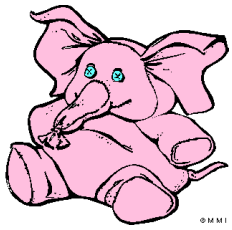
Polynomial Interpolators for High-Quality Resampling of Oversampled Audio



REVISED VERSION

by Olli Niemitalo in October 2001. Distribute, host and use this paper freely.
http://www.student.oulu.fi/~oniemita/DSP/INDEX.HTM
oniemitalo@sublevel3.org

Abstract

This paper discusses piece-wise polynomial interpolators used in audio resampling and presents new low-order designs that are optimized for high-quality resampling of oversampled audio. Source code and useful tables for using the interpolators are included.

Welcome to read the paper that took three entire weeks (24/7) of my life, approximately $\frac{1}{1000}$ of the whole deal. It was a very educational experience. I learned to play with genetic/evolutionary algorithms (big thanks to Bram de Jong for introducing Differential Evolution, which was also used in generation of the passband approximations, and to the Duane Wise and Robert Bristow-Johnson team-up, whose AES paper "Performance of Low-Order Polynomial Interpolators in the Presence of Oversampled Input", this one owes a lot to). And I learned a new language, PostScript, which was used to generate the graphs directly from a C++ program. And I got more and more familiar with LaTeX. Also some pretty nice interpolators were generated, and I'm sure to be using them in the future. I could easily say I need a short break from interpolation, but I won't because that's such an over-used closing joke.

You may notice that there aren't any references. The additional bits of information for creating this paper were gathered from Internet exclusively, and most of the sources were not named publications. So if you wish to find them, just figure out a few keywords and head to http://www.google.com.

You are very welcome to send error reports/comments/opinions/announcements of implementations/work offers/free audio software/anything except viruses/spam to my e-mail (under the title).

Contents

1. Introduction . . . 3
2. A bunch of interpolators . . . 5
   - 2.1 Drop-sample, linear, B-spline . . . 6
   - 2.2 Lagrange . . . 8
   - 2.3 Hermite (1st-order-osculating) . . . 10
   - 2.4 2nd-order-osculating . . . 13
   - 2.5 Watte tri-linear and "parabolic 2x" . . . 14
3. A quality measure . . . 16
4. New optimal designs . . . 19
   - 4.1 2-point, 3rd-order optimal . . . 19
   - 4.2 4-point, 2nd-order optimal . . . 21
   - 4.3 4-point, 3rd-order optimal . . . 22
   - 4.4 4-point, 4th-order optimal . . . 23
   - 4.5 6-point, 4th-order optimal . . . 25
   - 4.6 6-point, 5th-order optimal . . . 27
5. Comparison . . . 29
   - 5.1 Linear . . . 29
   - 5.2 B-spline . . . 30
   - 5.3 Lagrange . . . 31
   - 5.4 Hermite . . . 31
   - 5.5 2nd-order-osculating . . . 32
   - 5.6 Watte tri-linear . . . 33
   - 5.7 Parabolic 2x . . . 33
   - 5.8 Optimal . . . 34
6. Implementation . . . 36
   - 6.1 Linear . . . 38
   - 6.2 B-spline . . . 39
   - 6.3 Lagrange . . . 40
   - 6.4 Hermite . . . 42
   - 6.5 2nd-order-osculating . . . 45
   - 6.6 Watte tri-linear . . . 47
   - 6.7 Parabolic 2x . . . 48
   - 6.8 2-point, 3rd-order optimal . . . 49
   - 6.9 4-point, 2nd-order optimal . . . 50
   - 6.10 4-point, 3rd-order optimal . . . 51
   - 6.11 4-point, 4th-order optimal . . . 53
   - 6.12 6-point, 4th-order optimal . . . 54
   - 6.13 6-point, 5th-order optimal . . . 57

7. Summary 59
8. Pre-emphasis 61
9. Conclusion 64

1. Introduction

Sampled audio data is a discrete-time representation of a continuous signal, perhaps of the voltage that came from the microphone while recording. As a rule, the data holds the amplitude values of the continuous signal at the boundaries of evenly spaced time intervals. To change the sampling frequency by an unconstrained ratio – a common task in audio processing – or to create sub-sample length delays, both a form of resampling, one needs to be able to read the continuous signal between the samples.

The solution is to create an approximation of the continuous signal, from the information contained in the samples, and to sample that. This is called interpolation, finding the function value between known samples. A common interpolation method is linear interpolation, where the continuous function is approximated as piece-wise-linear by drawing lines between the successive samples. An even more crude form of interpolation is drop-sample interpolation, drawing a horizontal line from each sample until the following sample.

Drop-sample and linear interpolation (as such) are not adequate for high-quality resampling, but even linear interpolation is a big improvement compared to drop-sample. Both of them fall into the category of piece-wise polynomial interpolators. Theoretically, one could create a very high-order polynomial interpolator and get the desired quality. A rule of thumb was formed from the results of this paper: The dependence, of the interpolation error in dB scale and the computational complexity of a good polynomial interpolator, is a linear function with an offset. Unfortunately, the function is relatively gently sloping, so the polynomial order would need to be increased to something unreasonable to get transparent quality.

A hybrid solution is to first oversample the input by a simple ratio using discrete methods and then interpolate this oversampled data using a polynomial interpolator. When a symmetrical FIR is used as the discrete oversampling filter, people often call the method sinc interpolation, especially if the oversampling ratio is large, which makes the FIR lowpass coefficient table resemble a windowed sinc and the impulse response of the whole hybrid interpolator a piece-wise polynomial approximation of a windowed sinc. The exact same results can be achieved differently, by interpolating the FIR coefficient table with the polynomial interpolator and by filtering using the interpolated coefficients, but this approach is computationally more expensive and

not suggested.

This paper concentrates on improving the polynomial interpolation stage of the hybrid method, for oversampling ratios of 2, 4, 8, 16 and 32 on the oversampling stage.

A discrete oversampling filter can increase the sampling frequency to an integer N multiple, i.e. oversample by N. Typically, the filter is a FIR filter, because using a FIR one can do "random access" on the data with no extra computational cost – a useful property if N is high, because in such cases typically only a fraction of the samples in the oversampled signal are used. Another recent solution is a polyphase structure of (two) IIR all-pass filters[1]. Any lowpass structure could be used, so traditional multirate filters are also an alternative.

In the simple FIR case, the tap number and hence memory consumption grow in a linear relation to N. However, the instruction count per each obtained sample at the new samplerate remains the same as only every Nth tap needs to be computed for an output sample, the other taps landing on zero amplitude between the original samples.

After oversampling by N, the signal is still discrete and the amplitude of the continuous signal is only known at the new samplepoints. One could cheat a little and always use the value of the most recent samplepoint before the asked place. This is known as drop-sample, the lowest order member of the family of piece-wise polynomial interpolators. It distorts the continuous signal in a similar manner as a sample-and-hold circuit, making it look like stairsteps instead of the original. To really know in what way this kind of distortion is bad, one must look at the spectrum.

The spectrum of a discrete-time (audio) signal is periodic by the sampling frequency (f_s) and symmetrical around 0Hz (due to real, i.e. non-complex samples). Ideally, in range $-\frac{f_s}{2}\ldots+\frac{f_s}{2}$ a discrete signal has an identical spectrum with the continuous signal it is a representation of. The rest of the spectrum is stuffed with equally strong images of this band, each centered around an integer multiple of f_s, up to infinite frequencies. Direct resampling of such a signal would certainly lead to severe problems (you'd get nearly all of the new samples zero amplitude and possibly some occasional crackle here and there).

A polynomial interpolator, for example drop-sample, can and should be thought of as a filter with a continuous-time impulse response. A non-discrete impulse response yields a non-periodic frequency response that has an overall descending envelope. So the spectral images are attenuated by this continuous filter, making resampling a more sensible process. Ideally, there would be no images, as the continuous (audio) signal that we are trying to imitate is presumed to be bandlimited in range $-\frac{f_s}{2}\ldots+\frac{f_s}{2}$. The goal is to have the images attenuated to low enough a level so that when they in resampling map or alias over the audio band, they will not be audible.

[1] http://www.cmsa.wmin.ac.uk/~artur/Poly.html

With the hybrid interpolator, we shall assign the original sampling frequency the symbol f_{s0} and the sampling frequency after the discrete oversampling stage the symbol $f_{s1} = N f_{s0}$.

The N-times oversampling filter is a discrete lowpass filter that has its cutoff set at the original $\frac{f_{s0}}{2}$ (ideally). Because the impulse response is discrete, the frequency response will still be periodic, but with a period of $f_{s1} = N f_{s0}$. This period is a multiple of the original period f_{s0}, so we don't have aliasing problems at this phase; that's why we chose an integer N to begin with. The oversampling filter has a stopband on, and therefore (ideally) removes, all of the original images but those centered around multiples of f_{s1}.

The discrete oversampling filter can easily create the steep cutoff required and a low stopband at its operating range, and the polynomial interpolator can attenuate the remaining spectral images that could not be touched with discrete-time methods. Polynomial interpolators don't have a flat passband, which can be compensated for in the frequency response of the oversampling filter, or in some other stage.

Piece-wise polynomial interpolation in this context means that individual polynomials are created between successive samplepoints. These interpolators can be classified both by the number of samplepoints required from the neighborhood for calculating the value at a position, and by the order of the polynomial[2]. For example, if an interpolator takes four samplepoints and the polynomial is of third order, we shall classify it as *4-point, 3rd-order* (short *4p 3o*). Depending on the interpolator, the polynomial order is typically one less than the number of points, matching the number of coefficients in the polynomial to the number of samples, but there are many exceptions to this rule.

This paper only considers interpolators that follow the scheme described in the previous paragraph and have impulse responses symmetrical around zero, which rules out interpolators that operate on an odd number of points (potential causes of headache because they, when shifted in time to be symmetrical around zero, have polynomial transitions not at the samplepoints but halfway between them).

2. A bunch of interpolators

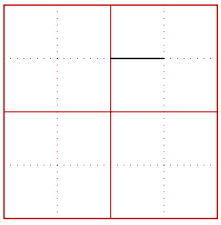
The following are the known piece-wise polynomial interpolators that are potentially useful for audio interpolation.

[2]The order of a polynomial is the order of the highest-order term in the polynomial. For example, $3x^2 + x - 2$ is second-order.

2.1 Drop-sample, linear, B-spline

B-splines are a family of interpolators that can be constructed by convolving a drop-sample interpolator by a drop-sample interpolator repeated times. The drop-sample interpolation impulse response is:

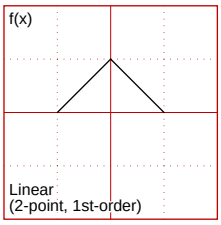
$$f(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{otherwise.} \end{cases}$$



Formula, figure: Drop-sample interpolation (also the 0th-order B-spline) impulse response. The zero-amplitude areas are unmarked

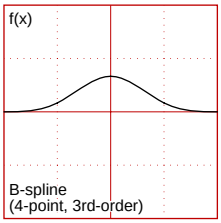
Drop-sample is the 0th-order B-spline that operates only on one point. It, and other even-order B-splines operate on an odd number of samples so we will not investigate them further. The first three odd-order B-spline impulse responses are: (The symmetry property has been exploited to shorten the expressions)

$$f(x) = \begin{cases} 1-x & 0 \le x < 1 \\ 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



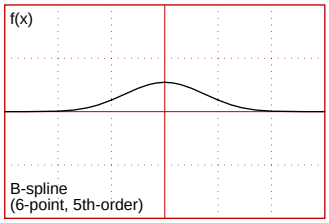
Formula, figure: 2-point, 1st-order linear interpolation (also the 1st-order B-spline) impulse response

$$f(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}x^3 & 0 \le x < 1 \\ \frac{4}{3} - 2x + x^2 - \frac{1}{6}x^3 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 3rd-order B-spline impulse response

$$f(x) = \begin{cases} \frac{11}{20} - \frac{1}{2}x^2 + \frac{1}{4}x^4 - \frac{1}{12}x^5 & 0 \le x < 1 \\ \frac{17}{40} + \frac{5}{8}x - \frac{7}{4}x^2 + \frac{5}{4}x^3 - \frac{3}{8}x^4 + \frac{1}{24}x^5 & 1 \le x < 2 \\ \frac{81}{40} - \frac{27}{8}x + \frac{9}{4}x^2 - \frac{3}{4}x^3 + \frac{1}{8}x^4 - \frac{1}{120}x^5 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order B-spline impulse response

It is notable that higher-order B-splines don't have zero crossings at integer x, therefore the interpolated curve will not necessarily go through the points. That as such is not a bad quality.

The higher the order of a B-spline, the more continuous derivatives it has. The number of continuous successive derivatives, also counting the impulse response function itself as the 0th derivative, has shown to define the slope of the overall spectral envelope of any piece-wise polynomial interpolator. The slope is that number plus 1, times -6dB/oct. For B-splines, the number is same as the order, as can be seen here.

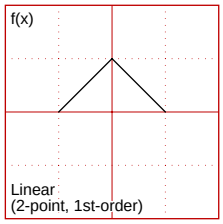


Figure: Linear interpolator, no continuous derivatives

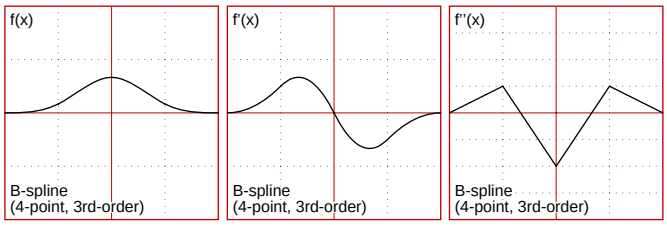


Figure: 3rd-order B-spline and its continuous derivatives

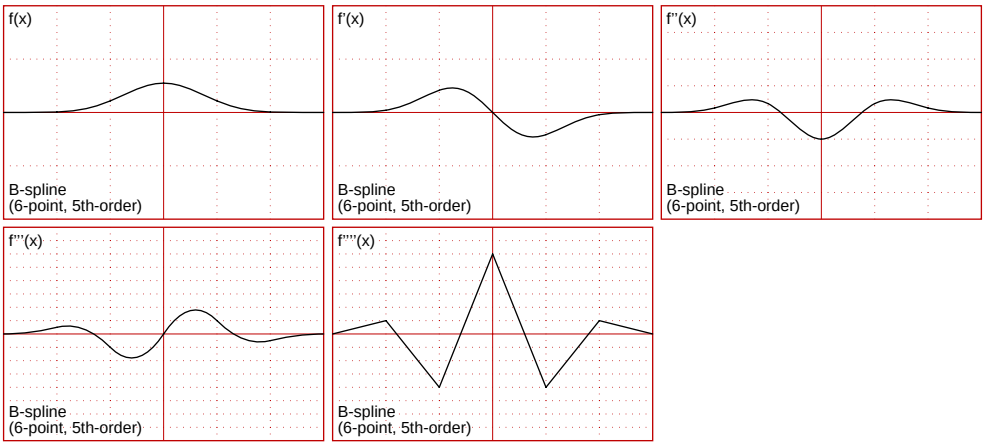


Figure: 5th-order B-spline and its continuous derivatives

Neither is having continuous derivatives a quality as such. When it's about audio, the frequency response is all that counts. In the following plot we use angular frequency

in radians on the horizontal axis. Angular frequency is same as $2\pi\frac{f}{f_s}$, i.e. two pi times frequency divided by sampling frequency. 0Hz maps to 0 in angular frequency, $\frac{f_s}{2}$ to π and f_s to 2π.

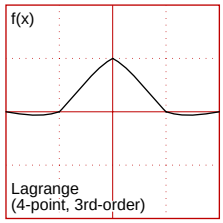
Figure: Frequency responses of the first three odd-order B-splines, including the linear interpolator

The frequency responses show wide holes at multiples of 2π. This means that as the images of the lowest audio frequencies land on these areas, they get heavily attenuated. On the other hand, the attenuation is not very strong at the images of near π frequencies. Also, with no oversampling, the highest audio frequencies in the passband are strongly attenuated, which certainly needs to be compensated for if higher-order B-splines are used with no oversampling. Typically, one would calculate the quality of an interpolator as the signal-to-noise ratio directly from the frequency response, for example by subtracting the magnitude at the strongest sidelobe top from the the magnitude at zero frequency, but we will later show why this is not an adequate quality measure when the interpolated signal is sampled audio.

