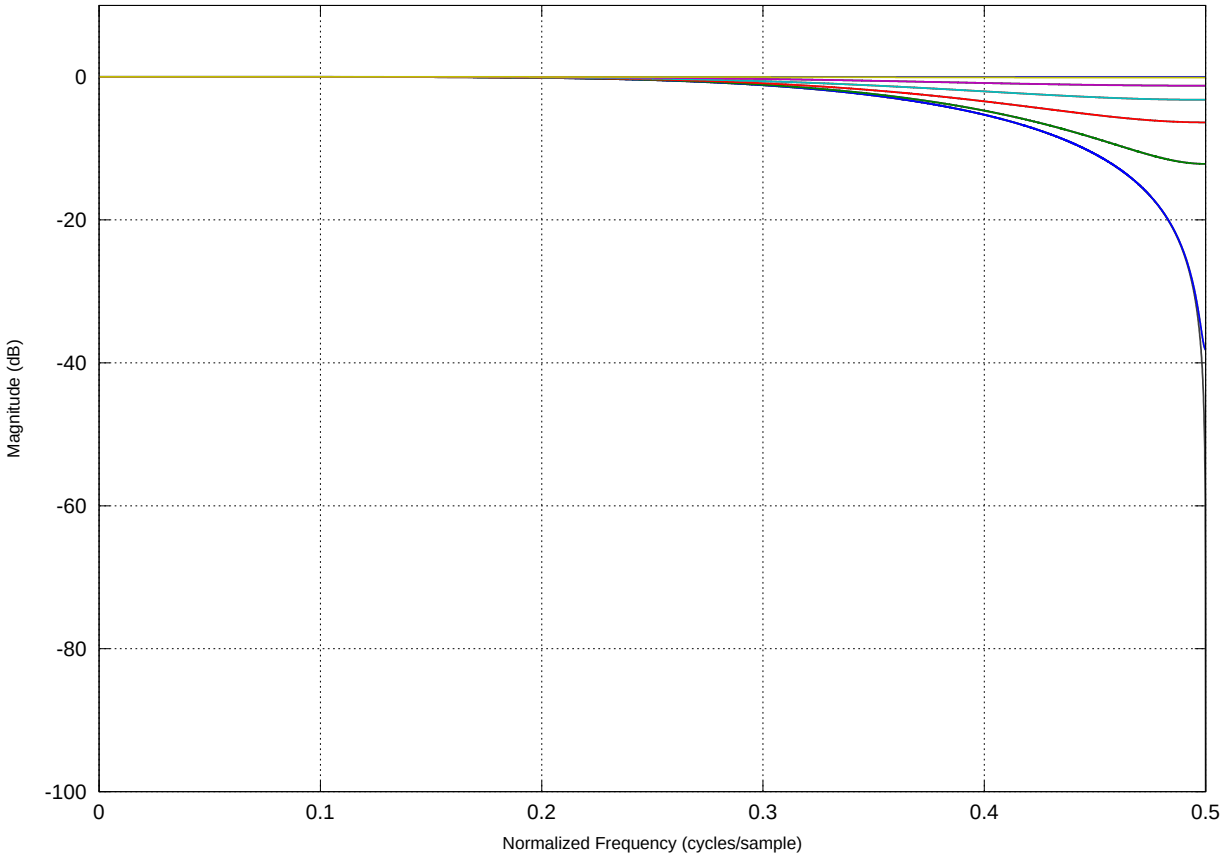
Order 4 Phase Delay Over a Range of Fractional Delays



$\Delta = 1.5 : 0.1 : 2.5$ plus 2.499

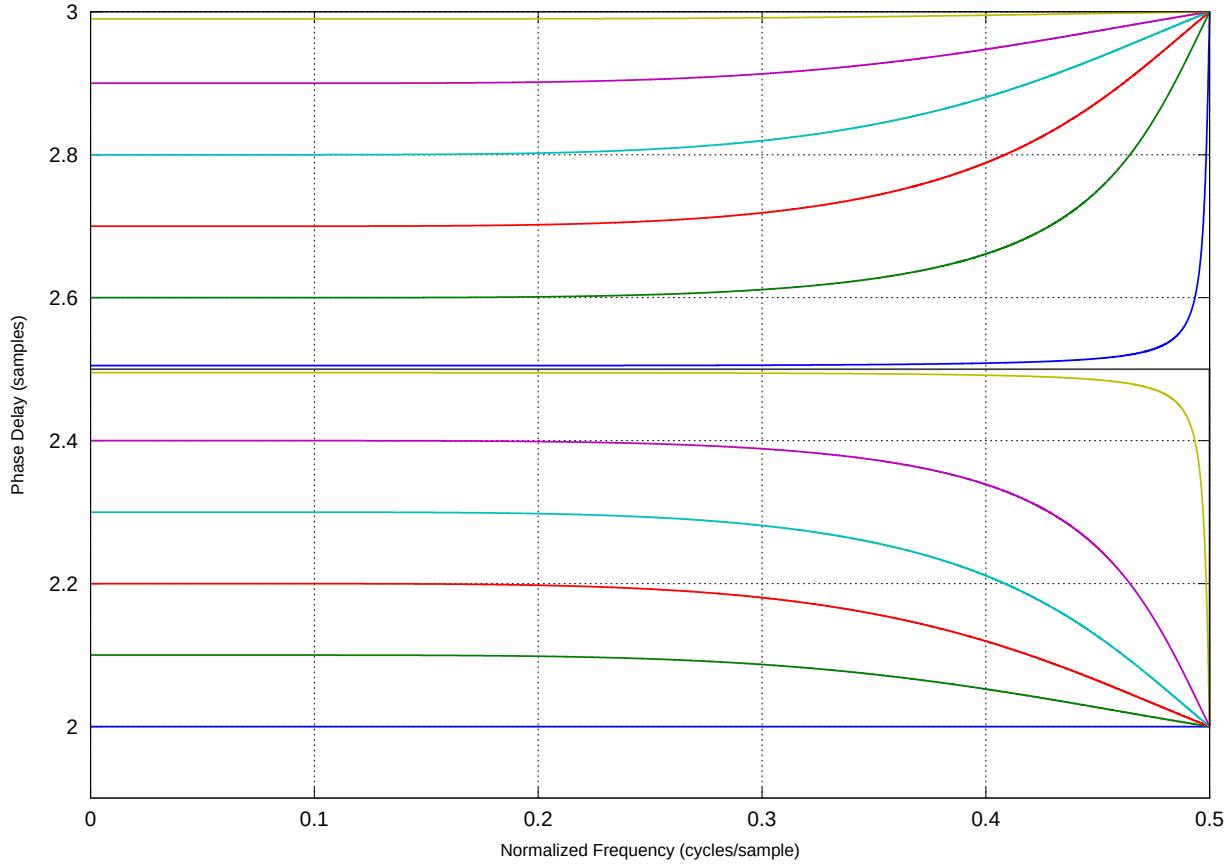
- Requested delay 2.5 veers off to 3 samples at $f_s/2$
- Requested delay 2.499 swings down to 2 samples at $f_s/2$ like the other delay curves
- Preferable to use modulated delays converging to the same integer at $f_s/2$ (possible over a one-sample modulation range)

