Simple Interpolators suitable for Real Time Fractional Delay Filtering

Linear Interpolation

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

where \( \eta \) = desired fractional delay.

One-multiply form:

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

- 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:

\[ y(n + \alpha) = y(n) + \alpha \dot{y}(n) + \frac{\alpha^2 \ddot{y}(n)}{2!} + \frac{\alpha^3 \dddot{y}(n)}{3!} + \cdots \]

where \( \alpha = -\eta = \text{fractional advance desired} \)

\( \text{interpolation time between samples } n \text{ and } n + 1 \)

- 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:**

\[ \dot{y} \left( n - \frac{1}{2} \right) = \frac{1}{2} \cdot y(n) + \frac{1}{2} \cdot y(n - 1) \]

**Quarter-sample delay:**

\[ \dot{y} \left( n - \frac{1}{4} \right) = \frac{3}{4} \cdot y(n) + \frac{1}{4} \cdot y(n - 1) \]

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

  \[ h(t) = \begin{cases} 1 - |t/T| & |t| \leq T \\ 0 & \text{otherwise} \end{cases} \]

  followed by sampling at the desired phase

  **Discrete-Time Approximation**

  \[ x(n) \xrightarrow{\downarrow M} \hat{x(n)} \xrightarrow{\downarrow M} \hat{s(n)} \]

  - Replacing \( h(t) \) by \( h_s(t) \triangleq \text{sinc} \left( \frac{t}{M} \right) \) converts linear interpolation to ideal bandlimited interpolation (see “sinc interpolation” below)
First-Order Allpass Interpolation

\[ x(n - \Delta) = y(n) = \eta \cdot x(n) + x(n - 1) - \eta \cdot y(n - 1) \]
\[ = \eta \cdot [x(n) - y(n - 1)] + x(n - 1) \]
\[ H(z) = \frac{\eta + z^{-1}}{1 + \eta z^{-1}} \]

- Low frequency delay given by
  \[ \Delta \approx \frac{1 - \eta}{1 + \eta} \quad \text{(exact at DC)} \]
- Same complexity as linear interpolation
- Good for delay-line interpolation, not random access
- Best used with fixed fractional delay \( \Delta \)
- 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 \( \tau \approx T/(1 - \eta_{max}) \)

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).

### Interpolation Overview

<table>
<thead>
<tr>
<th>Order</th>
<th>Well Known Closed-Form Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>FIR Linear</td>
</tr>
<tr>
<td>Large ( N )</td>
<td>IIR Allpass</td>
</tr>
<tr>
<td>( \infty )</td>
<td>Sinc</td>
</tr>
</tbody>
</table>

#### Tabulated Alternative (Order \( N \))

Design a digital filter (FIR or IIR) that approximates
\[ H_\Delta(e^{j\omega T}) = e^{-j\omega \Delta T}, \quad \Delta = \text{Desired delay in samples} \]
optimally in some sense, with coefficients tabulated over a range including \( \Delta \) 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
\[ h_s(t) \triangleq \text{sinc}(f_s t) \triangleq \frac{\sin(t)}{t} \quad (\text{sinc function}) \]

(Proof: sampling theorem)
The Sinc Function ("Cardinal Sine")

\[ \text{sinc}(t) \begin{cases} \frac{\sin(\pi t)}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases} \]

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 \( \text{sinc}(n - \eta) \) “evaluates” the signal at an arbitrary continuous time \( \eta \in \mathbb{R} \):

\[ y(\eta) = \sum_{n=-\infty}^{\infty} y(n) \text{sinc}(\eta - n) \]

Proof: Sampling Theorem

Ideal D/A Example

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

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 $\Delta$ samples:

- Ideal impulse response = bandlimited delayed impulse
  
  $$h_\Delta(t) = \text{sinc}(t - \Delta) = \sin(\pi(t - \Delta))/\pi(t - \Delta)$$

- Ideal frequency response = “brick wall” lowpass response, cutting off at $f_s/2$ and having linear phase
  
  $$e^{-j\omega \Delta T}$$
  
  $$H_\Delta(\omega) \triangleq \mathcal{F}(h_\Delta) = \begin{cases} e^{-j\omega \Delta}, & |\omega| < \pi f_s \\ 0, & |\omega| \geq \pi f_s \end{cases}$$
  
  $$H_\Delta(e^{j\omega T}) = e^{-j\omega \Delta T}, \quad -\pi \leq \omega T < \pi$$
  
  $$\leftrightarrow \text{sinc}(n - \Delta), \quad n = 0, \pm 1, \pm 2, \ldots$$

  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

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

FIR Frequency Response, Zero-Phase Alignment

$$\hat{H}_\Delta(e^{j\omega T}) = \sum_{n=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} h_\Delta(n)e^{-j\omega n T}$$