2.2 Lagrange

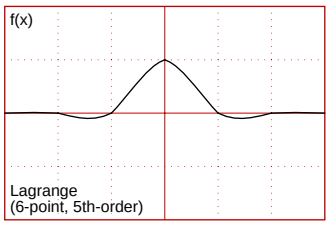
Lagrange polynomials are forced to go through a number of points. For example, the 4-point Lagrange interpolator polynomial is formed so that it goes through all of the four neighboring points, and the middle section is used. The 1st-order (2-point) Lagrange interpolator is the linear interpolator, which was already presented as part of the B-spline family. The order of the Lagrange polynomials is always one less than the number of points. The third- and fifth-order Lagrange impulse responses are:

$$f(x) = \begin{cases} 1-\frac{1}{2}x-x^2+\frac{1}{2}x^3 & 0 \le x < 1 \\ 1-\frac{11}{6}x+x^2-\frac{1}{6}x^3 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 3rd-order Lagrange impulse response

$$f(x) = \begin{cases} 1-\frac{1}{3}x-\frac{5}{4}x^2+\frac{5}{12}x^3+\frac{1}{4}x^4-\frac{1}{12}x^5 & 0 \le x < 1 \\ 1-\frac{13}{12}x-\frac{5}{8}x^2+\frac{25}{24}x^3-\frac{3}{8}x^4+\frac{1}{24}x^5 & 1 \le x < 2 \\ 1-\frac{137}{60}x+\frac{15}{8}x^2-\frac{17}{24}x^3+\frac{1}{8}x^4-\frac{1}{120}x^5 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order Lagrange impulse response

Lagrange interpolators do not have a continuous first derivative, but a surprise awaits as we derive further.

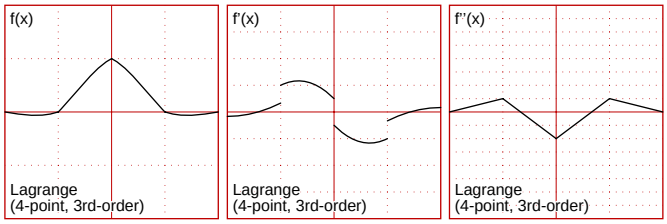


Figure: 3rd-order Lagrange and its early derivatives

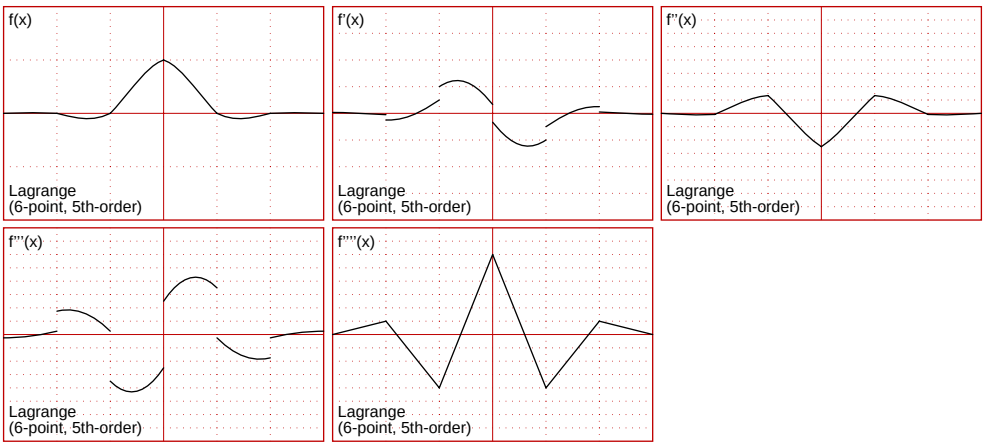


Figure: 5th-order Lagrange and its early derivatives

The surprise is every second derivative being continuous (except for the singular discontinuities). The frequency responses of these interpolators are:

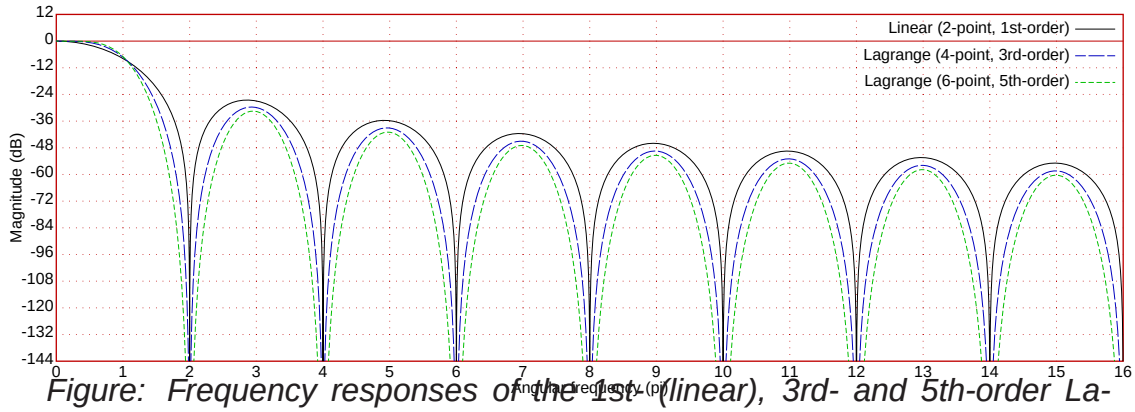


Figure: Frequency responses of the 1st- (linear), 3rd- and 5th-order Lagrange

The sidelobes don't get much lower as the order increases, but passband behaviour improves, also the holes at multiples of 2π become broader.

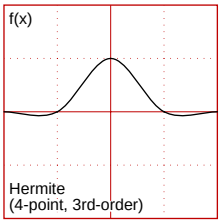
2.3 Hermite (1st-order-osculating)

Hermite interpolation is the first-order member of the family of osculating interpolators that in addition to matching the function value at the control points also match a number of derivatives. Hermite interpolation matches the first derivative. Since the actual derivative of the function (audio) is not known, Hermite interpolation matches to the derivatives of even-order Lagrangian polynomes. For example, the 4-point cubic Hermite will match its derivative at $y[x]$ to the derivative of a (Lagrangian) parabolic going through $y[x-1]$, $y[x]$ and $y[x+1]$. By increasing the order of the Lagrangians, one can create cubic Hermites that take advantage of more points. Note that as the order of Lagrangians increases, they converge to sinc interpolators, but the Hermite, limited by its cubic formulation, can only converge up to a point.

Another approach to create Hermite interpolators is to ramp between two successive Lagrangians with a linear weighting ramp. This also ensures the first derivative to match those of the Lagrangians at the points, but it also gives higher order Hermite interpolators, increasing how close they (theoretically) can approach the perfect sinc interpolator.

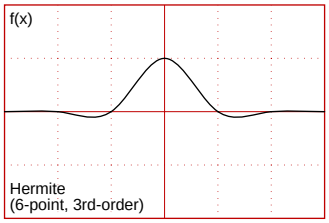
In fact, the 4-point Hermite can be constructed either way, and the same formula pops out. Here are the Hermite impulse reponses for the 4-point and both of the 6-point versions.

$$f(x) = \begin{cases} 1-\frac{5}{2}x^2+\frac{3}{2}x^3 & 0 \le x < 1 \\ 2-4x+\frac{5}{2}x^2-\frac{1}{2}x^3 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



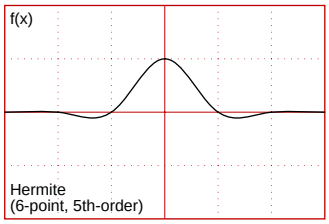
Formula, figure: 4-point, 3rd-order Hermite impulse response. This is also known as the Catmull-Rom spline, or the $\alpha = -\frac{1}{2}$ case of cardinal splines, where α is the derivative of the impulse response at $x = 1$.

$$f(x) = \begin{cases} 1-\frac{7}{3}x^2+\frac{4}{3}x^3 & 0 \le x < 1 \\ \frac{5}{2}-\frac{59}{12}x+3x^2-\frac{7}{12}x^3 & 1 \le x < 2 \\ -\frac{3}{2}+\frac{7}{4}x-\frac{2}{3}x^2+\frac{1}{12}x^3 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 3rd-order Hermite impulse response (first derivative matches with the first derivatives of the Lagrangians)

$$f(x) = \begin{cases} 1-\frac{25}{12}x^2+\frac{5}{12}x^3+\frac{13}{12}x^4-\frac{5}{12}x^5 & 0 \le x < 1 \\ 1+\frac{5}{12}x-\frac{35}{8}x^2+\frac{35}{8}x^3-\frac{13}{8}x^4+\frac{5}{24}x^5 & 1 \le x < 2 \\ 3-\frac{29}{4}x+\frac{155}{24}x^2-\frac{65}{24}x^3+\frac{13}{24}x^4-\frac{1}{24}x^5 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order Hermite impulse response (linear ramp between two Lagrangians)

Hermite interpolators have a continuous first derivative by definition, but let's take a deeper look anyhow.

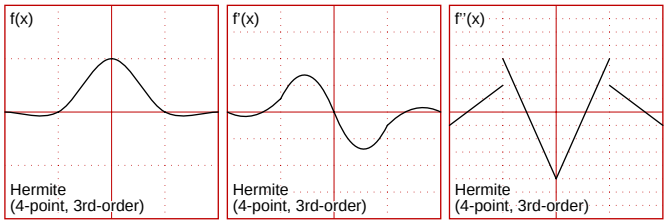


Figure: 4-point, 3rd-order Hermite and its early derivatives

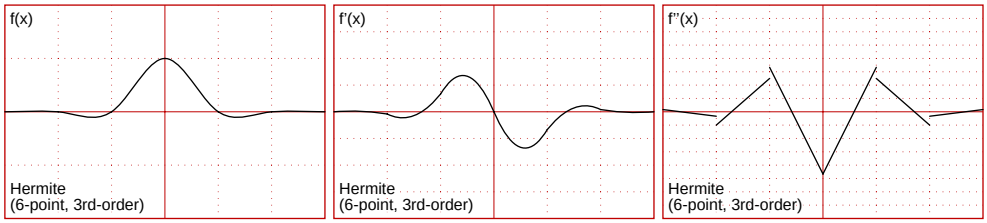


Figure: 6-point, 3rd-order Hermite and its early derivatives

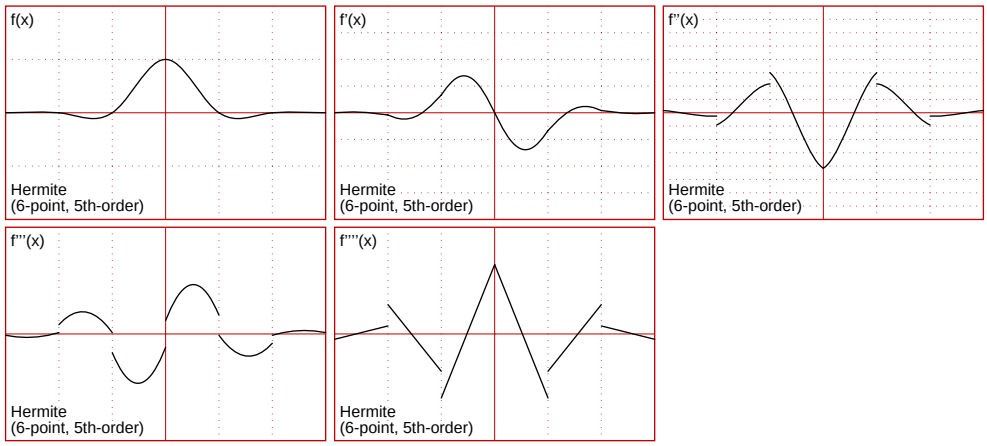


Figure: 6-point, 5th-order Hermite and its early derivatives

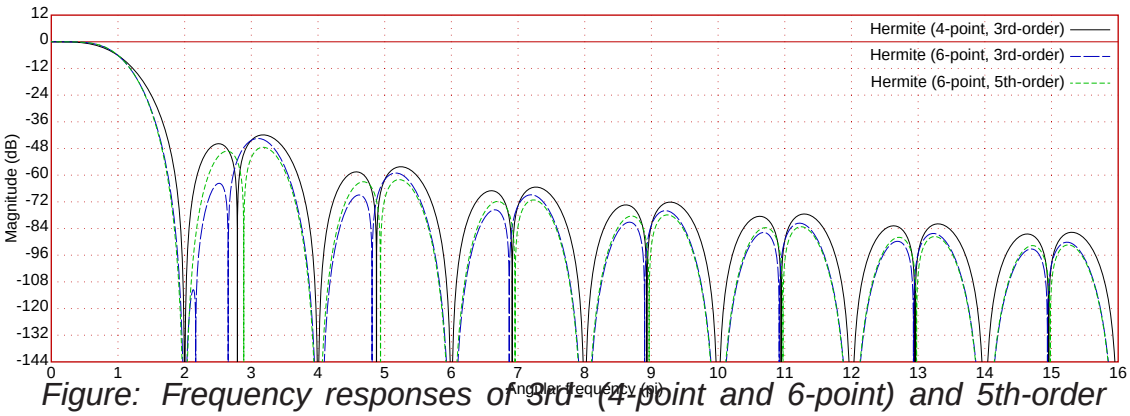


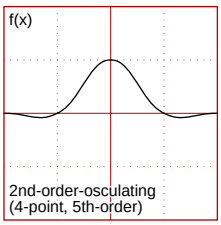
Figure: Frequency responses of 3rd- (4-point and 6-point) and 5th-order Hermite

An interesting feature in the Hermite frequency responses is that the largest side-lobes (compare for example to the frequency responses of Lagrange interpolators) have been "punctured". The passband behaviour is quite nice, too, so cubic Hermite is potentially useful as a stand-alone resampling interpolator.

2.4 2nd-order-osculating

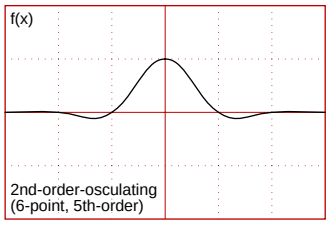
The definition of 2nd-order osculating interpolators is the same as of the Hermites, but also the second derivative is matched with the Lagrangians. Necessarily, the order of the interpolation polynomial must be at least 5, since there are 6 parameters, the function and two derivatives, all three at both of the two points, to match. Up to the 6-point version, these parameters leave no degrees of freedom in the 5th-order polynomial. The impulse responses are as follows.

$$f(x) = \begin{cases} 1 - x^2 - \frac{9}{2}x^3 + \frac{15}{2}x^4 - 3x^5 & 0 \le x < 1 \\ -4 + 18x - 29x^2 + \frac{43}{2}x^3 - \frac{15}{2}x^4 + x^5 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 5th-order 2nd-order-osculating impulse response

$$f(x) = \begin{cases} 1 - \frac{5}{4}x^2 - \frac{35}{12}x^3 + \frac{21}{4}x^4 - \frac{25}{12}x^5 & 0 \le x < 1 \\ -4 + \frac{75}{4}x - \frac{245}{8}x^2 + \frac{545}{24}x^3 - \frac{63}{8}x^4 + \frac{25}{24}x^5 & 1 \le x < 2 \\ 18 - \frac{153}{4}x + \frac{255}{8}x^2 - \frac{313}{24}x^3 + \frac{21}{8}x^4 - \frac{5}{24}x^5 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order 2nd-order-osculating impulse response

Here are the derivatives, two first of which should be continuous.

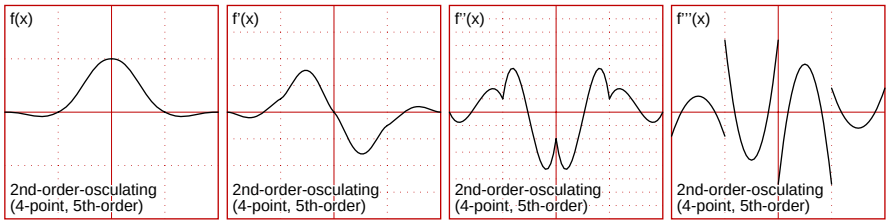


Figure: 4-point, 5th-order 2nd-order-osculating and its early derivatives

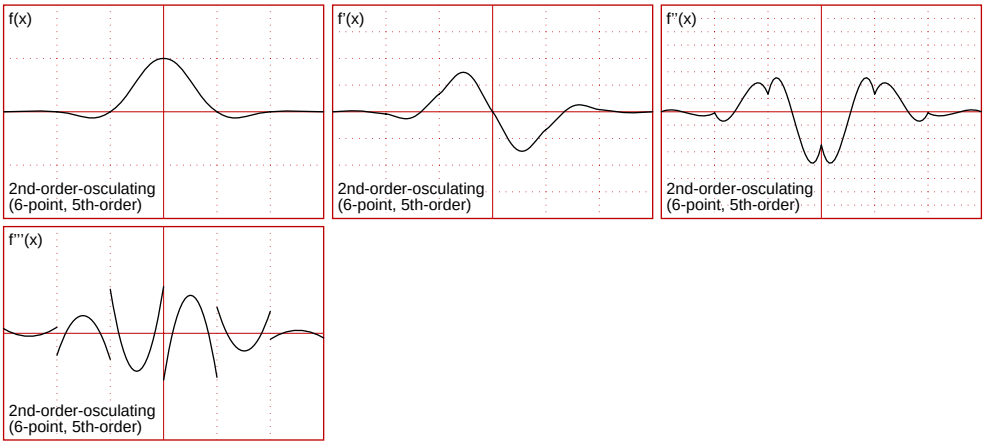


Figure: 6-point 5th-order 2nd-order-osculating and its early derivatives

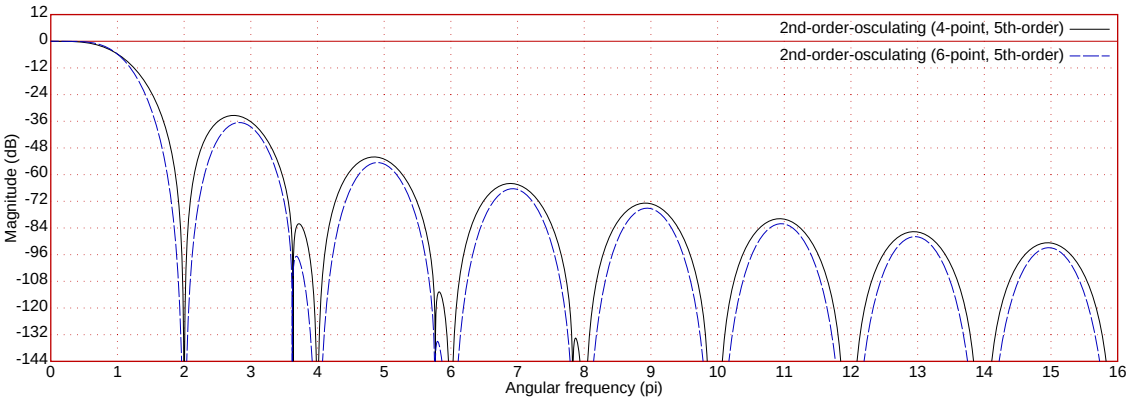


Figure: Frequency responses of 4- and 6-point 2nd-order-osculating

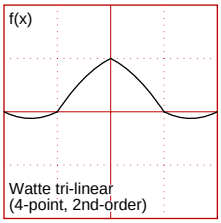
The steeper overall spectral envelope slope than Hermite can be seen in the frequency response. The sidelobes are not nearly as high as with Lagrange, but the hole at 2π is a little narrower.