Order 5 Amplitude Response Over a Range of Fractional Delays



$\Delta = 2.0 : 0.1 : 3.0$ plus 2.495 and 2.505

Order 5 Phase Delay Over a Range of Fractional Delays

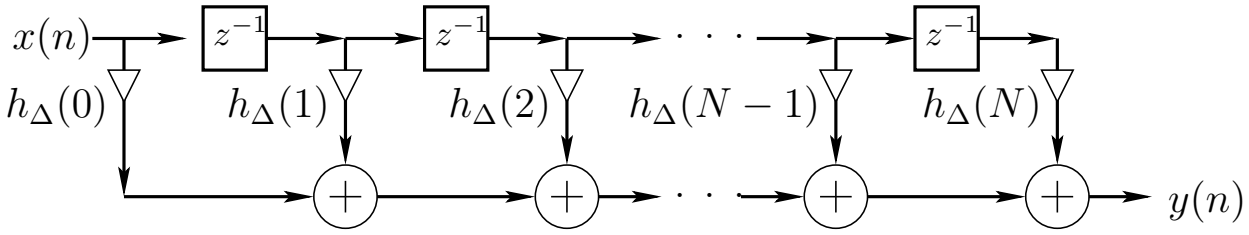


$\Delta = 2.0 : 0.1 : 3.0$ plus 2.495 and 2.505

- Notice sudden jump to an integer delay at $f_s/2$ for curves near delay 2.5 samples
  - Phase delay for 2.495 samples swings down to 2
  - Phase delay for 2.505 samples swings up to 3
- Phase delay for 2.5 samples can be considered exact (gain=0 at $f_s/2$)

Explicit Formula for Lagrange Interpolation Coefficients

$$h_\Delta(n) = \prod_{\substack{k=0 \\ k \neq n}}^{N} \frac{\Delta - k}{n - k}, \quad n = 0, 1, 2, \ldots, N$$



Lagrange Interpolation Coefficients Orders 1, 2, and 3

$h_\Delta Order$	$h_\Delta(0)$	$h_\Delta(1)$	$h_\Delta(2)$	$h_\Delta(3)$
$N = 1$	$1 - \Delta$	Δ		
$N = 2$	$\frac{(\Delta-1)(\Delta-2)}{2}$	$-\Delta(\Delta - 2)$	$\frac{\Delta(\Delta-1)}{2}$	
$N = 3$	$-\frac{(\Delta-1)(\Delta-2)(\Delta-3)}{6}$	$\frac{\Delta(\Delta-2)(\Delta-3)}{2}$	$-\frac{\Delta(\Delta-1)(\Delta-3)}{2}$	$\frac{\Delta(\Delta-1)(\Delta-2)}{6}$

- For $N = 1$, Lagrange interpolation reduces to *linear interpolation* $h = [1 - \Delta, \Delta]$, as before
- For order N, desired delay should be in a one-sample range centered about $\Delta = N/2$

Matlab Code For Lagrange Fractional Delay

```
function h = lagrange(N, delay)
%LAGRANGE  h=lagrange(N,delay) returns order N FIR
%          filter h which implements given delay
%          (in samples).  For best results,
%          delay should be near N/2 +/- 1.
n = 0:N;
h = ones(1,N+1);
for k = 0:N
    index = find(n ~= k);
    h(index) = h(index) *  (delay-k)./ (n(index)-k);
end
```

Faust Code For Lagrange Fractional Delay

For an intro to Faust, see, *e.g.*,

http://ccrma.stanford.edu/~jos/spf/

```
// Fourth-order case - delay d should be at least 1.5

fdelay4(n,d,x) =
    delay(n,id,x) * fdm1*fdm2*fdm3*fdm4/24
  + delay(n,id+1,x) * (0-fd*fdm2*fdm3*fdm4)/6
  + delay(n,id+2,x) * fd*fdm1*fdm3*fdm4/4
  + delay(n,id+3,x) * (0-fd*fdm1*fdm2*fdm4)/6
  + delay(n,id+4,x) * fd*fdm1*fdm2*fdm3/24
with {
  o = 1.49999;
  dmo = d - o; // assumed nonnegative
  id = int(dmo);
  fd = o + frac(dmo);
  fdm1 = fd-1;
  fdm2 = fd-2;
  fdm3 = fd-3;
  fdm4 = fd-4;
};
```

Faust-Generated C++ Code

Inner-loop function generated by Faust:

```
virtual void
  compute (int count, float** input, float** output)
{
  float* output0 = output[0];
  for (int i=0; i<count; i++) {
    iVec0[0] = 1;
    iRec0[0] = ((1 + iRec0[1]) & 15);
    ftbl0[iRec0[0]] = (1 - iVec0[1]);
    output0[i] =
         ((((0.468750f * ftbl0[((iRec0[0] - 1) & 15)])
      + (2.343750e-02f * ftbl0[((iRec0[0] - 4) & 15)]))
      + (0.703125f * ftbl0[((iRec0[0] - 2) & 15)]))
      - ((0.156250f * ftbl0[((iRec0[0] - 3) & 15)])
      + (3.906250e-02f * ftbl0[(iRec0[0] & 15)])));
    // post processing
    iRec0[1] = iRec0[0];
    iVec0[1] = iVec0[0];
  }
}
```

Test Program (Faust 0.9.9.3):

```
import("filter.lib");
N = 16;
impulse = 1-1';
process  = impulse : fdelay4(N,1.5);
```

Faust Test Program for Generating Above Frequency-Response Examples

```
import("filter.lib"); % Faust v0.9.9.3
N = 16;  % Allocated delay-line length
D = 5.4; % Requested delay
process  = impulse <: (
                        fdelay4(N, 1.5),
                        fdelay4(N, 1.6),
                        fdelay4(N, 1.7),
                        ...
                        fdelay4(N, 2.4),
                        fdelay4(N, 2.499),
                        fdelay4(N, 2.5));
// To see amplitude responses (for example):
// [in a shell]:
//   faust2octave tlagrange.dsp
// [at the Octave command prompt]:
//   plot(db(fft(faustout,1024)(1:512,:)));

// Alternate example for testing a range of orders
// process = 1-1' <: (fdelay1(N,D),
//                    fdelay2(N,D),
//                    fdelay3(N,D),
//                    fdelay4(N,D),
//                    fdelay5(N,D));
```

Relation of Lagrange Interpolation to Windowed Sinc Interpolation

- For an *infinite* number of *equally spaced* samples, with spacing $x_{k+1} - x_k = \Delta$, the Lagrange-interpolation basis polynomials converge to shifts of the *sinc function*, i.e.,

  $$l_k(x) = \text{sinc}\left(\frac{x - k\Delta}{\Delta}\right), \quad k = \ldots, -2, -1, 0, 1, 2, \ldots$$

  Proof: As order $\to \infty$, the binomial window $\to$ Gaussian window $\to$ constant (unity).