Error to Minimize

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

$L_2$ Error Norm

$$J(\hat{h}) \triangleq \|E\|^2 = \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} |E(e^{j\omega T})|^2 d\omega$$

$$= \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} \|H_\Delta(e^{j\omega T}) - \hat{H}_\Delta(e^{j\omega T})|^2 d\omega$$

By Parseval’s Theorem

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

Optimal Least-Squares FIR Interpolator

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

Truncated-Sinc Interpolation

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

- Vertical axis in dB, horizontal axis in spectral samples
- Optimal in least-squares sense
- Poor stop-band rejection ($\approx 20$ 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
  
  $$\hat{h}_\Delta(n) = \begin{cases} w(n - \Delta)\text{sinc}[\alpha(n - \Delta)], & 0 \leq n \leq L - 1 \\ 0, & \text{otherwise} \end{cases}$$

- 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

Frequency Response : Kaiser Window

-400 -200 200 400

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

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 $\Delta \approx L_1$
- 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
  ⇒ 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

Lowpass Filter Design

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

Online references (FIR interpolator design)
- Music 421 Lecture 2 on Windows
- Music 421 Lecture 3 on FIR Digital Filter Design
- Optimal FIR Interpolator Design

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
Interpolator Types

<table>
<thead>
<tr>
<th>Order</th>
<th>1</th>
<th>N</th>
<th>Large N</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>Linear</td>
<td>Lagrange</td>
<td>Windowed Sinc</td>
<td></td>
</tr>
<tr>
<td>IIR</td>
<td>Allpass</td>
<td>Thiran</td>
<td>Sinc</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
l_k(x) \triangleq \frac{(x-x_0) \cdots (x-x_{k-1})(x-x_{k+1}) \cdots (x-x_N)}{(x_k-x_0) \cdots (x_k-x_{k-1})(x_k-x_{k+1}) \cdots (x_k-x_N)}
\]

- Numerator gives a zero at all samples but the \( k \)th
- Denominator simply normalizes \( l_k(x) \) to 1 at \( x = x_k \)

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:

\[
\frac{d^m E(e^{j\omega})}{d\omega^m} \bigg|_{\omega=0} = 0, \quad m = 0, 1, 2, \ldots, N,
\]

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

\[
E(e^{j\omega}) \triangleq e^{-j\omega\Delta} - \sum_{n=0}^{N} h_\Delta(n)e^{-jn}\omega
\]

and \( \Delta \) 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:

\[
|H(e^{j\omega})| \leq 1, \quad \forall \omega T \in [-\pi, \pi]
\]
Order 4 Amplitude Response Over a Range of Fractional Delays

\[ \Delta = 1.5:0.1:2.5 \]

Order 5 Amplitude Response Over a Range of Fractional Delays

\[ \Delta = 2.0:0.1:3.0 \text{ plus } 2.495 \text{ and } 2.505 \]

Order 4 Phase Delay Over a Range of Fractional Delays

\[ \Delta = 1.5:0.1:2.5 \text{ plus } 2.499 \]

- 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 Phase Delay Over a Range of Fractional Delays

\[ \Delta = 2.0:0.1:3.0 \text{ plus } 2.495 \text{ and } 2.505 \]

- 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

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

Lagrange Interpolation Coefficients
Orders 1, 2, and 3

<table>
<thead>
<tr>
<th>( h_\Delta ) Order</th>
<th>( h_\Delta(0) )</th>
<th>( h_\Delta(1) )</th>
<th>( h_\Delta(2) )</th>
<th>( h_\Delta(3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N = 1 )</td>
<td>( 1 - \Delta )</td>
<td>( \Delta )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N = 2 )</td>
<td>( \frac{(\Delta - 1)(\Delta - 2)}{2} )</td>
<td>( -\Delta(\Delta - 2) )</td>
<td>( \frac{(\Delta - 1)}{2} )</td>
<td>( \frac{(\Delta - 1)(\Delta - 2)}{2} )</td>
</tr>
<tr>
<td>( N = 3 )</td>
<td>( \frac{(\Delta - 1)(\Delta - 2)(\Delta - 3)}{6} )</td>
<td>( \frac{(\Delta - 2)(\Delta - 3)}{2} )</td>
<td>( \frac{\Delta - 1}{2} )</td>
<td>( \frac{(\Delta - 2)(\Delta - 3)(\Delta - 4)}{6} )</td>
</tr>
</tbody>
</table>

- 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 \( \Delta = N/2 \)

Faust Code For Lagrange Fractional Delay

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


Matlab Code For Lagrange Fractional Delay

```matlab
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-Generated C++ Code

Inner-loop function generated by Faust:

```cpp
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] & 15)])
    - ((0.156250f * ftbl0[(((iRec0[0] - 3) & 15)])
    + (3.906250e-02f * ftbl0[(((iRec0[0] & 15)])])))
    + (0.878906f * ftbl0[((iRec0[0] & 15)]))));
// post processing
iRec0[1] = iRec0[0];
iVec0[1] = iVec0[0];
}
}
```

Test Program (Faust 0.9.9.3):

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

Variable FIR Interpolating Filter

Basic idea: Each FIR filter coefficient $h_n$ becomes an order $N_c$ polynomial in the delay parameter $\Delta$:

$$h_\Delta(n) \overset{\Delta}{=} \sum_{m=0}^{N_c} c_n(m)\Delta^m, \quad n = 0, 1, 2, \ldots, N_h$$

$$\Leftrightarrow H_\Delta(z) \overset{\Delta}{=} \sum_{n=0}^{N_h} h_\Delta(n)z^{-n}$$

$$= \sum_{n=0}^{N_h} \left[ \sum_{m=0}^{N_c} c_n(m)\Delta^m \right] z^{-n}$$

$$= \sum_{m=0}^{N_c} \left[ \sum_{n=0}^{N_h} c_n(m)z^{-n} \right] \Delta^m$$

$$\overset{\Delta}{=} \sum_{m=0}^{N_c} C_m(z)\Delta^m$$

- More generally: $H_\Delta(x) = \sum_m \alpha(\Delta)C_m(x)$ 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 $\Delta$ using Horner’s rule:

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

⇒ filter structure becomes

As delay $\Delta$ varies, “basis filters” $C_k(z)$ remain fixed
⇒ very convenient for changing $\Delta$ over time

Farrow Structure Design Procedure

Solve the $N_\Delta$ 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 $N$th-order Lagrange interpolation filter using $O(N^2)$ 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 $\Delta^k$, computed recursively, where $\Delta$ is the desired delay:

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

- The next section describes a formulation achieving complexity $O(N)$.

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

$$H\Delta(q^{-1}) = H\Delta(1) + H\Delta'(1)(q-1) + \frac{1}{2}H\Delta''(1)(q-1)^2 + \frac{1}{3!}H\Delta'''(1)(q-1)^3 + \cdots$$

$$+ \cdots + \frac{1}{N!} \prod_{k=0}^{N} (\Delta - k) (q-1)^N$$

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 $N$th-order Lagrange FIR interpolation filter is obtain by truncating this expansion at order $N$:

$$\hat{H}\Delta(q^{-1}) \triangleq 1 + \Delta(q-1) + \frac{1}{2}\Delta(\Delta-1)(q-1)^2 + \frac{1}{3!}\Delta(\Delta-1)(\Delta-2)(q-1)^3 + \cdots + \frac{1}{N!} \prod_{k=0}^{N} (\Delta - k) (q-1)^N$$

$$\triangleq h\Delta(0) + h\Delta(1)(q-1) + h\Delta(2)(q-1)^2 + \cdots + h\Delta(N)(q-1)^N$$

where

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

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

Lagrange Interpolation by Taylor Expansion

The transfer function of a $\Delta$-sample delay is

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

- $N$th-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

$$E(z) = H\Delta(z) - \hat{H}\Delta(z) \text{ about dc } (z = 1)$$

- This means matching the first $N + 1$ terms of the Taylor expansion of $H\Delta(z) = z^{-\Delta}$ about $z = 1$

Recursive Term Computation

Our Lagrange interpolation filter is again

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

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

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

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

$$h\Delta(n) = h\Delta(n-1) \cdot \frac{\Delta - n + 1}{n} (q-1)^n = \frac{\Delta - n + 1}{n} \frac{1}{(q-1)^{n-1}} \cdot (q-1)$$

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

$$x(n) \rightarrow \Delta \rightarrow \Delta^{-1} \rightarrow \cdots \rightarrow \Delta^{-N+1}$$
Efficient Time-Invariant Lagrange Interpolation

We derived the following efficient implementation:

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

Note that Lagrange interpolators of all orders $1, 2, \ldots, N$ are available at the summer outputs. The signal $y_k(n)$ is the $k$th-order Lagrange-interpolation approximation of $x(n - \Delta)$.

Note that this recursive implementation is only valid for constant $\Delta$ since the first-order differences “remember” past values of $\Delta$.

A More Elegant Derivation

Define the backwards difference operator $\delta$ by

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

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

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

These give a discrete-time counterpart to

$$\frac{d}{dx} x^{[N]} = N x^{[N-1]}$$

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

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