2.5 Watte tri-linear and "parabolic 2x"

These two interpolators do not fall in any of the previous categories.

Watte tri-linear was presented by Jon Watte on the music-dsp mailing list. It is constructed by first creating two linear functions, first going through $y[x+1]$ and $y[x-1]$ and the second through $y[x]$ and $y[x+2]$, by shifting the first to penetrate $y[x]$ and the second to $y[x+1]$, and by weighting between them with a linear ramp. The impulse response, which visually quite resembles a 4-point Lagrangian, is as follows.

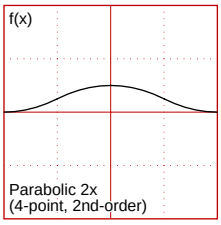
$$f(x) = \begin{cases} 1-\frac{1}{2}x-\frac{1}{2}x^2 & 0 \le x < 1 \\ 1-\frac{3}{2}x+\frac{1}{2}x^2 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 2nd-order Watte tri-linear impulse response

"Parabolic 2x" is my own design, and was created to be the lowest order 4-point interpolator with continuous function and first differential. As the differential must be zero at $x = 0$ and at the endpoints $x = \pm 2$, and be continuous at the borders of the sections, the only parameter left to define was the height of the curve, which was set so that the integral $x = -2..2$ became unity to ensure magnitude 1 for DC in the frequency response. The interpolated curve does not necessarily go through the points. The formula:

$$f(x) = \begin{cases} \frac{1}{2}-\frac{1}{4}x^2 & 0 \le x < 1 \\ 1-x+\frac{1}{4}x^2 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 2nd-order parabolic 2x impulse response

Just for the fun of it, the derivatives for these odd-balls, followed by the frequency response plots:

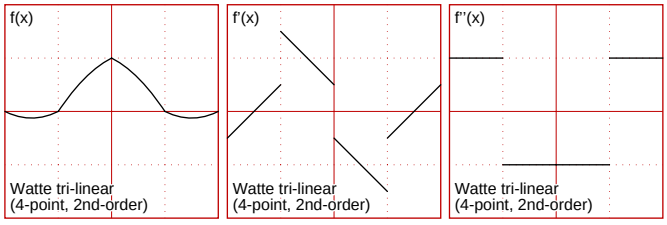


Figure: Watte tri-linear and its early derivatives

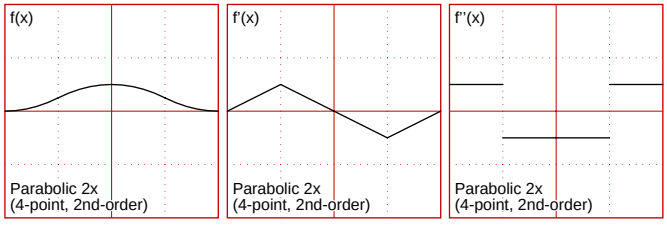


Figure: Parabolic 2x and its early derivatives

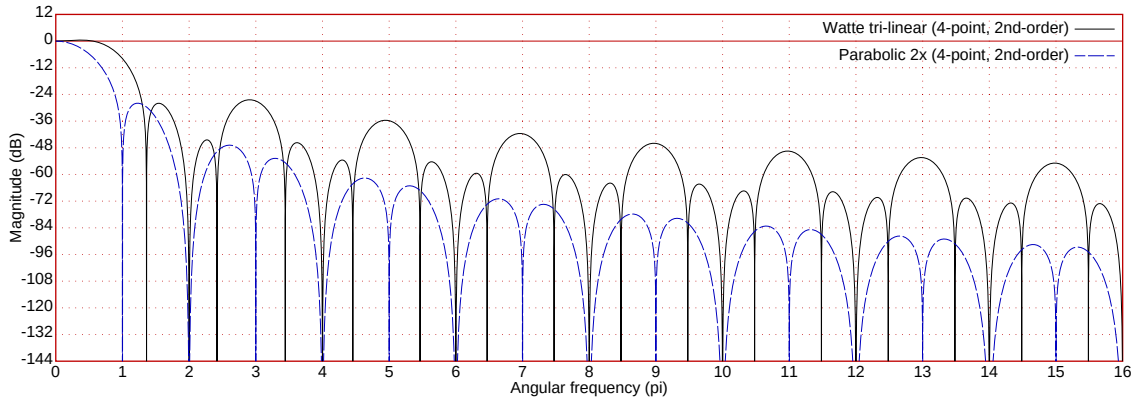


Figure: Frequency responses of Watte tri-linear and parabolic 2x

Looking at the frequency responses (put in the same graph to save paper), Watte tri-linear has an extraordinary steep cutoff slope, until the first short sidelobe. Parabolic 2x has a hole at π, which makes it suitable for use with oversampled data only. It has nicely low sidelobes and wide holes at multiples of 2π though.

3. A quality measure

When measuring the quality of interpolators working on oversampled input data, three things should be noted.

The first thing is the amount of input oversampling. We shall demonstrate this on a linear interpolator. The following frequency response was plotted so that the angular frequency corresponds to the sampling frequency of the oversampled data (N times the original).

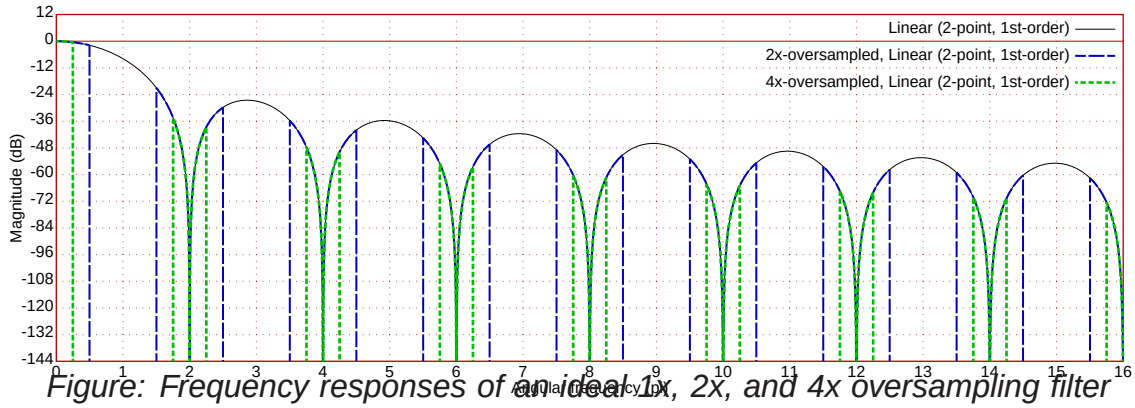


Figure: Frequency responses of an ideal 1x, 2x, and 4x oversampling filter combined with linear interpolation

As can be seen, the stopband required for the interpolator gets narrower as the oversampling ratio N increases. This is because if the data is ideally oversampled,

there are no frequencies above $\frac{\pi}{N}$ in the audio baseband, and the spectral images are also narrowed accordingly.

The second thing to note is that the passband of an interpolator is not flat and this must be compensated for through filtering, possibly in the oversampling stage. In most cases, there is some attenuation towards higher passband frequencies, so this compensation heightens the spectral images. We shall call this compensation *pre-emphasis* and it depends only on the interpolator frequency response.

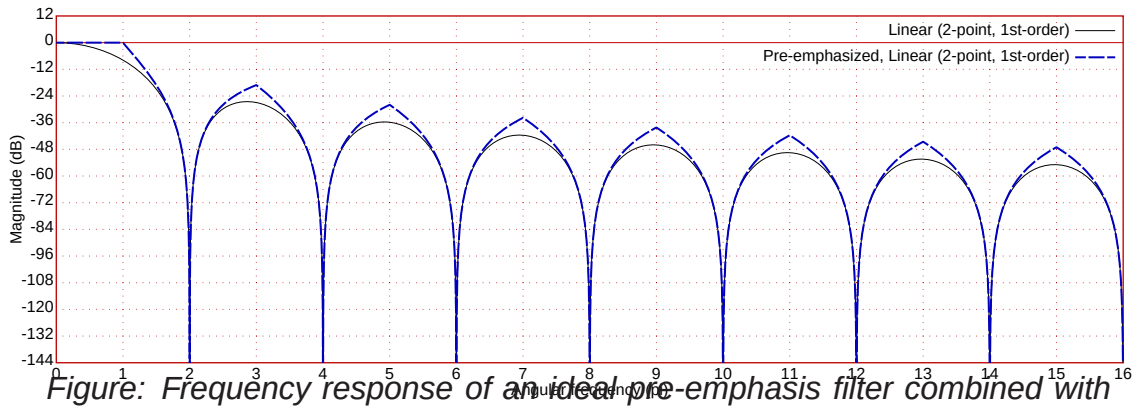


Figure: Frequency response of an ideal pre-emphasis filter combined with linear interpolation

The third thing to note is that audio generally has a pink spectral envelope. This is a much better presumption than white. Pieces of music are generally equalized to pink. Pink means that the spectrum decreases 3dB per an octave increase in frequency. To take this into account in interpolator quality evaluation, we filter the spectral images with a pinking filter, whose magnitude is proportional to $\frac{1}{\sqrt{w}}$, where w is the angular frequency of the passband frequency that creates the image. We shall call this process *pinking*. The pinking filter is normalized so that the magnitude at stopband edges is unity, so the pinking filter depends only on the amount of oversampling. The frequency responses of the pinking filters are:

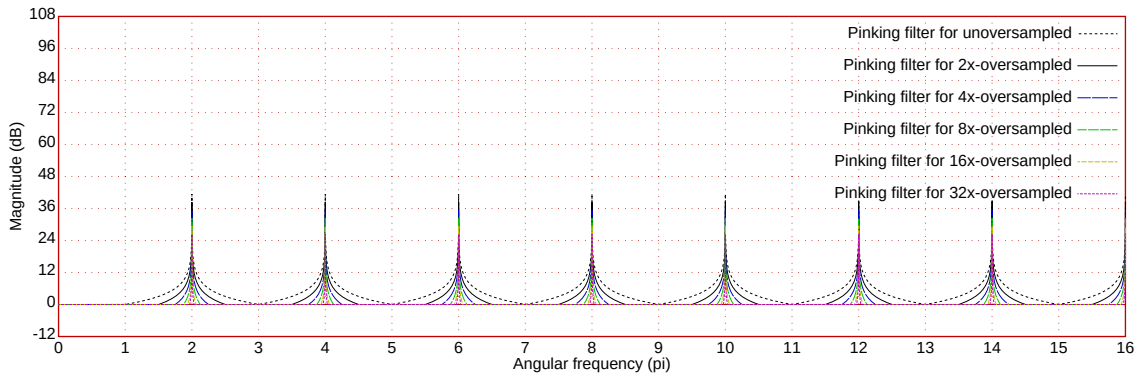


Figure: Frequency responses of the pinking filters

A demonstration on pinking:

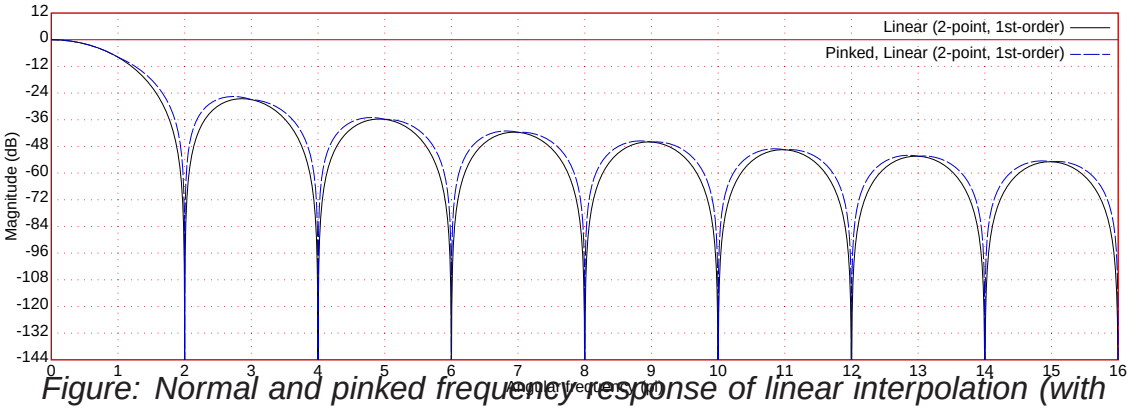


Figure: Normal and pinked frequency response of linear interpolation (with no oversampling)

In the demonstration with linear interpolation, pinking has a notable effect - the first sidelobe top has moved left and heightened sligtly.

Pinking emphasizes the importance of stopband attenuation near frequencies a multiple of 2π. This has proven to be important as some interpolators may have OK-looking frequency responses, but sound really bad when there are typical amounts of low frequencies in the input, compared to testing with white noise. Because pinking would be infinitely strong near 0Hz, we choose to keep increasing the pinking gain only down to the frequency corresponding to 5Hz in a 44100Hz sampling frequency (before oversampling) input signal, and keep the pinking gain at the same level from that point to 0Hz.

The effects of the oversampling, pre-emphasis and pinking can be combined. We shall call the frequency responses obtained this way *modified frequency responses*. From a modified frequency response, we shall find the maximum (peak) magnitude frequency response from the stopbands, convert that to dB, flip the sign, and call this value the *modified SNR* (signal-to-noise ratio) and presume that it is a rather good and comparable measure of the quality of an interpolator.

Interpolating non-oversampled data is out of the scope of this kind of a comparison. There would be problems with defining the passband-to-stopband transition band. With the presumption of an ideal oversampling filter, the transition bands of the interpolator are rendered invisible. Also, the passband attenuation is not an issue because of pre-emphasis, which shows its price at the stopbands.

4. New optimal designs

With the modified SNR as a quality measure, it was possible to design the best possible interpolators of chosen oversampling ratios, orders and numbers of points.

The optimization was done directly on the impulse response coefficients, using Differential Evolution[3], a genetic algorithm developed by Rainer Storn and Kenneth Price. In short, the algorithm finds (or at least tries to) the global minimum of a cost function that takes a parameter vector as an argument, which in this case consisted of the coefficients of the polynomial(s). In the cost function, the six first stopbands in the modified frequency response were sampled at 33 positions each, and the largest magnitude was given as the cost which was then minimized by the Differential Evolution algorithm. The normalization for unity gain at DC was also implemented as an added penalty in the cost function.

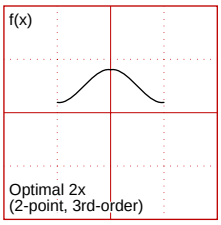
Pre-emphasis was excluded from the cost function by accident, but it later turned out that this was necessary to prevent the interpolators from developing ridiculously huge transition band magnitude peaks.

Here are the impulse responses of all the potentially useful generated interpolators for oversampling ratios 2, 4, 8, 16 and 32. Note that there is some air in the decimals of the coefficients, so some further quantization is OK.

4.1 2-point, 3rd-order optimal

As can be seen from the following, 2-point, 3rd-order optimal interpolators converge to linear interpolation as the oversampling ratio increases. This is an indication to use linear interpolation at very high oversampling ratios.