Variable FIR Interpolating Filter

Basic idea: Each FIR filter coefficient h_n becomes an order N_c *polynomial* in the delay parameter Δ:

$$\begin{aligned}
h_\Delta(n) &\stackrel{\Delta}{=} \sum_{m=0}^{N_c} c_n(m)\Delta^m, \quad n = 0, 1, 2, \ldots, N_h \\
\Leftrightarrow \; H_\Delta(z) &\stackrel{\Delta}{=} \sum_{n=0}^{N_h} h_\Delta(n) z^{-n} \\
&= \sum_{n=0}^{N_h} \left[\sum_{m=0}^{N_c} c_n(m)\Delta^m \right] z^{-n} \\
&= \sum_{m=0}^{N_c} \left[\sum_{n=0}^{N_h} c_n(m) z^{-n} \right] \Delta^m \\
&\stackrel{\Delta}{=} \sum_{m=0}^{N_c} C_m(z)\Delta^m
\end{aligned}$$

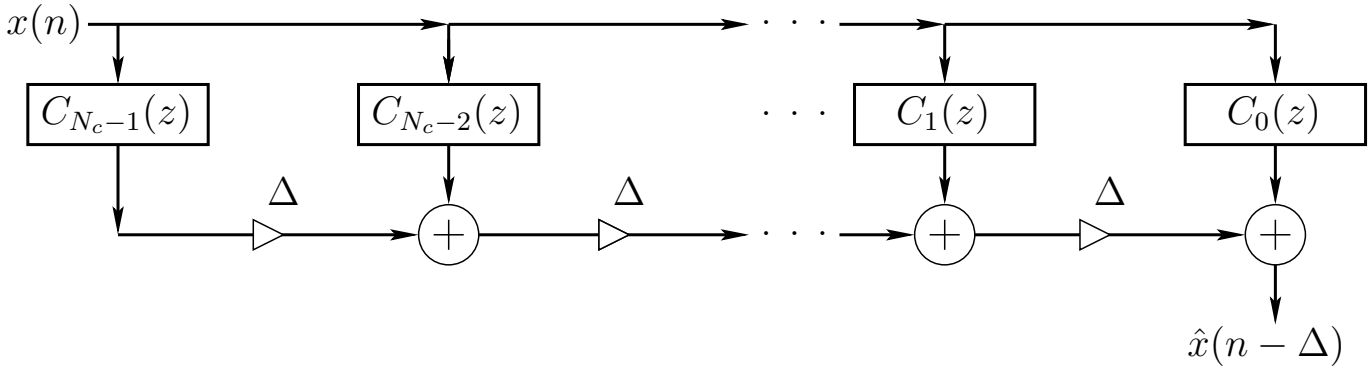
- More generally: $H_\Delta(x) = \sum_m \alpha(\Delta) C_m(z)$ where $\alpha(\Delta)$ is provided by a *table lookup*
- Basic idea applies to any *one-parameter* filter variation
- Also applies to *time-varying* filters ($\Delta \leftarrow t$)

Farrow Structure for Variable Delay FIR Filters

Evaluate polynomial in Δ using *Horner's rule*:

$$\hat{X}_{n-\Delta}(z) = C_0X + \Delta\left[C_1X + \Delta\left[C_2X + \cdots + \Delta\left[C_3X + \cdots\right]\right]\right],$$

$\Rightarrow$ filter structure becomes



As delay Δ varies, "basis filters" $C_k(z)$ remain *fixed*
$\Rightarrow$ *very convenient for changing* Δ over time

Farrow Structure Design Procedure

Solve the N_Δ equations

$$z^{-\Delta_i} = \sum_{k=0}^{N} C_k(z)\Delta_i^k, \quad i = 1, 2, \ldots, N_\Delta$$

for the $N_h + 1$ FIR transfer functions $C_k(z)$, each order N_c in general

References: Laakso et al., Farrow

Farrow Interpolation Features

- The Farrow structure computes Nth-order Lagrange interpolation using $\mathcal{O}(N^2)$ multiplies and adds
- This is the same complexity as plain FIR filtering using the closed-form coefficient formulas above
- However, the Farrow structure is more convenient for variable-delay applications because it is a linear combination of N *fixed* FIR filters $C_k(z)$, $k = 1, \ldots, N$, with linear combination coefficients given by Δ^k, computed recursively, where Δ is the desired delay:

$$\hat{x}(n-\Delta) = x(n)+\Delta\cdot[(c_1 * x)(n)+\Delta\cdot[(c_2 * x)(n)+\cdots$$

- The next section describes a formulation achieving complexity $\mathcal{O}(N)$

Lagrange Interpolation by Taylor Expansion

The transfer function of a Δ-sample delay is

$$H_\Delta(z) = z^{-\Delta}$$

- Nth-order Lagrange interpolation approximates $H_\Delta(z)$ with a length $N+1$ FIR filter

$$\hat{H}_\Delta(z) = h_\Delta(0) + h_\Delta(1)\,z^{-1} + \cdots + h_\Delta(N)\,z^{-N}$$

- We know Lagrange interpolation is maximally flat about dc
- Maximally flat means using *all degrees of freedom* $\{h_\Delta(n)\}_{n=0}^{N}$ to *zero leading terms in the Taylor series expansion* of the frequency-response error $E(z) = H_\Delta(z) - \hat{H}_\Delta(z)$ about dc ($z = 1$)
- This means *matching* the first $N+1$ terms of the Taylor expansion of $H_\Delta(z) = z^{-\Delta}$ about $z = 1$

Taylor Series Expansion of $z^{-\Delta} \triangleq q^{\Delta}$

To obtain a causal FIR filter, we will expand $H_\Delta(z)$ in powers of z^{-1} instead of z. For simplicity of notation, define $q = z^{-1}$. Then we obtain the Taylor series expansion of $H_\Delta(q^{-1}) = q^\Delta$ about $q = 1$ to be

$$\begin{aligned} H_\Delta(q^{-1}) &= H_\Delta(1) + H'_\Delta(1)(q-1) + \frac{1}{2}H''_\Delta(1)(q-1)^2 + \frac{1}{3!}H'''_\Delta(1)(q-1)^3 + \cdots \\ &= 1 + \Delta\, q^{\Delta-1}\big|_{q=1}(q-1) + \frac{1}{2}\Delta(\Delta-1)\, q^{\Delta-2}\big|_{q=1}(q-1)^2 + \cdots \\ &= 1 + \Delta\cdot(q-1) + \frac{1}{2}\Delta(\Delta-1)\cdot(q-1)^2 + \frac{1}{3!}\Delta(\Delta-1)(\Delta-2)\cdot(q-1)^3 + \cdots \end{aligned}$$

where the derivatives are with respect to $q = z^{-1}$, e.g., $H'_\Delta(q) \triangleq \frac{d}{dq}q^\Delta = \Delta q^{\Delta-1}$. Our maximally flat Nth-order Langrange FIR interpolation filter is obtain by truncating this expansion at order N:

$$\begin{aligned} \hat{H}_\Delta(q^{-1}) &\triangleq 1 + \Delta(q-1) + \frac{1}{2}\Delta(\Delta-1)(q-1)^2 + \frac{1}{3!}\Delta(\Delta-1)(\Delta-2)(q-1)^3 \\ &\quad + \cdots + \frac{1}{N!}\left[\prod_{k=0}^{N-1}(\Delta-k)\right](q-1)^N \\ &\triangleq \hbar_\Delta(0) + \hbar_\Delta(1)(q-1) + \hbar_\Delta(2)(q-1)^2 + \cdots + \hbar_\Delta(N)(q-1)^N \end{aligned}$$

where

$$\hbar_\Delta(n) \triangleq \frac{1}{n!}\prod_{k=0}^{n-1}(\Delta-k) = \frac{\Delta(\Delta-1)(\Delta-2)\cdots(\Delta-n+1)}{n!}$$

This can be viewed as an FIR filter structure in which the usual delay elements are replaced by $q - 1 = z^{-1} - 1$.