- Known as “Newton’s Backward Difference Formula”
- Truncating this expansion at $n = N$ again yields $N$th-order Lagrange interpolation on uniformly spaced samples

Time-Varying Lagrange Interpolation

To allow for time-varying $\Delta$, the coefficients $h_{\Delta}(k)$ 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 $(1 - z^{-1})^k$, $k = 0, 1, 2, \ldots N$.

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

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

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

$$\frac{d}{dx} x^{[N]} = N x^{[N-1]}$$

This gives the same efficient computational form found previously:

where $\Delta \overset{\Delta}{=} k - t$ is the desired delay for fractional-delay filtering, and $y_k(n)$ is the output signal for $k$th-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 $N$th-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

- http://home.comcast.net/~matt.boytim/aerc/
- http://mathworld.wolfram.com/-NewtonsDividedDifferenceInterpolationFormula.html

Lagrange Interpolation in Faust, Fixed Delay $d$

Fast time-invariant-delay algorithm \texttt{fdelaylti} — derived by truncated Taylor series expansion above — in Faust’s \texttt{filter.lib}

\begin{verbatim}
import("music.lib");

\texttt{fdelaylti(N,n,d,x)}
\begin{verbatim}
  = 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.
\end{verbatim}
\end{verbatim}

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:

\begin{verbatim}
import("music.lib");

\texttt{fdelayltv(N,n,d,x)} = sum(i, N+1, delay(n,id+i,x) * h(N,fd,i))
\begin{verbatim}
  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.
\end{verbatim}
\end{verbatim}

Recall explicit formula for Lagrange interpolation coefficients:

$$h_d(n) = \prod_{k=0}^{N} \frac{d-k}{n-k}, \quad n = 0, 1, 2, \ldots, N$$
Summary of Lagrange Interpolators Considered

\[ x(n) \xrightarrow{h_\Delta(0)} x(n-\Delta) \xrightarrow{h_\Delta(1)} x(n-2\Delta) \quad \cdots \quad x(n-N\Delta) \xrightarrow{h_\Delta(N)} y(n) \]

Generic FIR

\[ C_{N-1}(z) \xrightarrow{\Delta} C_{N-2}(z) \xrightarrow{\Delta} \cdots \xrightarrow{\Delta} C_0(z) \]

Farrow Structure

\[ x(n) \xrightarrow{1-z^{-1}} \xrightarrow{1-z^{-1}} \cdots \xrightarrow{1-z^{-1}} \xrightarrow{1-z^{-1}} \]

Truncated Taylor Expansion

\[ (\Delta - N + k + n) \quad k = 0, 1, 2, \ldots, N \]

\[ a_k = (-1)^k \binom{N}{k} \prod_{n=0}^{\Delta-N} \frac{\Delta - N + n}{\Delta - N + k + n}, \quad k = 0, 1, 2, \ldots, N \]

\[ \binom{N}{k} = \frac{N!}{k!(N-k)!} \]

denotes the \( k \)th binomial coefficient

- \( a_0 = 1 \) without further scaling
- For sufficiently large \( \Delta \), stability is guaranteed
- Rule of thumb: \( \Delta \approx N \) order
- Mean group delay is always \( N \) samples
- Only known closed-form case for allpass interpolators of arbitrary order

Thiran Allpass Interpolators

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

\[ H(z) = \frac{z^{-N} A(z^{-1})}{A(z)} = a_N + a_{N-1}z^{-1} + \cdots + a_1z^{-(N-1)} + z^{-N} \]

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

can be designed having maximally flat group delay equal to \( \Delta \) at DC using the formula

\[ a_k = (-1)^k \binom{N}{k} \prod_{n=0}^{\Delta-N} \frac{\Delta - N + n}{\Delta - N + k + n}, \quad k = 0, 1, 2, \ldots, N \]

\[ \binom{N}{k} = \frac{N!}{k!(N-k)!} \]

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

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 long 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 associated filtering) to another.
L-Infinity (Chebyshev) Fractional Delay Filters

- Use Linear Programming (LP) for real-valued $L_\infty$-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 for details.

---

Chebyshev FD-FIR Design Example

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

---

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

Well Known Closed-Form Solutions

<table>
<thead>
<tr>
<th>Order</th>
<th>1</th>
<th>N</th>
<th>Large N</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>Linear</td>
<td>Lagrange</td>
<td>Windowed Sinc</td>
<td></td>
</tr>
<tr>
<td>IIR</td>
<td>Allpass</td>
<td>Thiran</td>
<td>Sinc</td>
<td></td>
</tr>
</tbody>
</table>

Tabulated Alternative (Order $N$)

Design a digital filter (FIR or IIR) that approximates

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

optimally in some sense, with coefficients tabulated over a range of

$\Delta$ samples (and interpolated on lookup).