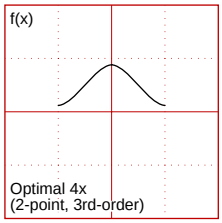
$$f(x) = \begin{cases} 1.57015627178718420x^3 - 2.35977550974341630x^2 + \\ 0.17594740788514596x + 0.80607906469176971 & 0 \le x < 1 \\ \hline 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 2-point, 3rd-order optimal 2x impulse response

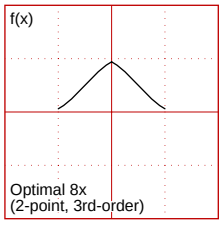
[3] http://www.icsi.berkeley.edu/~storn/code.html

$$f(x) = \begin{cases} 1.32598918957298410x^3 - 1.99054787320203810x^2 - \\ 0.10012219395448523x + 0.88207975731800936 & 0 \le x < 1 \\ \hline 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



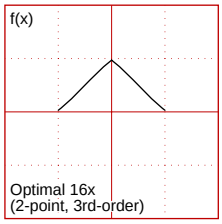
Formula, figure: 2-point, 3rd-order optimal 4x impulse response

$$f(x) = \begin{cases} 0.73514591836770027x^3 - 1.10319974084152170x^2 - \\ 0.51213628865925998x + 0.94001491168487883 & 0 \le x < 1 \\ \hline 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



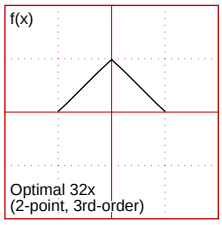
Formula, figure: 2-point, 3rd-order optimal 8x impulse response

$$f(x) = \begin{cases} 0.38606621963374965x^3 - 0.57923093055631791x^2 - \\ 0.74617479745643256x + 0.96964782067188493 & 0 \le x < 1 \\ \hline 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 2-point, 3rd-order optimal 16x impulse response

$$f(x) = \begin{cases} 0.19775766248673177x^3 - 0.29667081825572522x^2 - \\ 0.87053863725307623x + 0.98472017575676363 & 0 \le x < 1 \\ \hline 0 & 1 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 2-point, 3rd-order optimal 32x impulse response

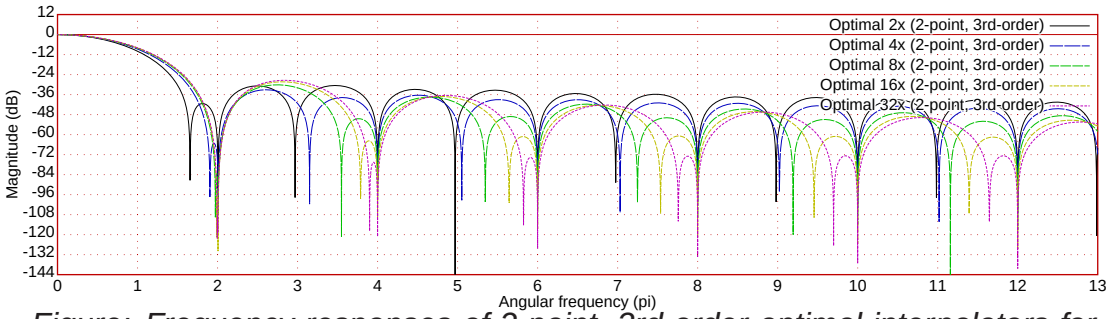


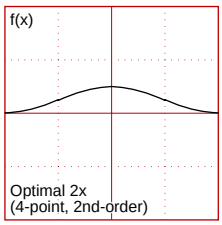
Figure: Frequency responses of 2-point, 3rd-order optimal interpolators for different oversampling ratios

The modified frequency responses will be shown in the comparison section of this paper.

4.2 4-point, 2nd-order optimal

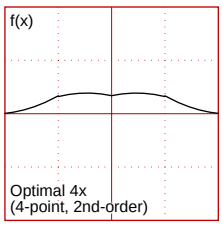
The 4-point, 2nd-order optimal interpolators are a bit strange - the impulse responses, especially the higher oversampling ratio versions, do not even resemble anything that we have previously seen. The explanation is that there is a transfer function zero on the transition band. This causes a large sidelobe that at higher oversampling ratios exceeds unity in magnitude.

$$f(x) = \begin{cases} -0.21343978756177684x^2 - 0.04782068534965925x + 0.50061662213752656 & 0 \le x < 1 \\ 0.21303593243799016x^2 - 0.88689658749623701x + 0.92770135528027386 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



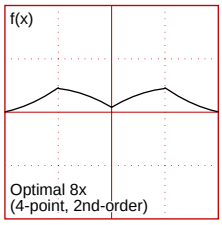
Formula, figure: 4-point, 2nd-order optimal 2x impulse response

$$f(x) = \begin{cases} -0.22865399531858188x^2 + 0.21144498075197282x + 0.33820365736567115 & 0 \le x < 1 \\ 0.22858390767180370x^2 - 1.01414466618792900x + 1.12014639874555470 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



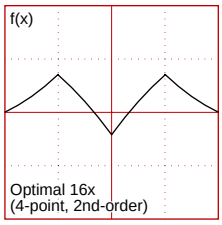
Formula, figure: 4-point, 2nd-order optimal 4x impulse response

$$f(x) = \begin{cases} -0.24005206207889518x^2 + 0.59257579283164508x + 0.09224718574204172 & 0 \le x < 1 \\ 0.24004281672637814x^2 - 1.17126532964206100x + 1.38828036063664320 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



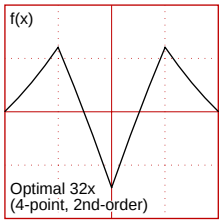
Formula, figure: 4-point, 2nd-order optimal 8x impulse response

$$f(x) = \begin{cases} -0.24506117865474364x^2 + 1.36361593203840510x - 0.41849525763976203 & 0 \le x < 1 \\ 0.24506002360805534x^2 - 1.44144384373471430x + 1.90873339502208310 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 2nd-order optimal 16x impulse response

$$f(x) = \begin{cases} -0.24755243839713828x^2 + 2.87083485132510450x - 1.42170796824052890 & 0 \le x < 1 \\ 0.24755229501840223x^2 - 1.95043794419108290x + 2.91684291662070860 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 2nd-order optimal 32x impulse response

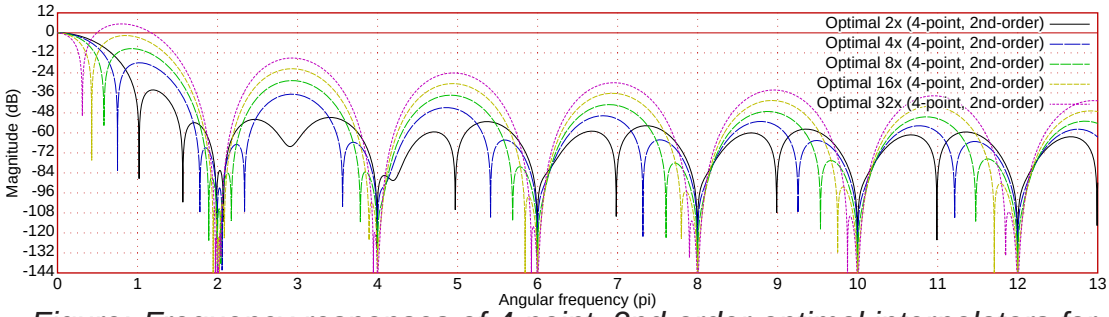
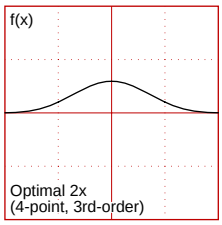


Figure: Frequency responses of 4-point, 2nd-order optimal interpolators for different oversampling ratios

4.3 4-point, 3rd-order optimal

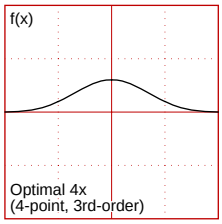
Visually, the impulse responses of the 4-point, 3rd-order optimal interpolators resemble that of the B-spline. The frequency response shows groups of zeros bombarding the stopbands for greater modified frequency response flatness.

$$f(x) = \begin{cases} 0.36030925263849456x^3 - 0.78664888597764893x^2 + 0.03573669883299365x + 0.59244492420272321 & 0 \le x < 1 \\ -0.10174985775982505x^3 + 0.70401463131621556x^2 - 1.60101160971478710x + 1.20220428331406090 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



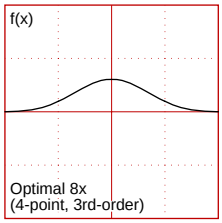
Formula, figure: 4-point, 3rd-order optimal 2x impulse response

$$f(x) = \begin{cases} 0.42912649274763925x^3 - 0.89223007211175309x^2 + 0.05694012453786401x + 0.60304009430474115 & 0 \le x < 1 \\ -0.13963062613760227x^3 + 0.87687351895686727x^2 - 1.85072890189700660x + 1.31228823423882930 & 1 \le x < 2 \\ 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



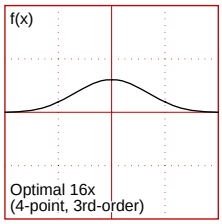
Formula, figure: 4-point, 3rd-order optimal 4x impulse response

$$f(x) = \begin{cases} 0.46789242171187317x^3 - 0.95149675410360302x^2 + \\ 0.07280793921972525x + 0.60658368706046584 & 0 \le x < 1 \\ \hline -0.15551896027602030x^3 + 0.94949311590826524x^2 - \\ 1.95618744839533010x + 1.35919815911169020 & 1 \le x < 2 \\ \hline 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



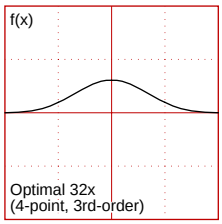
Formula, figure: 4-point, 3rd-order optimal 8x impulse response

$$f(x) = \begin{cases} 0.48601256046234864x^3 - 0.97894238166068270x^2 + \\ 0.07980169577604959x + 0.60844825096346644 & 0 \le x < 1 \\ \hline -0.16195131297091253x^3 + 0.97870442828560433x^2 - \\ 1.99807048591354810x + 1.37724137476464990 & 1 \le x < 2 \\ \hline 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 3rd-order optimal 16x impulse response

$$f(x) = \begin{cases} 0.49369595780454456x^3 - 0.99052586766084594x^2 + \\ 0.08298544053689563x + 0.60908264223655417 & 0 \le x < 1 \\ \hline -0.16455902278580614x^3 + 0.99049753216621961x^2 - \\ 2.01496368680360890x + 1.38455689452848450 & 1 \le x < 2 \\ \hline 0 & 2 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 3rd-order optimal 32x impulse response

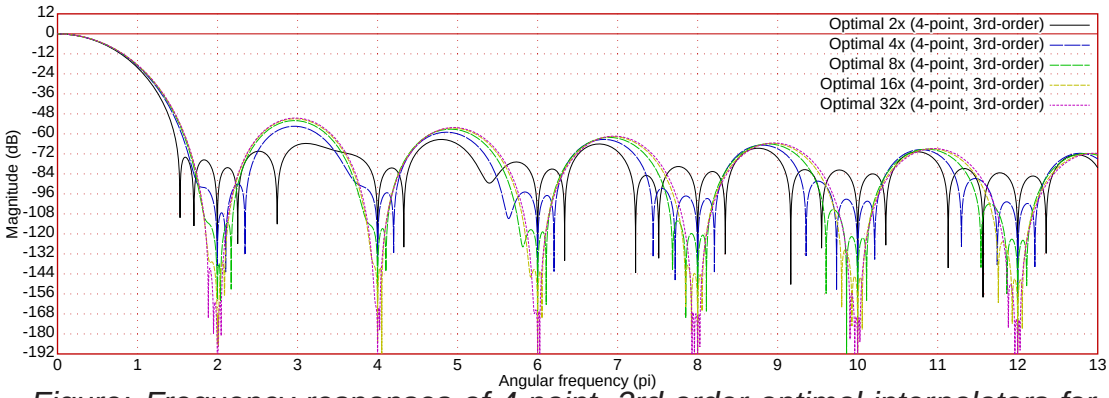
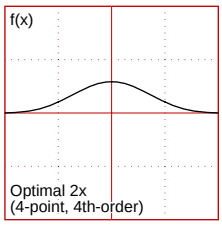


Figure: Frequency responses of 4-point, 3rd-order optimal interpolators for different oversampling ratios

4.4 4-point, 4th-order optimal

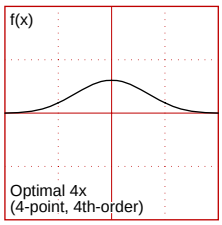
The 4-point, 4th-order optimal interpolators don't differ much from the 3rd-order ones. They are better, though.

$$f(x)=\begin{cases} 0.04252164479749607x^4+0.29412762852131868x^3- & 0\le x<1 \\ 0.75864870418278074x^2+0.04442540676862300x+ & \\ 0.58448510036125145 & \\ \hline -0.04289144034653719x^4+0.13781898240764315x^3+ & 1\le x<2 \\ 0.21256821036268256x^2-1.16581445347275190x+ & \\ 1.06598379704160570 & \\ \hline 0 & 2\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



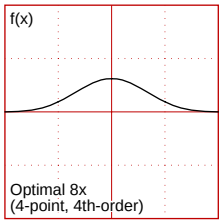
Formula, figure: 4-point, 4th-order optimal 2x impulse response

$$f(x)=\begin{cases} 0.00986988334359864x^4+0.44922093286355397x^3- & 0\le x<1 \\ 0.94057832565094635x^2+0.06128937679587994x+ & \\ 0.61340295990566229 & \\ \hline -0.00989340017126506x^4-0.09642760567543440x^3+ & 1\le x<2 \\ 0.81943257721092366x^2-1.82814511658458520x+ & \\ 1.30835018075821670 & \\ \hline 0 & 2\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



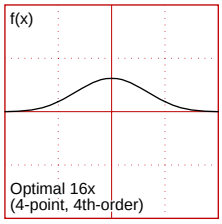
Formula, figure: 4-point, 4th-order optimal 4x impulse response

$$f(x)=\begin{cases} 0.00255074537015887x^4+0.48698871865064902x^3- & 0\le x<1 \\ 0.98489647972932193x^2+0.06389302461261143x+ & \\ 0.62095991632974834 & \\ \hline -0.00255226912537286x^4-0.14868053358928229x^3+ & 1\le x<2 \\ 0.95410568622888214x^2-1.97277963497287720x+ & \\ 1.35943398999940390 & \\ \hline 0 & 2\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



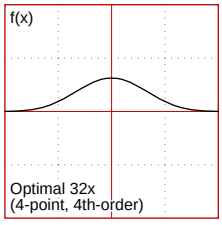
Formula, figure: 4-point, 4th-order optimal 8x impulse response

$$f(x)=\begin{cases} 0.00064264050033187x^4+0.49672182806667398x^3- & 0\le x<1 \\ 0.99620011474430481x^2+0.06443376638262904x+ & \\ 0.62293049365660191 & \\ \hline -0.00064273459469381x^4-0.16214364417487748x^3+ & 1\le x<2 \\ 0.98847675044522398x^2-2.00931632449031920x+ & \\ 1.37216269878963180 & \\ \hline 0 & 2\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 4th-order optimal 16x impulse response

$$f(x)=\begin{cases} 0.00016095224137360x^4+0.49917660509564427x^3- & 0\le x<1 \\ 0.99904509583176049x^2+0.06456923251842608x+ & \\ 0.62342449465938121 & \\ \hline -0.00016095810460478x^4-0.16553360612350931x^3+ & 1\le x<2 \\ 0.99711292321092770x^2-2.01847637982642340x+ & \\ 1.37534629142898650 & \\ \hline 0 & 2\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 4-point, 4th-order optimal 32x impulse response

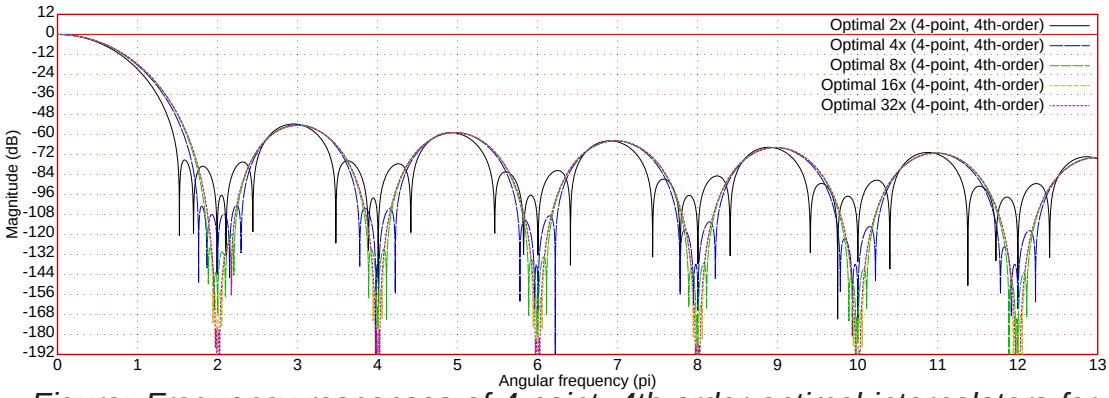
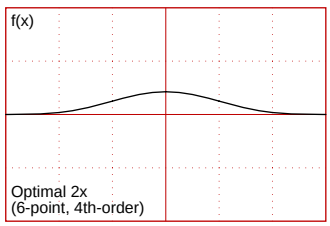