Recursive Term Computation

Our Lagrange interpolation filter is again

$$\hat{H}_\Delta(q^{-1}) \;=\; \sum_{n=0}^{N} \hbar_\Delta(n)(q-1)^n$$

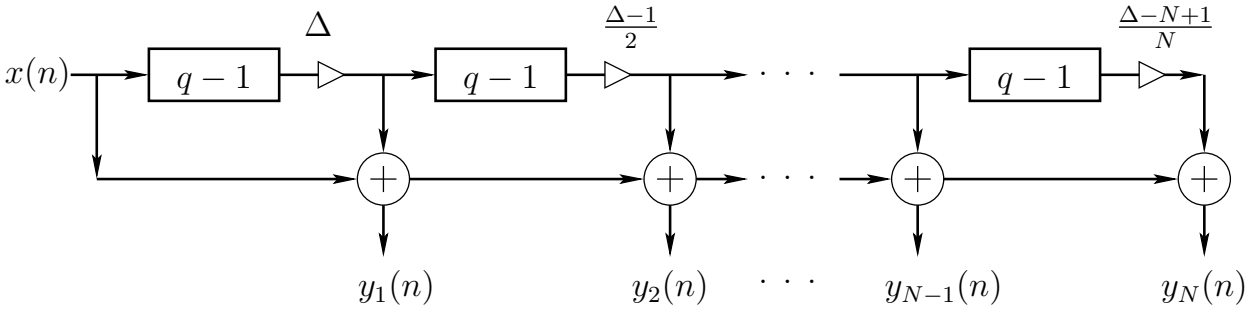
where the coefficients of $(q-1)^n = (z^{-1}-1)^n$ are again

$$\hbar_\Delta(n) \;\stackrel{\Delta}{=}\; \frac{1}{n!}\prod_{k=0}^{n-1}(\Delta - k) \;=\; \frac{\Delta(\Delta-1)(\Delta-2)\cdots(\Delta-n+1)}{n!}$$

Note that we can recursively compute the terms in the sum from left to right:

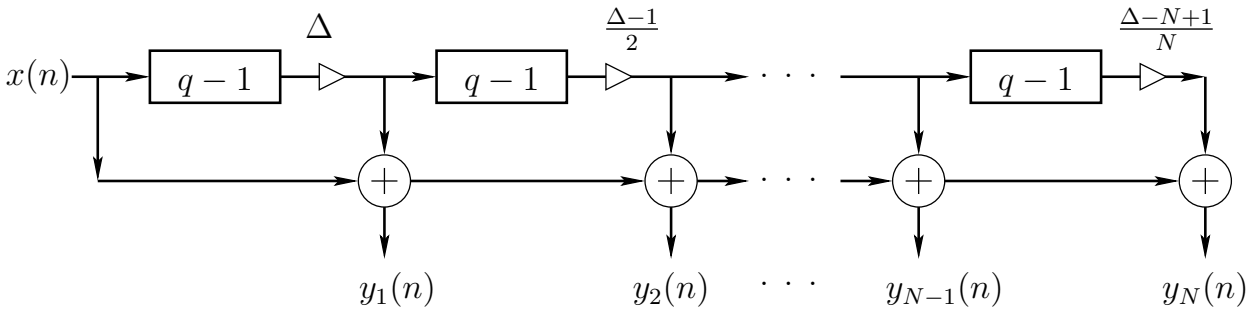
$$\begin{aligned}\hbar_\Delta(n) &= \hbar_\Delta(n-1)\cdot\frac{\Delta-n+1}{n}\\ (q-1)^n &= (q-1)^{n-1}\cdot(q-1)\end{aligned}$$

Thus, we can crank out the terms in series and sum the intermediate signals:

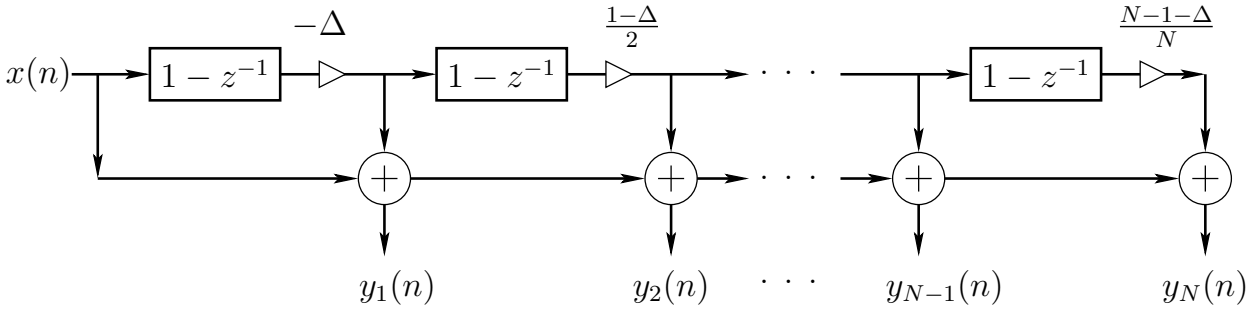


Efficient Time-Invariant Lagrange Interpolation

We derived the following efficient implementation:



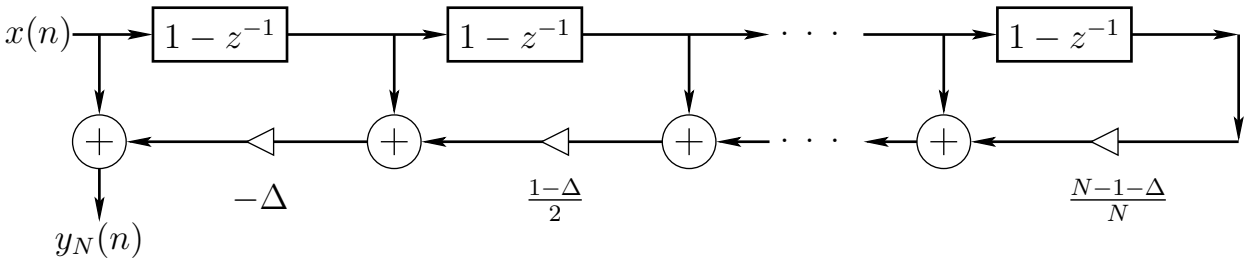
Replacing q by z^{-1} and manipulating signs gives the following:



- Note that Langrange interpolators of all orders $1, 2, \ldots, N$ are available at the summer outputs. The signal $y_k(n)$ is the kth-order Lagrange-interpolation approximation of $x(n - \Delta)$.
- Note that this recursive implementation is only valid for constant Δ since the first-order differences "remember" past values of Δ.

Time-Varying Lagrange Interpolation

To allow for time-varying Δ, the coefficients $\hbar_\Delta(k)$ can be moved out of the $(q-1)^k$ chain as follows:



- In this case the lower-order interpolations are not readily available.
- As in the Farrow structure, we obtain a variable linear combination of *fixed* FIR filters $(1-z^{-1})^k$, $k = 0, 1, 2, \ldots N$.

A More Elegant Derivation

Define the *backwards difference operator* δ by

$$\delta f(n) \stackrel{\Delta}{=} f(n) - f(n-1)$$

and the *factorial polynomials* (aka *rising factorials* or *Pochhammer symbol*) by

$$x^{[N]} \stackrel{\Delta}{=} x(x+1)(x+2)\cdots(x+N-2)(x+N-1)$$

These give a discrete-time counterpart to $\frac{d}{dx}x^N = Nx^{N-1}$, viz.,

$$\delta\, x^{[N]} = N\, x^{[N-1]}$$

In these terms, a *discrete-time Taylor series* about $n = k$ can be defined:

$$\hat{f}(t) \stackrel{\Delta}{=} \sum_{n=0}^{\infty} [\delta^n f(k)] \frac{(t-k)^{[n]}}{n!}$$

- Known as "Newton's Backward Difference Formula"
- Truncating this expansion at $n = N$ again yields Nth-order *Lagrange interpolation* on uniformly spaced samples

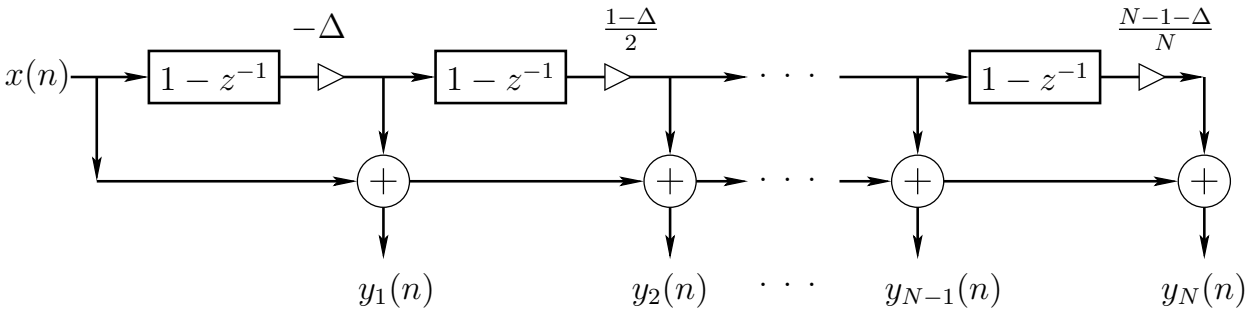
Nth-order Lagrange interpolation via truncated *discrete-time Taylor series expansion* about time $n = k$:

$$\hat{f}(t) \stackrel{\Delta}{=} \sum_{n=0}^{N} [\delta^n f(k)] \frac{(t-k)^{[n]}}{n!}$$

Each term in the expansion can be computed *recursively* from the previous term:

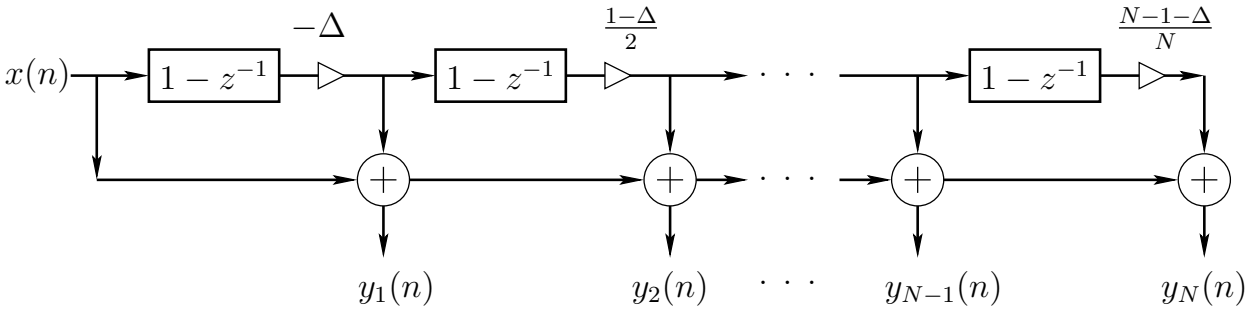
$$[\delta^n f(k)] \frac{(t-k)^{[n]}}{n!} = [\delta^{n-1} f(k)] \frac{(t-k)^{[n-1]}}{(n-1)!} \times \boxed{\frac{t-k+N-1}{N} \cdot [\delta f(k)]}$$

This gives the same efficient computational form found previously:



where $\Delta \stackrel{\Delta}{=} k - t$ is the *desired delay* for fractional-delay filtering, and $y_k(n)$ is the output signal for kth-order Lagrange interpolation (modular!). See also *Newton's divided difference interpolation formula*.

Features of Truncated-Taylor Lagrange-Interpolation



- Computational Complexity $= 3N - 1$ multiplies and $3N - 2$ additions for Nth-order interpolation
- If multiplication factors for a given delay $\Delta = k - t$ are stored in memory, then the complexity reduces to N multiplies and $2N - 2$ adds per output sample (multiplies comparable to a plain FIR interpolator, but additions are $\approx$ doubled)
- Interpolation order can be increased or decreased by modularly adding/deleting sections on the right

References

- Timo Laakso et al., "Splitting the Unit Delay—Tools for Fractional Delay Filter Design." IEEE Signal Processing Magazine, vol. 13, no. 1, pp. 30–60, Jan 1996.
- Philippe Depalle and Stephan Tassart, "Fractional Delay Lines using Lagrange Interpolators," ICMC Proceedings, pp. 341–343, 1996.
- Candan, C., "An Efficient Filtering Structure for Lagrange Interpolation," Signal Processing Letters, IEEE, vol. 14, no. 1, pp. 17–19, Jan. 2007
- Vesa Lehtinen and Markku Renfors, "Structures for Interpolation, Decimation, and Nonuniform Sampling Based on Newton's Interpolation Formula." SAMPTA'09, May 2009, Marseille, France.
- Andreas Franck, "Efficient Algorithms and Structures for Fractional Delay Filtering Based on Lagrange Interpolation," JAES vol. 56, no. 12, 2008.
- `http://boytim.orgfree.com/asrc/`
- `http://mathworld.wolfram.com/-NewtonsDividedDifferenceInterpolationFormula.html`

Lagrange Interpolation in Faust, Fixed Delay d

Fast time-invariant-delay algorithm `fdelaylti` — derived by truncated Taylor series expansion above — in Faust's `filter.lib`

```
import("music.lib");

fdelaylti(N,n,d,x)
  = delay(n,id,x) <: seq(i,N,section(i)) : !,_
with {
  o = (N-1.00001)/2;  // ~center FIR interpolator
  dmo = d - o; // assumed >=0 [d > (N-1)/2]
  id = int(dmo);
  fd = o + frac(dmo);
  section(i,x,y) = (x-x') * c(i) <: _,+(y);
  c(i) = (i - fd)/(i+1);
};

process = fdelayl(5,1024,5.4); // 5th-order ex.
```