Figure: Frequency responses of 4-point, 4th-order optimal interpolators for different oversampling ratios

4.5 6-point, 4th-order optimal

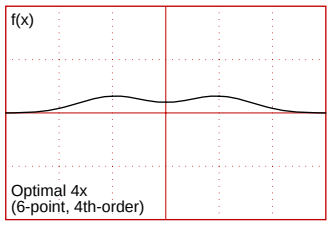
The same phenomenon can be seen with the 6-point, 4th-order optimal interpolators as with the 4-point, 2nd-order ones. The transition band zero makes the impulse response weird-looking and causes a large sidelobe in the transition band. The sidelobe height greatly exceeds unity with higher oversampling ratios. This may cause design problems in the oversampling stage in form of increased stopband attenuation requirements.

$$f(x) = \begin{cases} 0.03134095684084392x^4 + 0.00255494211547300x^3 - 0.20486985491012843x^2 - 0.00525580294341423x + 0.42640922432669054 & 0 \le x < 1 \\ -0.04385804833432710x^4 + 0.31182026815653541x^3 - 0.70564644117967990x^2 + 0.37868437559565432x + 0.30902529029941583 & 1 \le x < 2 \\ 0.01249475765486819x^4 - 0.16695522597587154x^3 + 0.83217835730406542x^2 - 1.83761742915820410x + 1.51897639740576910 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



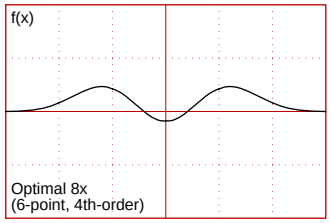
Formula, figure: 6-point, 4th-order optimal 2x impulse response

$$f(x)=\begin{cases} 0.02881527997393852x^4-0.42059475673758634x^3+ & 0\le x<1 \\ 0.56468711069379207x^2-0.06119274485321008x+ & \\ 0.20167941634921072 & \\ \hline -0.04250898918476453x^4+0.51926458031522660x^3- & 1\le x<2 \\ 1.85350543411307390x^2+2.33580825807694700x- & \\ 0.64579641436229407 & \\ \hline 0.01369173779618459x^4-0.22283280665600644x^3+ & 2\le x<3 \\ 1.27147464005834010x^2-3.09936092833253300x+ & \\ 2.76228852293285200 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



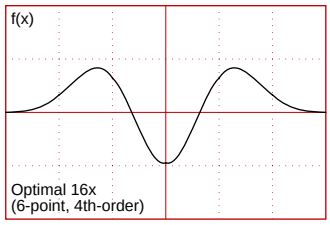
Formula, figure: 6-point, 4th-order optimal 4x impulse response

$$f(x)=\begin{cases} 0.03401038103941584x^4-1.15976496200057480x^3+ & 0\le x<1 \\ 1.87551558979819120x^2-0.15190225510786248x- & \\ 0.17436452172055789 & \\ \hline -0.05090907029392906x^4+0.93463067895166918x^3- & 1\le x<2 \\ 3.92391712129699590x^2+5.73320660746477540x- & \\ 2.26955357035241170 & \\ \hline 0.01689861603514873x^4-0.32814290420019698x^3+ & 2\le x<3 \\ 2.04584149450148180x^2-5.25661448354449060x+ & \\ 4.84834508915762540 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



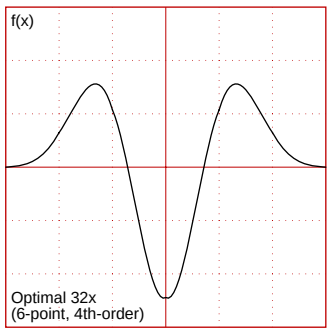
Formula, figure: 6-point, 4th-order optimal 8x impulse response

$$f(x)=\begin{cases} 0.03755086455339280x^4-2.64598691215356660x^3+ & 0\le x<1 \\ 4.53807483241466340x^2-0.33649680079382827x- & \\ 0.94730014688427577 & \\ \hline -0.05631219122315393x^4+1.70665858343069510x^3- & 1\le x<2 \\ 7.98288364772738750x^2+12.52871168241192600x- & \\ 5.55035312316726960 & \\ \hline 0.01876132424143207x^4-0.49470749109917245x^3+ & 2\le x<3 \\ 3.44447036756440590x^2-9.37021675593126700x+ & \\ 8.94785524286246310 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 4th-order optimal 16x impulse response

$$f(x) = \begin{cases} 0.03957507923965987x^4 - 5.50592307590218160x^3 + 9.67889243081689440x^2 - 0.69468212315980082x - 2.44391738331193720 & 0 \le x < 1 \\ -0.05936083498715066x^4 + 3.15288929279855570x^3 - 15.73068663442630400x^2 + 25.58633277328986500x - 11.87524595267807600 & 1 \le x < 2 \\ 0.01978575568000696x^4 - 0.79053754554850286x^3 + 6.05175140696421730x^2 - 17.17264148794549100x + 16.79403235763479100 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 4th-order optimal 32x impulse response

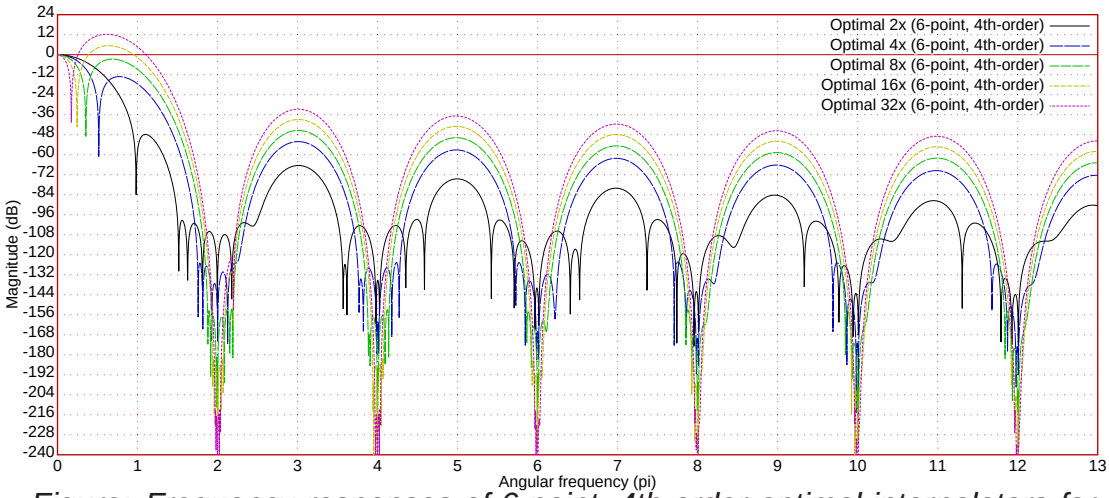
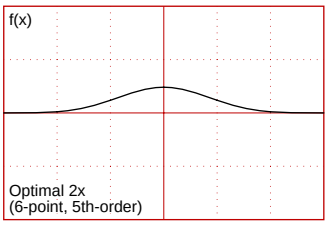


Figure: Frequency responses of 6-point, 4th-order optimal interpolators for different oversampling ratios

4.6 6-point, 5th-order optimal

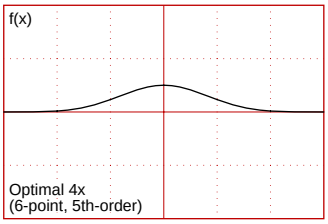
The 6-point, 5th-order optimal interpolator impulse responses resemble that of the B-spline, and the frequency responses look nice - lots of zeros at where they are mostly needed.

$$f(x)=\begin{cases} -0.04317950185225609x^5+0.14640674192652170x^4- & 0\le x<1 \\ 0.02014846731685776x^3-0.32675071713952775x^2- & \\ 0.00127577239632662x+0.48217702203158502 & \\ \hline 0.01802814255926417x^5-0.19234043023690772x^4+ & 1\le x<2 \\ 0.74995484587342742x^3-1.22477236472789920x^2+ & \\ 0.53534756396439365x+0.35095903476754237 & \\ \hline -0.00152170021558204x^5+0.03768876199398620x^4- & 2\le x<3 \\ 0.31577407091450355x^3+1.22220278720010690x^2- & \\ 2.26168360510917840x+1.62814578813495040 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



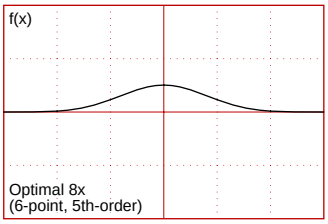
Formula, figure: 6-point, 5th-order optimal 2x impulse response

$$f(x)=\begin{cases} -0.06607747864416924x^5+0.20620318519804220x^4- & 0\le x<1 \\ 0.04512026308730401x^3-0.36229943140977111x^2- & \\ 0.00256790184606694x+0.50164509338655083 & \\ \hline 0.03255079211953620x^5-0.30560854964737405x^4+ & 1\le x<2 \\ 1.08365113099941970x^3-1.66940481896969310x^2+ & \\ 0.78336433172501685x+0.30718330223223800 & \\ \hline -0.00628989632244913x^5+0.09909173357642603x^4- & 2\le x<3 \\ 0.62765808573554227x^3+1.99766476840488070x^2- & \\ 3.19403437421534920x+2.05191571792256240 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



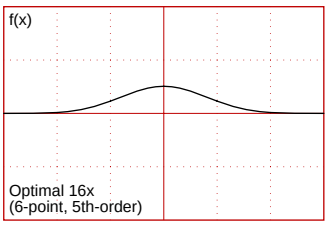
Formula, figure: 6-point, 5th-order optimal 4x impulse response

$$f(x)=\begin{cases} -0.07517133281176167x^5+0.22942797169644802x^4- & 0\le x<1 \\ 0.06070462616102962x^3-0.36434084624989699x^2- & \\ 0.00368143670114908x+0.50513183702821474 & \\ \hline 0.03751837438141215x^5-0.34363487882262922x^4+ & 1\le x<2 \\ 1.19588167464050650x^3-1.82581238657617080x^2+ & \\ 0.88385964850687193x+0.28281884957695946 & \\ \hline -0.00747588873055296x^5+0.11419603882898799x^4- & 2\le x<3 \\ 0.70370361187427199x^3+2.18592382088982260x^2- & \\ 3.42137079071284810x+2.15756386503245070 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



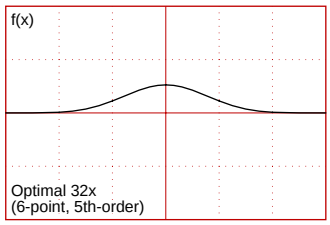
Formula, figure: 6-point, 5th-order optimal 8x impulse response

$$f(x)=\begin{cases} -0.07990500783668089x^5+0.24139298776307896x^4- & 0\le x<1 \\ 0.06616250180411522x^3-0.36990908725555449x^2- & \\ 0.00387117789818541x+0.50819303579369868 & \\ \hline 0.03994519162531633x^5-0.36203450650610985x^4+ & 1\le x<2 \\ 1.24834464824612510x^3-1.89281840112089440x^2+ & \\ 0.91870010875159547x+0.27758734130911511 & \\ \hline -0.00798609327859495x^5+0.12064126711558003x^4- & 2\le x<3 \\ 0.73559668875725392x^3+2.26228244623301580x^2- & \\ 3.50786533926449100x+2.19284545406407450 & \\ \hline 0 & 3\le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order optimal 16x impulse response

$$f(x) = \begin{cases} -0.08349799235675044x^5 + 0.25041444762720882x^4 - 0.04095676092513167x^3 - 0.42682321682847008x^2 + 0.00010896283126635x + 0.52558916128536759 & 0 \le x < 1 \\ 0.04174912841630993x^5 - 0.37562266426589430x^4 + 1.27215033630638800x^3 - 1.86228986389877100x^2 + 0.80946953063234006x + 0.33937904183610190 & 1 \le x < 2 \\ -0.00834987866042734x^5 + 0.12520821766375972x^4 - 0.75510203509083995x^3 + 2.28912105276248390x^2 - 3.48774662195185850x + 2.13606003964474490 & 2 \le x < 3 \\ 0 & 3 \le x \\ f(-x) & \text{otherwise.} \end{cases}$$



Formula, figure: 6-point, 5th-order optimal 32x impulse response

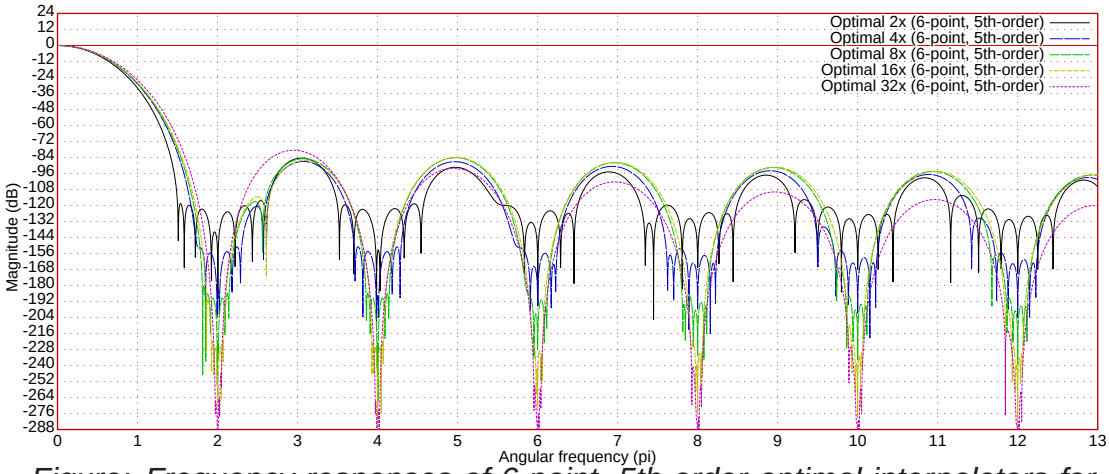


Figure: Frequency responses of 6-point, 5th-order optimal interpolators for different oversampling ratios

5. Comparison

This comparison evaluates the modified SNR for each presented interpolator, at different oversampling ratios. The optimal interpolators are only evaluated at their specific oversampling ratios.

5.1 Linear

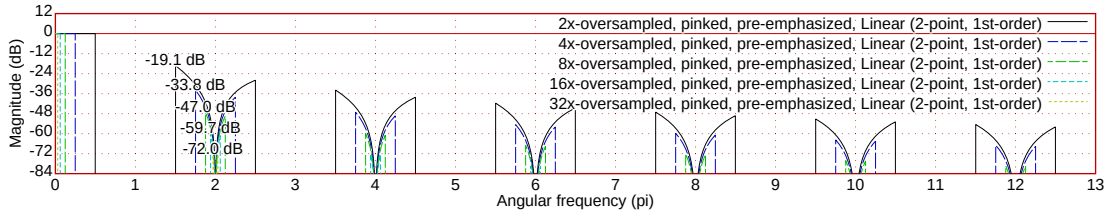


Figure: Modified frequency responses of the linear interpolator

Linear interpolation gives a modified SNR of 19.1dB for 2x-, 33.8dB for 4x-, 47.0dB for 8x-, 59.7dB for 16x- and 72.0dB for 32x-oversampled input.

5.2 B-spline

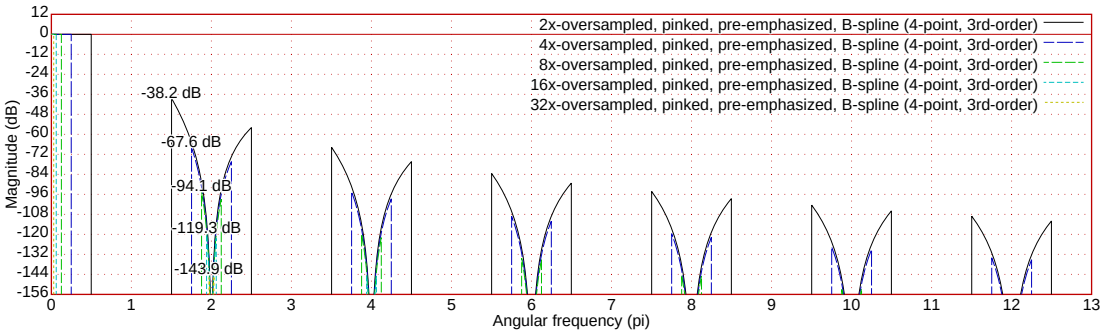


Figure: Modified frequency responses of the 3rd-order B-spline

3rd-order B-spline gives a modified SNR of 38.2dB for 2x-, 67.6dB for 4x-, 94.1dB for 8x-, 119.3dB for 16x- and 143.9dB for 32x-oversampled input.