Lagrange Interpolation in Faust, Variable Delay d

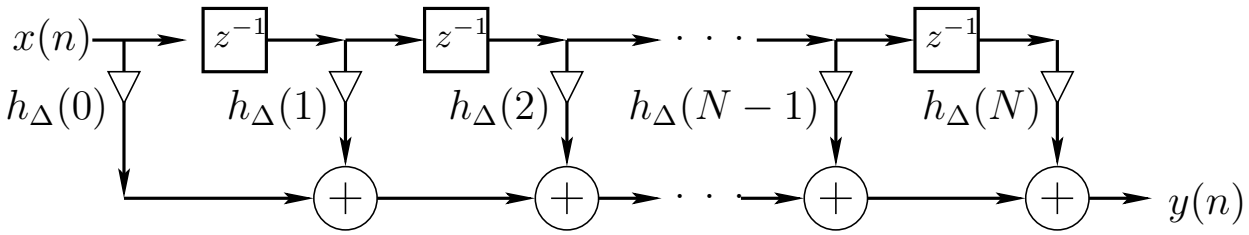
With variable delay, it is easiest to work with the plain FIR form, because the interpolation coefficients can jump around at will along the delay:

```
import("music.lib");

fdelayltv(N,n,d,x) = sum(i, N+1, delay(n,id+i,x) * h(N,fd,i))
with {
  o = (N-1.00001)/2;  // ~center FIR interpolator
  dmo = d - o; // assumed >=0 [d > (N-1)/2]
  id = int(dmo);
  fd = o + frac(dmo);
  h(N,d,n) = facs1(N,d,n) * facs2(N,d,n);
  facs1(N,d,n) = select2(n,1,prod(k,max(1,n),
                                  select2(k<n,1,fac(d,n,k))));
  facs2(N,d,n) = select2(n<N,1,prod(l,max(1,N-n),
                                  fac(d,n,l+n+1)));
  fac(d,n,k) = (d-k)/((n-k)+(n==k));
};

process = fdelayltv(5,1024,5.4); // 5th-order ex.
```

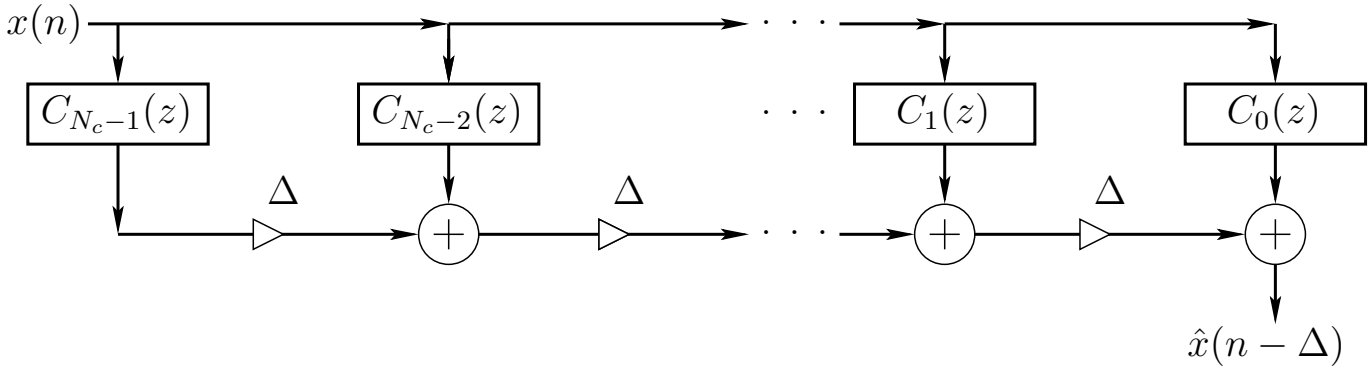
Recall explicit formula for Lagrange interpolation coefficients:

$$h_d(n) = \prod_{\substack{k=0 \\ k\neq n}}^{N} \frac{d-k}{n-k}, \qquad n = 0, 1, 2, \ldots, N$$

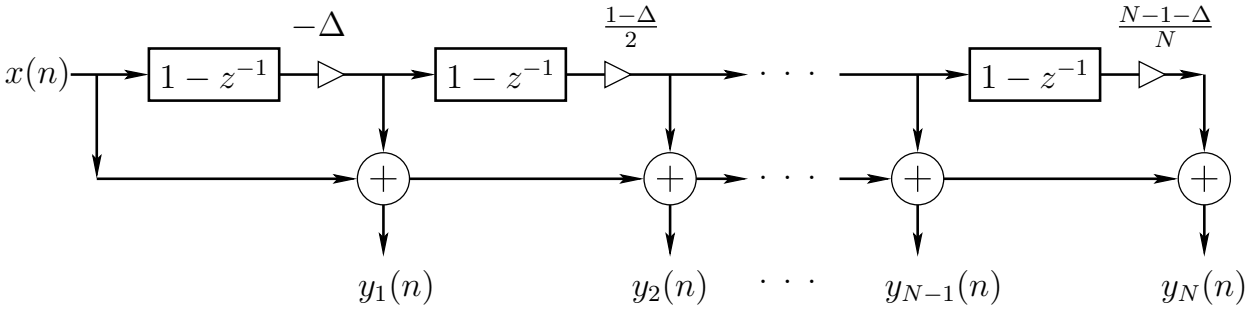
Summary of Lagrange Interpolators Considered



Generic FIR



Farrow Structure



Truncated Taylor Expansion

Thiran Allpass Interpolators

Given a desired delay $\Delta = N + \delta$ samples, an order N allpass filter

$$H(z) = \frac{z^{-N} A\left(z^{-1}\right)}{A(z)} = \frac{a_N + a_{N-1} z^{-1} + \cdots + a_1 z^{-(N-1)} + z^{-N}}{1 + a_1 z^{-1} + \cdots + a_{N-1} z^{-(N-1)} + a_N z^{-N}}$$

can be designed having *maximally flat group delay* equal to Δ at DC using the formula

$$a_k = (-1)^k \left(\begin{array}{c} N \\ k \end{array} \right) \prod_{n=0}^{N} \frac{\Delta - N + n}{\Delta - N + k + n}, \; k = 0, 1, 2, \ldots, N$$