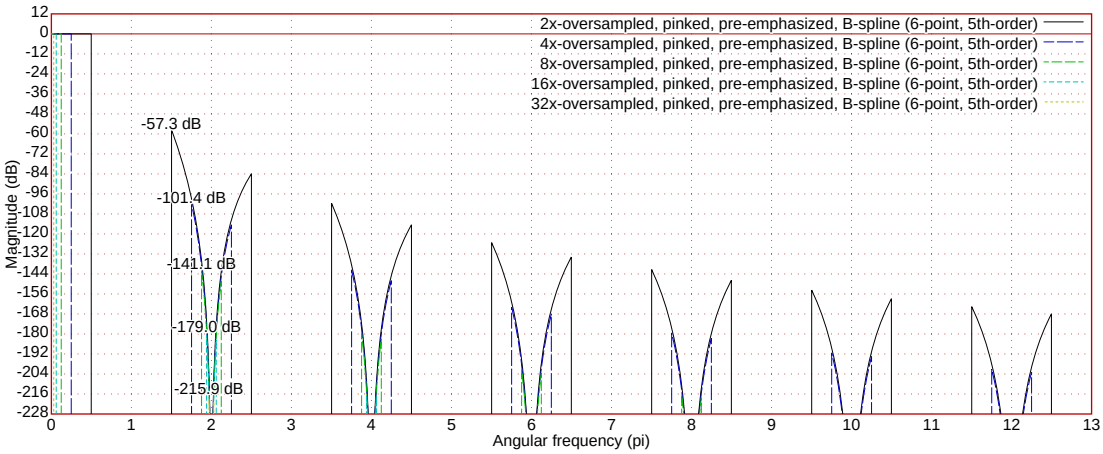


Figure: Modified frequency responses of the 5th-order B-spline

5th-order B-spline gives a modified SNR of 57.3dB for 2x-, 101.4dB for 4x-, 141.1dB for 8x-, 179.0dB for 16x- and 215.9dB for 32x-oversampled input.

5.3 Lagrange

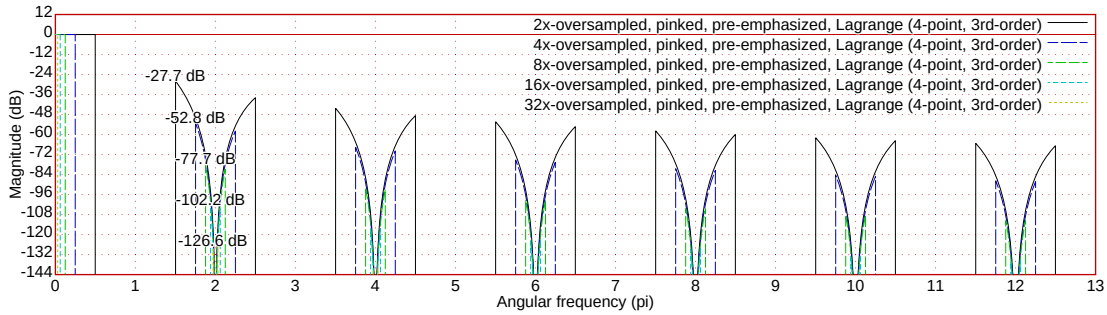


Figure: Modified frequency responses of the 3rd-order Lagrange

3rd-order Lagrange gives a modified SNR of 27.7dB for 2x-, 52.8dB for 4x-, 77.7dB for 8x-, 102.2dB for 16x- and 126.6dB for 32x-oversampled input.

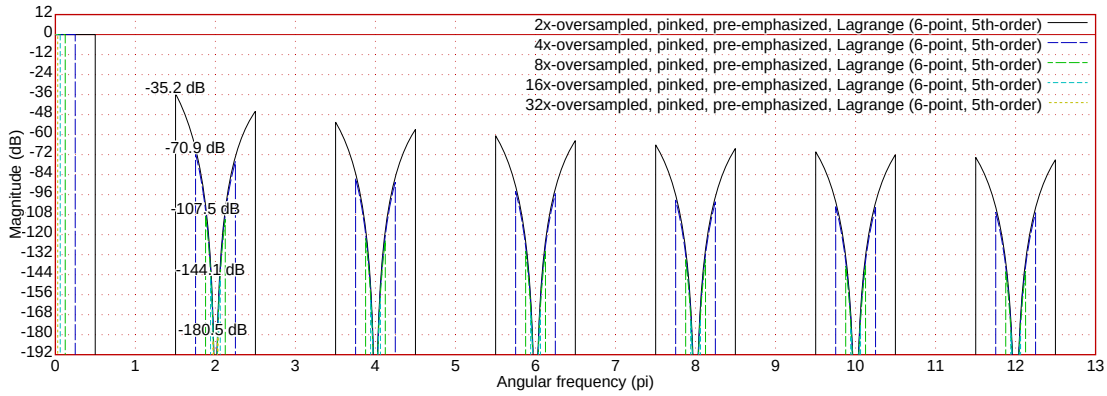


Figure: Modified frequency responses of the 5th-order Lagrange

5th-order Lagrange gives a modified SNR of 35.2dB for 2x-, 70.9dB for 4x-, 107.5dB for 8x-, 144.1dB for 16x- and 180.5dB for 32x-oversampled input.

5.4 Hermite

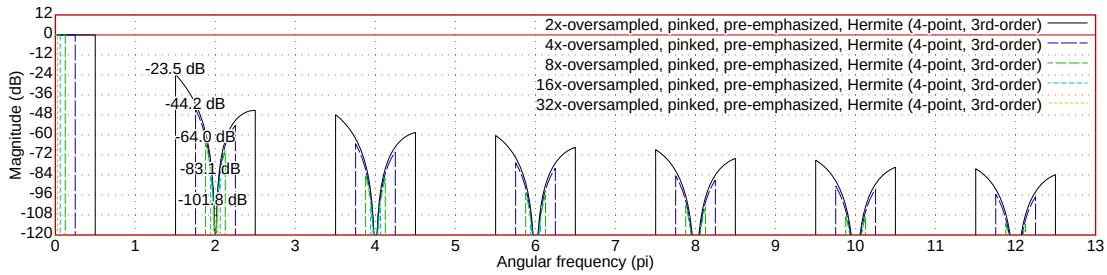


Figure: Modified frequency responses of the 4-point, 3rd-order Hermite

4-point, 3rd-order Hermite gives a modified SNR of 23.5dB for 2x-, 44.2dB for 4x-, 64.0dB for 8x-, 83.1dB for 16x- and 101.8dB for 32x-oversampled input.

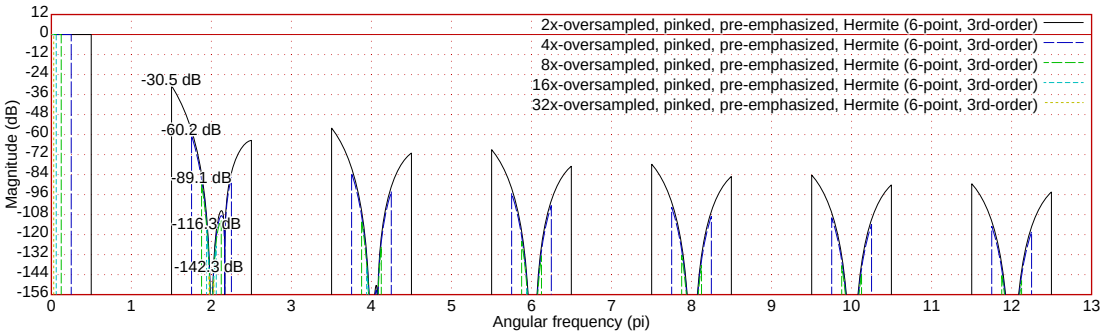


Figure: Modified frequency responses of the 6-point, 3rd-order Hermite

6-point, 3rd-order Hermite gives a modified SNR of 30.5dB for 2x-, 60.2dB for 4x-, 89.1dB for 8x-, 116.3dB for 16x- and 142.3dB for 32x-oversampled input.

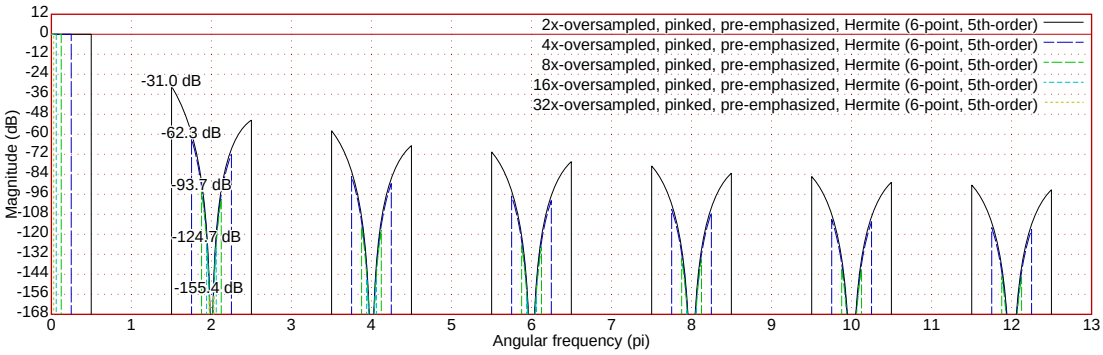


Figure: Modified frequency responses of the 6-point, 5th-order Hermite

6-point, 5th-order Hermite gives a modified SNR of 31.0dB for 2x-, 62.3dB for 4x-, 93.7dB for 8x-, 124.7dB for 16x- and 155.4dB for 32x-oversampled input.

5.5 2nd-order-osculating

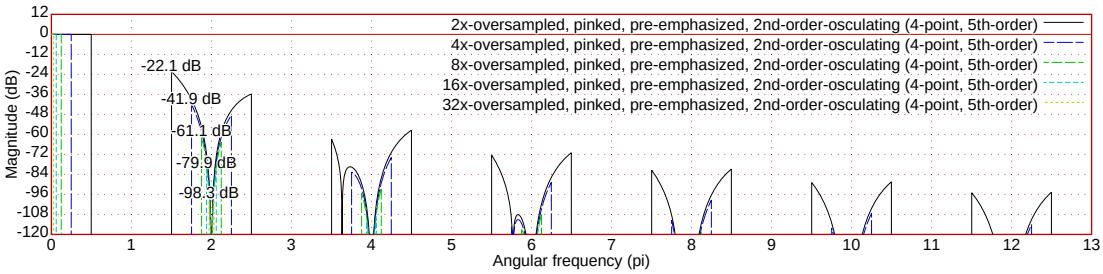


Figure: Modified frequency responses of the 4-point, 5th-order 2nd-order-osculating

4-point, 5th-order 2nd-order-osculating gives a modified SNR of 22.1dB for 2x-, 41.9dB for 4x-, 61.1dB for 8x-, 79.9dB for 16x- and 98.3dB for 32x-oversampled input.

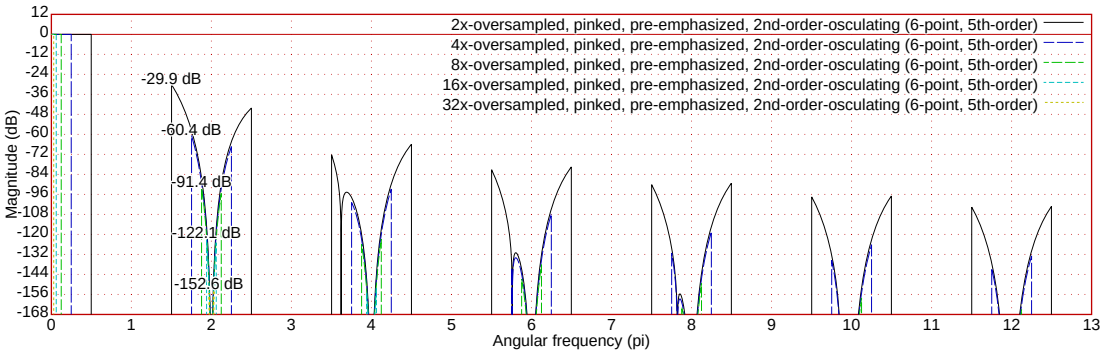


Figure: Modified frequency responses of the 6-point, 5th-order 2nd-order-osculating

6-point, 5th-order 2nd-order-osculating gives a modified SNR of 29.9dB for 2x-, 60.4dB for 4x-, 91.4dB for 8x-, 122.1dB for 16x- and 152.6dB for 32x-oversampled input.

5.6 Watte tri-linear

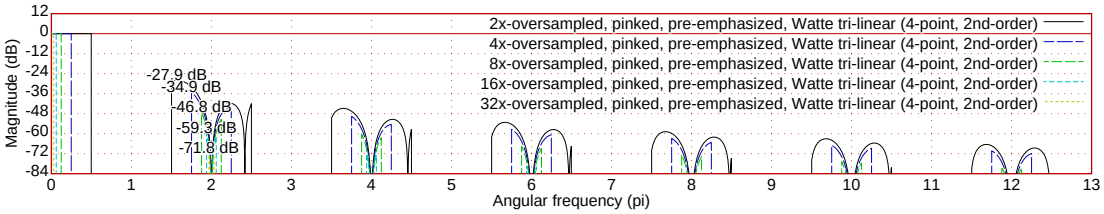


Figure: Modified frequency responses of the Watte tri-linear

Watte tri-linear gives a modified SNR of 27.9dB for 2x-, 34.9dB for 4x-, 46.8dB for 8x-, 59.3dB for 16x- and 71.8dB for 32x-oversampled input.

5.7 Parabolic 2x

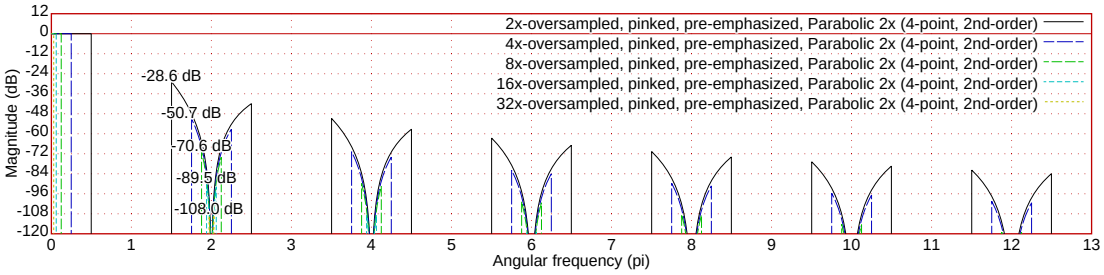


Figure: Modified frequency responses of the parabolic 2x

Parabolic 2x gives a modified SNR of 28.6dB for 2x-, 50.7dB for 4x-, 70.6dB for 8x-, 89.5dB for 16x- and 108.0dB for 32x-oversampled input.

5.8 Optimal

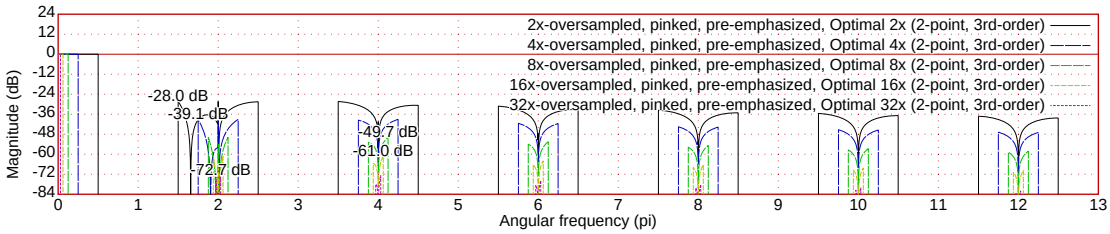


Figure: Modified frequency responses of the 2-point, 3rd-order optimal interpolators

2-point, 3rd-order optimal interpolators give modified SNRs of 28.0dB for 2x-, 39.1dB for 4x-, 49.7dB for 8x-, 61.0dB for 16x- and 72.7dB for 32x-oversampled input.

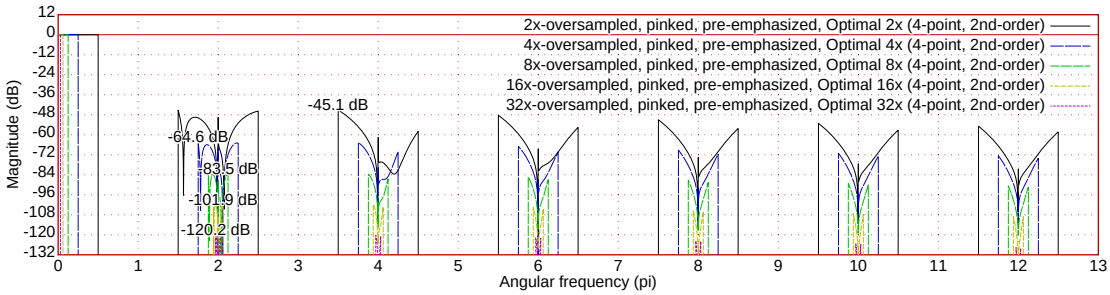


Figure: Modified frequency responses of the 4-point, 2nd-order optimal interpolators

4-point, 2nd-order optimal interpolators give modified SNRs of 45.1dB for 2x-, 64.6dB for 4x-, 83.5dB for 8x-, 101.9dB for 16x- and 120.2dB for 32x-oversampled input.

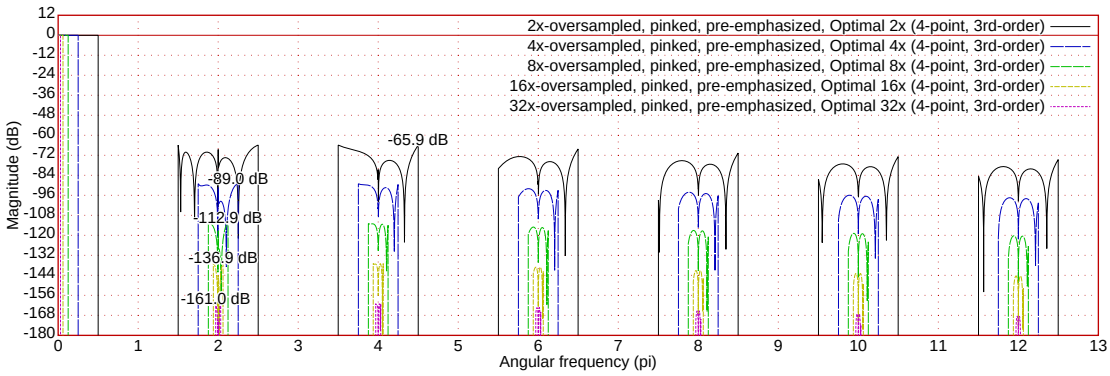


Figure: Modified frequency responses of the 4-point, 3rd-order optimal interpolators

4-point, 3rd-order optimal interpolators give modified SNRs of 65.9dB for 2x-, 89.0dB for 4x-, 112.9dB for 8x-, 136.9dB for 16x- and 161.0dB for 32x-oversampled input.

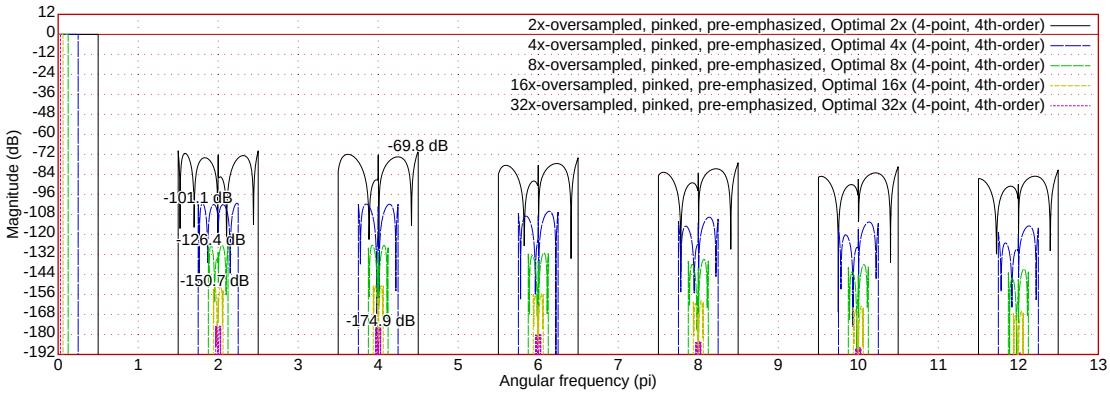


Figure: Modified frequency responses of the 4-point, 4th-order optimal interpolators

4-point, 4th-order optimal interpolators give modified SNRs of 69.8dB for 2x-, 101.1dB for 4x-, 126.4dB for 8x-, 150.7dB for 16x- and 174.9dB for 32x-oversampled input.

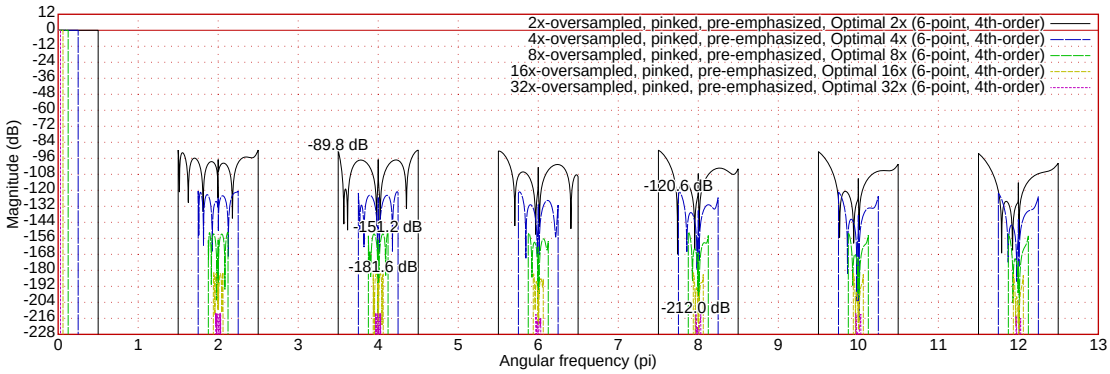


Figure: Modified frequency responses of the 6-point, 4th-order optimal interpolators

6-point, 4th-order optimal interpolators give modified SNRs of 89.8dB for 2x-, 120.6dB for 4x-, 151.2dB for 8x-, 181.6dB for 16x- and 212.0dB for 32x-oversampled input.

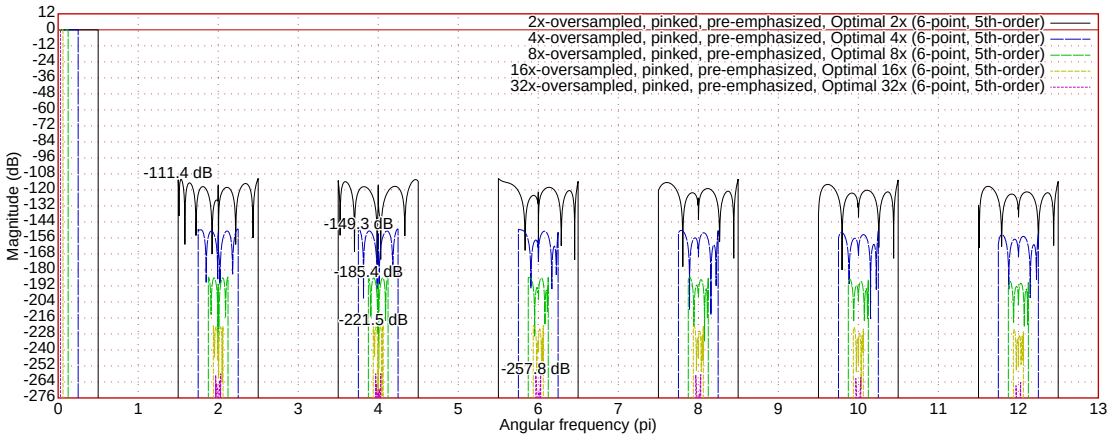


Figure: Modified frequency responses of the 6-point, 5th-order optimal interpolators

6-point, 5th-order optimal interpolators give modified SNRs of 111.4dB for 2x-, 149.3dB for 4x-, 185.4dB for 8x-, 221.5dB for 16x- and 257.8dB for 32x-oversampled input.

A summary of the comparison will be given after evaluating the implementation complexity of each interpolator.

6. Implementation

The impulse response form of an interpolator is not the one used in actual code. The interpolation routine can be simplified to the problem of interpolating the samples $y[x]$ in range $x = 0..1$. The position to interpolate is given in variable x and the nearby samplepoints are provided, say $y[-1]$, $y[0]$, $y[1]$ and $y[2]$ for a 4-point interpolator. The interpolation can be described as convolution with the interpolator impulse response – here's a demonstration on cubic Hermite.

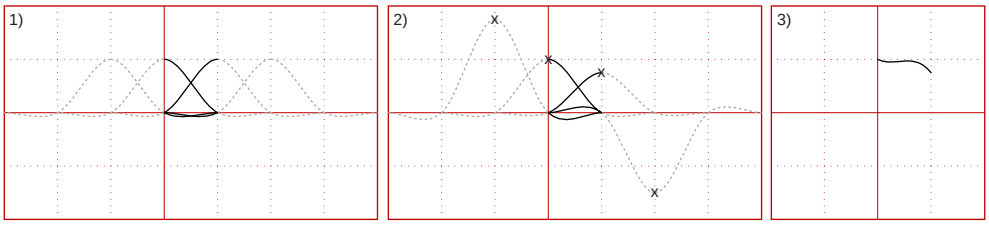


Figure: 1) Shifted cubic Hermite impulse responses form the basis functions, $f_i(x)$ (the non-grayed, non-dashed sections), 2) Basis functions scaled by the samplepoints $y[-1] = 1.75$, $y[0] = 1$, $y[1] = 0.75$, $y[2] = -1.5$, 3) Sum of the scaled basis functions becomes an interpolation section $g(x)$

The basis functions, $f_i(x)$, are the impulse response sections shifted in time to range $x = 0..1$. They are scaled by the samplepoints and summed to form the the interpolation function section $g(x)$, from which the value at a chosen x is taken.

The following formulations for the basis functions $f_i(x)$ for cubic Hermite were solved by substituting x with $x - i$ in the impulse response $f(x)$.

$$\begin{aligned} f_{-1}(x) &= -\tfrac{1}{2}x + x^2 - \tfrac{1}{2}x^3 \\ f_0(x) &= 1 - \tfrac{5}{2}x^2 + \tfrac{3}{2}x^3 \\ f_1(x) &= \tfrac{1}{2}x + 2x^2 - \tfrac{3}{2}x^3 \\ f_2(x) &= -\tfrac{1}{2}x^2 + \tfrac{1}{2}x^3 \end{aligned}$$

Formula: 4-point, 3rd-order Hermite basis functions

These basis functions are next scaled by $y[-1]$, $y[0]$, $y[1]$ and $y[2]$, respectively, and summed to form $g(x)$, which is the interpolation function in range $x = 0..1$.

$$
\begin{aligned}
g(x) &= y[-1]\,f_{-1}(x)+y[0]\,f_0(x)+y[1]\,f_1(x)+y[2]\,f_2(x) \\
&= \begin{array}{lllllll}
y[-1] & (& & -\frac{1}{2}x & +x^2 & -\frac{1}{2}x^3) & + \\
y[0] & (& 1 & & -\frac{5}{2}x^2 & +\frac{3}{2}x^3) & + \\
y[1] & (& & \frac{1}{2}x & +2x^2 & -\frac{3}{2}x^3) & + \\
y[2] & (& & & -\frac{1}{2}x^2 & +\frac{1}{2}x^3) &
\end{array} \\
&= \begin{array}{lllllll}
 & (& & y[0] & & &) \; + \\
x & (& -\frac{1}{2}y[-1] & & +\frac{1}{2}y[1] & &) \; + \\
x^2 & (& y[-1] & -\frac{5}{2}y[0] & +2y[1] & -\frac{1}{2}y[2]) & + \\
x^3 & (& -\frac{1}{2}y[-1] & +\frac{3}{2}y[0] & -\frac{3}{2}y[1] & +\frac{1}{2}y[2]) &
\end{array}
\end{aligned}
$$

A matrix representation simplifies the notation.

$$
g(x) = \begin{pmatrix} 1 & x & x^2 & x^3 \end{pmatrix}
\begin{pmatrix}
0 & 1 & 0 & 0 \\
-\frac{1}{2} & 0 & \frac{1}{2} & 0 \\
1 & -\frac{5}{2} & 2 & -\frac{1}{2} \\
-\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2}
\end{pmatrix}
\begin{pmatrix} y[-1] \\ y[0] \\ y[1] \\ y[2] \end{pmatrix}
$$

Formula: The x-form implementation of cubic Hermite interpolation

In the coefficient matrix, if there are numbers on a row that have the same absolute value, the corresponding multiplications can be combined, increasing the speed of the interpolation routine. The symmetry of the impulse responses can be exploited to limit the number of required multiplications to half the number of matrix elements, or below if we are lucky, by a substitution $x = \frac{1}{2}+z$, which shifts the symmetry axis to $z = 0$. This trick may not be effective for matrices that already have many pairable elements, zeros or minus or plus ones that need not be multiplied. Nevertheless, we shall demonstrate this on the cubic Hermite.

$$
g(x) = \begin{pmatrix} 1 & z & z^2 & z^3 \end{pmatrix}
\begin{pmatrix}
-\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\
\frac{1}{8} & -\frac{11}{8} & \frac{11}{8} & -\frac{1}{8} \\
\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\
-\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2}
\end{pmatrix}
\begin{pmatrix} y[-1] \\ y[0] \\ y[1] \\ y[2] \end{pmatrix}, \quad z = x - \frac{1}{2}
$$

Formula: The z-form implementation of cubic Hermite interpolation

We shall call these implementations the x- and the z-form, and their coefficient matrices X and Z, respectively. With cubic Hermite, the x-form requires 6 multiplications to calculate the polynomial $g(x)$ coefficients, and the z-form requires 7. So, for cubic Hermite, it is better to use the x-form given earlier.

After the coefficients have been computed, the interpolation polynomial $g(x)$ must be evaluated. Horner's rule can be used to reduce multiplications, for example:

$$c_3x^3+c_2x^2+c_1x+c_0 = ((c_3x+c_2)x+c_1)x+c_0$$

Formula: Horner's rule

So evaluating a polynomial requires the same number of multiplications its order is. Summing this number with the number of multiplications required by the cheaper matrix-vector multiplication (x-form for cubic Hermite) gives the total number of multiplications required (9 for cubic Hermite). It should be noted, however, that on some platforms, using Horner's rule is not beneficial, so feel free to modify the source codes given later.

Next we shall try finding the best implementation form for each interpolator.

To choose between the x- and the z-form, we find out the number of operations in both. First we find out which needs more multiplications. If multiplications are expensive on the target platform, then that is what matters. However, if multiplications are as cheap as addition and subtraction operations, then the total number of operations counts. Optimized C language source code for the x- and z-form algorithms is given in the following, and the total number of operations is counted from each.

Looking at the matrices X and Z, some of the coefficients can be paired. This is marked with blue color, and two such coefficients require only one multiplication. Usually, the pair can be found from the same row, but sometimes on the same column. If more coefficients can be combined into larger groups that reduce multiplications, they are marked with red color. The Z-matrices have all rows either symmetrical or antisymmetrical, so pairing is guaranteed.

Zeros and plus and minus ones require no multiplications. If a coefficient is otherwise trivial, for example $\frac{1}{8}$, the multiplication might be possible to perform with a binary shift operation. In this paper, however, a floating point platform with no such scaling command is presumed and coefficients like that are treated as usual.

6.1 Linear

Linear interpolation is best done simply as:

$$g(x) = y[0]+x(y[1]-y[0])$$

Formula: Linear interpolation implementation

requiring total 1 multiplication.

```
// Linear
return y[0] + x*(y[1]-y[0]);
```

Routine: Linear interpolation implementation

The total number of operations in the routine is 3 (1 mul, 2 adds/subs).

6.2 B-spline

$$X = \begin{pmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \end{pmatrix}, \; Z = \begin{pmatrix} \frac{1}{48} & \frac{23}{48} & \frac{23}{48} & \frac{1}{48} \\ -\frac{1}{8} & -\frac{5}{8} & \frac{5}{8} & \frac{1}{8} \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \end{pmatrix}$$

Formula: 4-point, 3rd-order B-spline x- and z-form coefficient matrices

For 4-point, 3rd-order B-spline, X requires 6 multiplications and Z requires 7. Note that the polynomial evaluation requires some more, but the number is typically same for both the x- and the z-form of an interpolator.

```
// 4-point, 3rd-order B-spline (x-form)
float ym1py1 = y[-1]+y[1];
float c0 = 1/6.0*ym1py1 + 2/3.0*y[0];
float c1 = 1/2.0*(y[1]-y[-1]);
float c2 = 1/2.0*ym1py1 - y[0];
float c3 = 1/2.0*(y[0]-y[1]) + 1/6.0*(y[2]-y[-1]);
return ((c3*x+c2)*x+c1)*x+c0;
```

Routine: 4-point, 3rd-order B-spline x-form implementation

```
// 4-point, 3rd-order B-spline (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], modd1 = y[2]-y[-1];
float even2 = y[0]+y[1],  modd2 = y[1]-y[0];
float c0 = 1/48.0*even1 + 23/48.0*even2;
float c1 = 1/8.0*modd1 + 5/8.0*modd2;
float c2 = 1/4.0*(even1-even2);
float c3 = 1/6.0*modd1 - 1/2.0*modd2;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order B-spline z-form implementation

For 4-point, 3rd-order B-spline, the total number of operations in the x-form routine is 19 (9 muls, 10 adds/subs) and in the z-form 22 (10 muls, 12 adds/subs), so the x-form is favorable.

$$X = \begin{pmatrix} \frac{1}{120} & \frac{13}{60} & \frac{11}{20} & \frac{13}{60} & \frac{1}{120} & 0 \\ -\frac{1}{24} & -\frac{5}{12} & 0 & \frac{5}{12} & \frac{1}{24} & 0 \\ \frac{1}{12} & \frac{1}{6} & -\frac{1}{2} & \frac{1}{6} & \frac{1}{12} & 0 \\ -\frac{1}{12} & \frac{1}{6} & 0 & -\frac{1}{6} & \frac{1}{12} & 0 \\ \frac{1}{24} & -\frac{1}{6} & \frac{1}{4} & -\frac{1}{6} & \frac{1}{24} & 0 \\ -\frac{1}{120} & \frac{1}{24} & -\frac{1}{12} & \frac{1}{12} & -\frac{1}{24} & \frac{1}{120} \end{pmatrix}$$

$$Z = \begin{pmatrix} \frac{1}{3840} & \frac{79}{1280} & \frac{841}{1920} & \frac{841}{1920} & \frac{79}{1280} & \frac{1}{3840} \\ -\frac{1}{384} & -\frac{25}{128} & -\frac{77}{192} & \frac{77}{192} & \frac{25}{128} & \frac{1}{384} \\ \frac{1}{96} & \frac{7}{32} & -\frac{11}{48} & -\frac{11}{48} & \frac{7}{32} & \frac{1}{96} \\ -\frac{1}{48} & -\frac{1}{16} & \frac{7}{24} & -\frac{7}{24} & \frac{1}{16} & \frac{1}{48} \\ \frac{1}{48} & -\frac{1}{16} & \frac{1}{24} & \frac{1}{24} & -\frac{1}{16} & \frac{1}{48} \\ -\frac{1}{120} & \frac{1}{24} & -\frac{1}{12} & \frac{1}{12} & -\frac{1}{24} & \frac{1}{120} \end{pmatrix}$$

Formula: 6-point, 5th-order B-spline x- and z-form coefficient matrices

For 6-point, 5th-order B-spline, X requires 15 multiplications against the 18 of Z, so the x-form is probably more favorable.