where

$$\left(\begin{array}{c} N \\ k \end{array} \right) = \frac{N!}{k!(N-k)!}$$

denotes the kth *binomial coefficient*

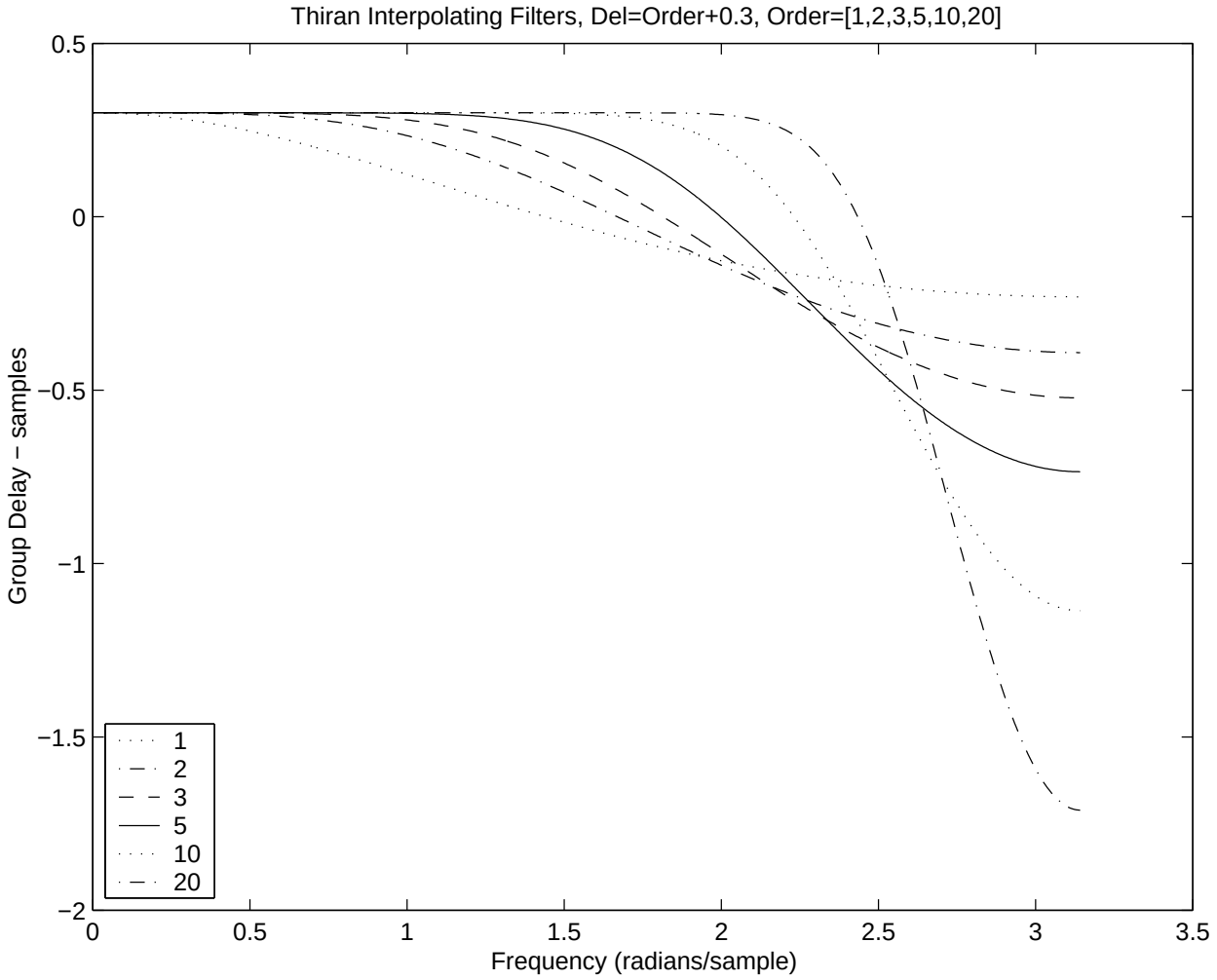
- $a_0 = 1$ without further scaling
- For sufficiently large Δ, stability is guaranteed
  rule of thumb: $\Delta \approx$ order
- Mean group delay is always N samples
  (for any stable Nth-order allpass filter):

  $$\frac{1}{2\pi} \int_0^{2\pi} D(\omega) d\omega \stackrel{\Delta}{=} -\frac{1}{2\pi} \int_0^{2\pi} \Theta'(\omega) d\omega = -\frac{1}{2\pi} \left[\Theta(2\pi) - \Theta(0)\right] = N$$

- Only known closed-form case for allpass interpolators of arbitrary order

- Effective for delay-line interpolation needed for *tuning* since pitch perception is most acute at low frequencies.

Frequency Responses of Thiran Allpass Interpolators for Fractional Delay



Large Delay Changes

When implementing large delay-length changes (by many samples), a useful implementation is to *cross-fade* from the initial delay line configuration to the new configuration.

- Computation doubled during cross-fade
- Cross-fade should be slow enough to sound smooth
- Not a true "morph" from one delay length to another, since we do not pass through the intermediate delay lengths
- A single delay line can be *shared* such that the cross-fade occurs from one *read-pointer* (plus interpolation filtering) to another
- Typically used with a *cross-correlator* that looks for good cross-fade delays (where cross-correlation is maximized)
  - Used in Synchronous OverLap Add (SOLA) methods for Time Scale Modification (TSM) and Frequency Scaling
  - Used (originally?) by the Eventide Harmonizer (Frequency Scaling)
  - Can alternatively do a *power* cross-fade[5] (instead of the usual *amplitude* cross-fade) where the cross-correlation is at a *minimum* in magnitude[6]

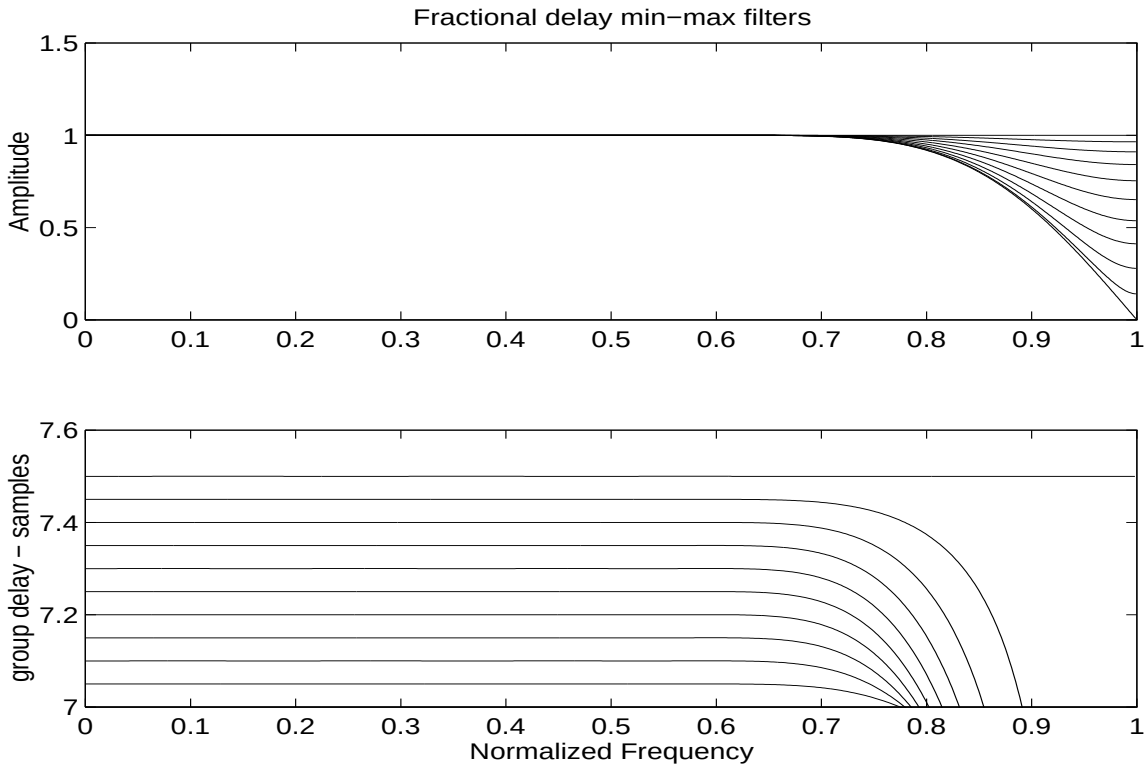
[5] `https://ccrma.stanford.edu/~jos/sasp/Panning_Problem.html`
[6] pointed out in an email from Robert Bristow-Johnson received 2019-08-16

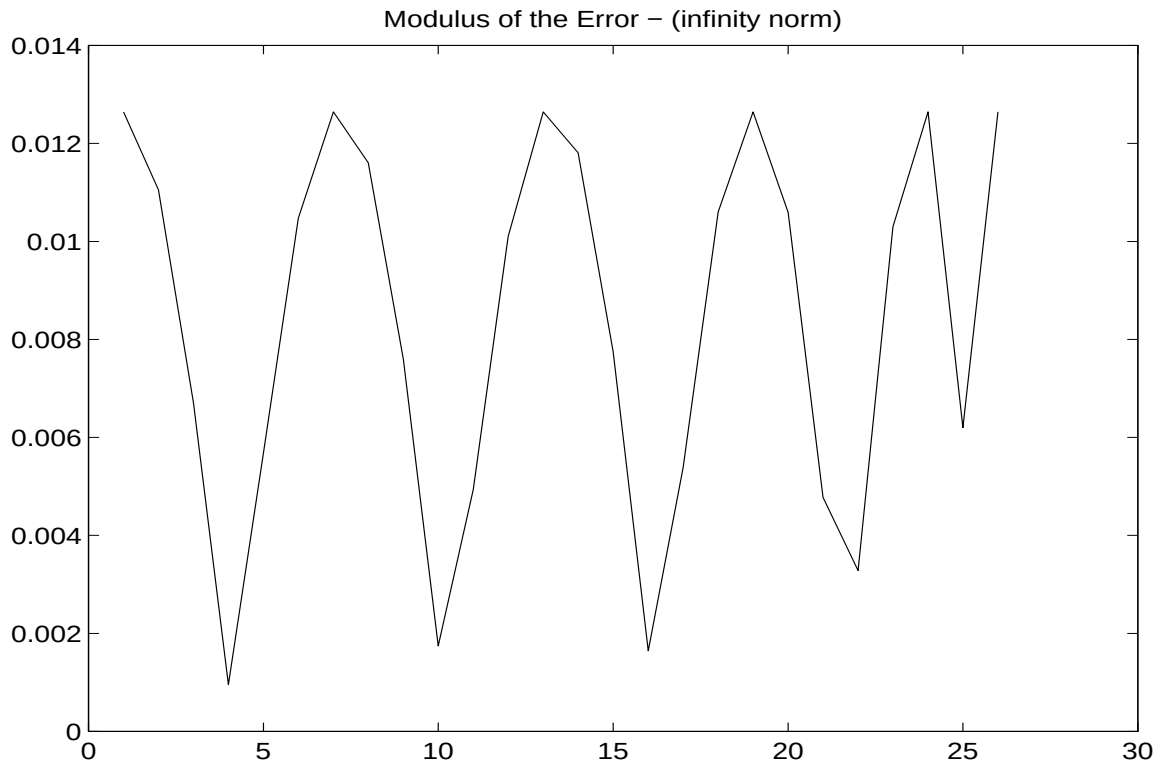
L-Infinity (Chebyshev) Fractional Delay Filters

- Use Linear Programming (LP) for real-valued L_∞-norm minimization
- Remez exchange algorithm:
  `firpm` (formerly `remez`): real FIR design
  `cfirpm` (formerly `cremez`): complex FIR design
- In the complex case, we have a problem known as a *Quadratically Constrained Quadratic Program*
- Approximated by sets of linear constraints
  (e.g., a *polygon* can be used to approximate a *circle*)
- Can solve with `cvx` code developed by Prof. Boyd's group
- See Mohonk-97 paper[7] for details.

[7] `http://ccrma.stanford.edu/~jos/resample/optfir.pdf`

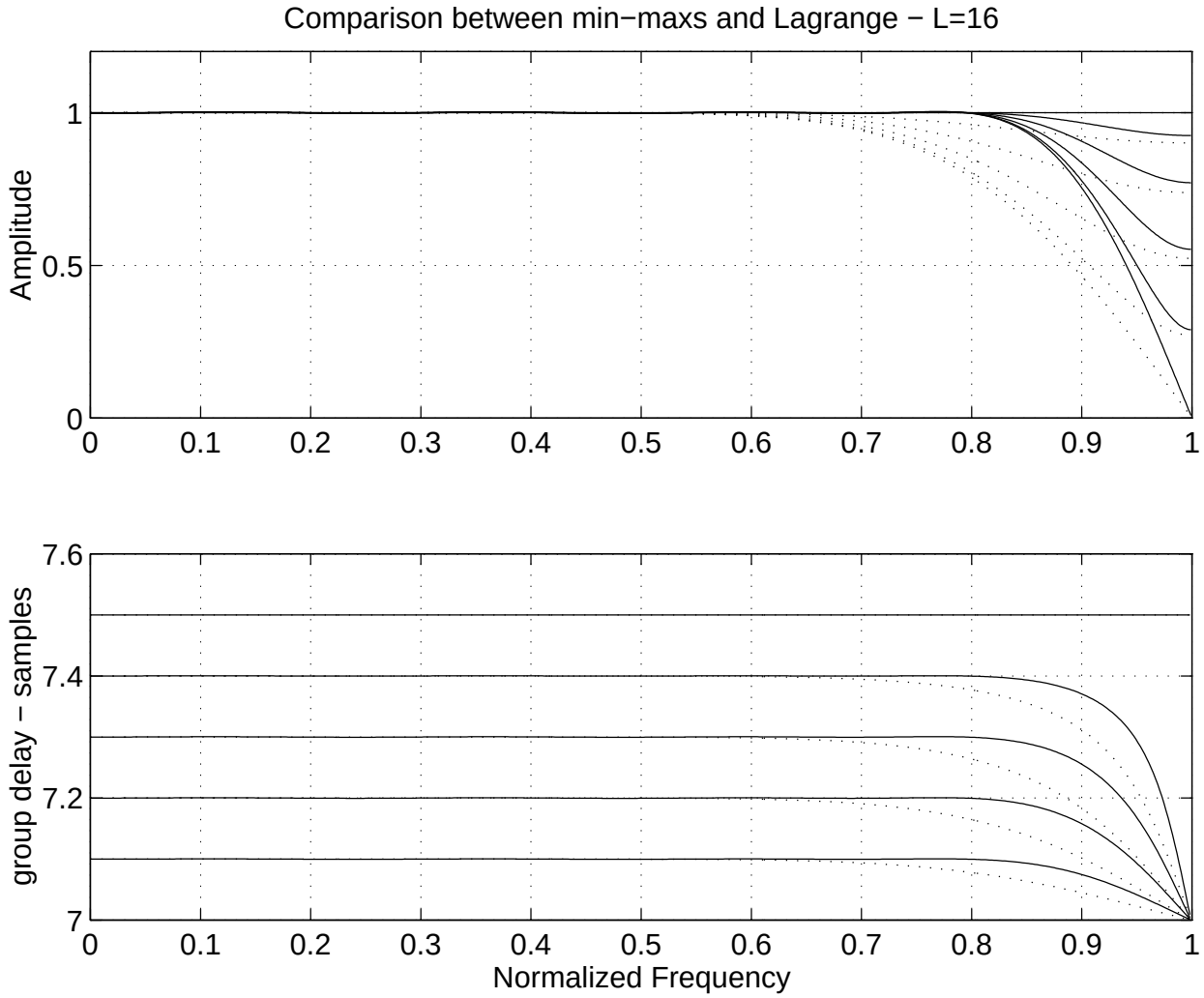
Chebyshev FD-FIR Design Example





- Equal-ripple in passband
- Must tabulate FIR coefficients for each Δ
- Such tables interpolate well when Δ densely sampled

Comparison of Lagrange and Optimal Chebyshev Fractional-Delay Filter Frequency Responses



Interpolation Summary

Well Known Closed-Form Solutions

	Order			
	1	N	Large N	∞
FIR	Linear	Lagrange	Windowed Sinc	
IIR	Allpass_1	Thiran		Sinc

Tabulated Alternative (Order N)

Design a digital filter (FIR or IIR) that approximates

$$H_\Delta\left(e^{j\omega T}\right) = e^{-j\omega\Delta T}, \quad \Delta = \text{Desired delay in samples}$$

optimally in some sense, with coefficients tabulated over a range of Δ samples (and interpolated on lookup).