```
// 6-point, 5th-order B-spline (x-form)
float ym2py2 = y[-2]+y[2], ym1py1 = y[-1]+y[1];
float y2mym2 = y[2]-y[-2], y1mym1 = y[1]-y[-1];
float sixthym1py1 = 1/6.0*ym1py1;
float c0 = 1/120.0*ym2py2 + 13/60.0*ym1py1 + 11/20.0*y[0];
float c1 = 1/24.0*y2mym2 + 5/12.0*y1mym1;
float c2 = 1/12.0*ym2py2 + sixthym1py1 - 1/2.0*y[0];
float c3 = 1/12.0*y2mym2 - 1/6.0*y1mym1;
float c4 = 1/24.0*ym2py2 - sixthym1py1 + 1/4.0*y[0];
float c5 = 1/120.0*(y[3]-y[-2]) + 1/24.0*(y[-1]-y[2]) +
    1/12.0*(y[1]-y[0]);
return ((((c5*x+c4)*x+c3)*x+c2)*x+c1)*x+c0;
```

Routine: 6-point, 5th-order B-spline x-form implementation

For 6-point, 5th-order B-spline, the total number of operations in the x-form routine is 42 (20 mul, 22 add/sub). There is a rule that a z-form routine with only pairing, no larger grouping, requires $order \times (points+1)+2 \times points$ operations total. In this case, the z-form would require 47 operations, so the x-form is a clear winner.

6.3 Lagrange

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \end{pmatrix}, \; Z = \begin{pmatrix} -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\ \frac{1}{24} & -\frac{9}{8} & \frac{9}{8} & -\frac{1}{24} \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \end{pmatrix}$$

Formula: 4-point, 3rd-order Lagrange x- and z-form coefficient matrices

For 4-point, 3rd-order Lagrange, X requires 6 multiplications and Z requires 7.

```
// 4-point, 3rd-order Lagrange (x-form)
float c0 = y[0];
float c1 = y[1] - 1/3.0*y[-1] - 1/2.0*y[0] - 1/6.0*y[2];
float c2 = 1/2.0*(y[-1]+y[1]) - y[0];
float c3 = 1/6.0*(y[2]-y[-1]) + 1/2.0*(y[0]-y[1]);
return ((c3*x+c2)*x+c1)*x+c0;
```

Routine: 4-point, 3rd-order Lagrange x-form implementation

```
// 4-point, 3rd-order Lagrange (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], odd1 = y[-1]-y[2];
float even2 = y[0]+y[1],  odd2 = y[0]-y[1];
float c0 = 9/16.0*even2 - 1/16.0*even1;
float c1 = 1/24.0*odd1 - 9/8.0*odd2;
float c2 = 1/4.0*(even1-even2);
float c3 = 1/2.0*odd2 - 1/6.0*odd1;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order Lagrange z-form implementation

For 4-point, 3rd-order Lagrange, the total number of operations in the x-form routine is 20 (9 muls, 11 adds/subs) and in the z-form 22 (10 muls, 12adds/subs), so the x-form is more favorable.

$$X = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{20} & -\frac{1}{2} & -\frac{1}{3} & 1 & -\frac{1}{4} & \frac{1}{30} \\ -\frac{1}{24} & \frac{2}{3} & -\frac{5}{4} & \frac{2}{3} & -\frac{1}{24} & 0 \\ -\frac{1}{24} & -\frac{1}{24} & \frac{5}{12} & -\frac{7}{12} & \frac{7}{24} & -\frac{1}{24} \\ \frac{1}{24} & -\frac{1}{6} & \frac{1}{4} & -\frac{1}{6} & \frac{1}{24} & 0 \\ -\frac{1}{120} & \frac{1}{24} & -\frac{1}{12} & \frac{1}{12} & -\frac{1}{24} & \frac{1}{120} \end{pmatrix}$$

$$Z = \begin{pmatrix} \frac{3}{256} & -\frac{25}{256} & \frac{75}{128} & \frac{75}{128} & -\frac{25}{256} & \frac{3}{256} \\ -\frac{3}{640} & \frac{25}{384} & -\frac{75}{64} & \frac{75}{64} & -\frac{25}{384} & \frac{3}{640} \\ -\frac{5}{96} & \frac{13}{32} & -\frac{17}{48} & -\frac{17}{48} & \frac{13}{32} & -\frac{5}{96} \\ \frac{1}{48} & -\frac{13}{48} & \frac{17}{24} & -\frac{17}{24} & \frac{13}{48} & -\frac{1}{48} \\ \frac{1}{48} & -\frac{1}{16} & \frac{1}{24} & \frac{1}{24} & -\frac{1}{16} & \frac{1}{48} \\ -\frac{1}{120} & \frac{1}{24} & -\frac{1}{12} & \frac{1}{12} & -\frac{1}{24} & \frac{1}{120} \end{pmatrix}$$

Formula: 6-point, 5th-order Lagrange x- and z-form coefficient matrices

For 6-point, 5th-order Lagrange, X requires 17 multiplications and Z requires 18.

```
// 6-point, 5th-order Lagrange (x-form)
float ym1py1 = y[-1]+y[1];
float twentyfourthym2py2 = 1/24.0*(y[-2]+y[2]);
float c0 = y[0];
float c1 = 1/20.0*y[-2] - 1/2.0*y[-1] - 1/3.0*y[0] + y[1] -
    1/4.0*y[2] + 1/30.0*y[3];
float c2 = 2/3.0*ym1py1 - 5/4.0*y[0] - twentyfourthym2py2;
float c3 = 5/12.0*y[0] - 7/12.0*y[1] + 7/24.0*y[2] -
    1/24.0*(y[-2]+y[-1]+y[3]);
float c4 = 1/4.0*y[0] - 1/6.0*ym1py1 + twentyfourthym2py2;
float c5 = 1/120.0*(y[3]-y[-2]) + 1/24.0*(y[-1]-y[2]) +
    1/12.0*(y[1]-y[0]);
return ((((c5*x+c4)*x+c3)*x+c2)*x+c1)*x+c0;
```

Routine: 6-point, 5th-order Lagrange x-form implementation

```
// 6-point, 5th-order Lagrange (z-form)
float z = x - 1/2.0;
float even1 = y[-2]+y[3], odd1 = y[-2]-y[3];
float even2 = y[-1]+y[2], odd2 = y[-1]-y[2];
float even3 = y[0]+y[1],  odd3 = y[0]-y[1];
float c0 = 3/256.0*even1 - 25/256.0*even2 + 75/128.0*even3;
float c1 = 25/384.0*odd2 - 75/64.0*odd3 - 3/640.0*odd1;
float c2 = 13/32.0*even2 - 17/48.0*even3 - 5/96.0*even1;
float c3 = 1/48.0*odd1 - 13/48.0*odd2 + 17/24.0*odd3;
float c4 = 1/48.0*even1 - 1/16.0*even2 + 1/24.0*even3;
float c5 = 1/24.0*odd2 - 1/12.0*odd3 - 1/120.0*odd1;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order Lagrange z-form implementation

For 6-point, 5th-order Lagrange, the total number of operations in the x-form routine is 48 (22 muls, 26 adds/subs), and in the z-form routine 47 (23 muls, 24 adds/subs). Depending on the platform, either is faster.

6.4 Hermite

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 1 & -\frac{5}{2} & 2 & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \end{pmatrix}, \; Z = \begin{pmatrix} -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\ \frac{1}{8} & -\frac{11}{8} & \frac{11}{8} & -\frac{1}{8} \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \end{pmatrix}$$

Formula: 4-point, 3rd-order Hermite x- and z-form coefficient matrices

For 4-point, 3rd-order Hermite, X requires 6 multiplications and Z requires 7.

```
// 4-point, 3rd-order Hermite (x-form)
float c0 = y[0];
float c1 = 1/2.0*(y[1]-y[-1]);
float c2 = y[-1] - 5/2.0*y[0] + 2*y[1] - 1/2.0*y[2];
float c3 = 1/2.0*(y[2]-y[-1]) + 3/2.0*(y[0]-y[1]);
return ((c3*x+c2)*x+c1)*x+c0;
```

Routine: 4-point, 3rd-order Hermite x-form implementation

```
// 4-point, 3rd-order Hermite (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], odd1 = y[-1]-y[2];
float even2 = y[0]+y[1],  odd2 = y[0]-y[1];
float c0 = 9/16.0*even2 - 1/16.0*even1;
float c1 = 1/8.0*odd1 - 11/8.0*odd2;
float c2 = 1/4.0*(even1-even2);
float c3 = 3/2.0*odd2 - 1/2.0*odd1;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order Hermite z-form implementation

For 4-point, 3rd-order Hermite, the total number of operations in the x-form routine is 19 (9 muls, 10 adds/subs), and in the z-form routine 22 (10 muls, 12 adds/subs). So the x-form routine is favorable.

$$X = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} & 0 \\ -\frac{1}{6} & \frac{5}{4} & -\frac{7}{3} & \frac{5}{3} & -\frac{1}{2} & \frac{1}{12} \\ \frac{1}{12} & -\frac{7}{12} & \frac{4}{3} & -\frac{4}{3} & \frac{7}{12} & -\frac{1}{12} \end{pmatrix}$$

$$Z = \begin{pmatrix} \frac{1}{96} & -\frac{3}{32} & \frac{7}{12} & \frac{7}{12} & -\frac{3}{32} & \frac{1}{96} \\ -\frac{1}{48} & \frac{7}{48} & -\frac{4}{3} & \frac{4}{3} & -\frac{7}{48} & \frac{1}{48} \\ -\frac{1}{24} & \frac{3}{8} & -\frac{1}{3} & -\frac{1}{3} & \frac{3}{8} & -\frac{1}{24} \\ \frac{1}{12} & -\frac{7}{12} & \frac{4}{3} & -\frac{4}{3} & \frac{7}{12} & -\frac{1}{12} \end{pmatrix}$$

Formula: 6-point, 3rd-order Hermite x- and z-form coefficient matrices

For 6-point, 3rd-order Hermite, X requires 11 multiplications and Z requires 11.

```
// 6-point, 3rd-order Hermite (x-form)
float c0 = y[0];
float c1 = 1/12.0*(y[-2]-y[2]) + 2/3.0*(y[1]-y[-1]);
float c2 = 5/4.0*y[-1] - 7/3.0*y[0] + 5/3.0*y[1] -
    1/2.0*y[2] + 1/12.0*y[3] - 1/6.0*y[-2];
float c3 = 1/12.0*(y[-2]-y[3]) + 7/12.0*(y[2]-y[-1]) +
    4/3.0*(y[0]-y[1]);
return ((c3*x+c2)*x+c1)*x+c0;
```

Routine: 6-point, 3rd-order Hermite x-form implementation

```
// 6-point, 3rd-order Hermite (z-form)
float z = x - 1/2.0;
float even1 = y[-2]+y[3], odd1 = y[-2]-y[3];
float even2 = y[-1]+y[2], odd2 = y[-1]-y[2];
float even3 = y[0]+y[1], fourthirdthodd3 = 4/3.0*(y[0]-y[1]);
float c0 = 1/96.0*even1 - 3/32.0*even2 + 7/12.0*even3;
float c1 = 7/48.0*odd2 - fourthirdthodd3 - 1/48.0*odd1;
float c2 = 3/8.0*even2 - 1/3.0*even3 - 1/24.0*even1;
float c3 = 1/12.0*odd1 - 7/12.0*odd2 + fourthirdthodd3;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 3rd-order Hermite z-form implementation

For 6-point, 3rd-order Hermite, the total number of operations in the x-form routine is 30 (14 muls, 16 adds/subs), and in the z-form routine 32 (14 muls, 18 adds/subs). So the x-form routine is more favorable.

$$X = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} & 0 \\ -\frac{1}{8} & \frac{13}{12} & -\frac{25}{12} & \frac{3}{2} & -\frac{11}{24} & \frac{1}{12} \\ -\frac{1}{24} & -\frac{1}{24} & \frac{5}{12} & -\frac{7}{12} & \frac{7}{24} & -\frac{1}{24} \\ \frac{1}{8} & -\frac{7}{12} & \frac{13}{12} & -1 & \frac{11}{24} & -\frac{1}{12} \\ -\frac{1}{24} & \frac{5}{24} & -\frac{5}{12} & \frac{5}{12} & -\frac{5}{24} & \frac{1}{24} \end{pmatrix}$$

$$Z = \begin{pmatrix} \frac{3}{256} & -\frac{25}{256} & \frac{75}{128} & \frac{75}{128} & -\frac{25}{256} & \frac{3}{256} \\ -\frac{3}{128} & \frac{61}{384} & -\frac{87}{64} & \frac{87}{64} & -\frac{61}{384} & \frac{3}{128} \\ -\frac{5}{96} & \frac{13}{32} & -\frac{17}{48} & -\frac{17}{48} & \frac{13}{32} & -\frac{5}{96} \\ \frac{5}{48} & -\frac{11}{16} & \frac{37}{24} & -\frac{37}{24} & \frac{11}{16} & -\frac{5}{48} \\ \frac{1}{48} & -\frac{1}{16} & \frac{1}{24} & \frac{1}{24} & -\frac{1}{16} & \frac{1}{48} \\ -\frac{1}{24} & \frac{5}{24} & -\frac{5}{12} & \frac{5}{12} & -\frac{5}{24} & \frac{1}{24} \end{pmatrix}$$

Formula: 6-point, 5th-order Hermite x- and z-form coefficient matrices

For 6-point, 5th-order Hermite, X requires 17 multiplications against the 18 of Z.

```
// 6-point, 5th-order Hermite (x-form)
float eighthym2 = 1/8.0*y[-2];
float eleventwentyfourthy2 = 11/24.0*y[2];
float twelfthy3 = 1/12.0*y[3];
float c0 = y[0];
float c1 = 1/12.0*(y[-2]-y[2]) + 2/3.0*(y[1]-y[-1]);
float c2 = 13/12.0*y[-1] - 25/12.0*y[0] + 3/2.0*y[1] -
    eleventwentyfourthy2 + twelfthy3 - eighthym2;
float c3 = 5/12.0*y[0] - 7/12.0*y[1] + 7/24.0*y[2] -
    1/24.0*(y[-2]+y[-1]+y[3]);
float c4 = eighthym2 - 7/12.0*y[-1] + 13/12.0*y[0] - y[1] +
    eleventwentyfourthy2 - twelfthy3;
float c5 = 1/24.0*(y[3]-y[-2]) + 5/24.0*(y[-1]-y[2]) +
    5/12.0*(y[1]-y[0]);
return ((((c5*x+c4)*x+c3)*x+c2)*x+c1)*x+c0;
```

Routine: 6-point, 5th-order Hermite x-form implementation

```
// 6-point, 5th-order Hermite (z-form)
float z = x - 1/2.0;
float even1 = y[-2]+y[3], odd1 = y[-2]-y[3];
float even2 = y[-1]+y[2], odd2 = y[-1]-y[2];
float even3 = y[0]+y[1],  odd3 = y[0]-y[1];
float c0 = 3/256.0*even1 - 25/256.0*even2 + 75/128.0*even3;
float c1 = -3/128.0*odd1 + 61/384.0*odd2 - 87/64.0*odd3;
float c2 = -5/96.0*even1 + 13/32.0*even2 - 17/48.0*even3;
float c3 = 5/48.0*odd1 - 11/16.0*odd2 + 37/24.0*odd3;
float c4 = 1/48.0*even1 - 1/16.0*even2 + 1/24.0*even3;
float c5 = -1/24.0*odd1 + 5/24.0*odd2 - 5/12.0*odd3;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order Hermite z-form implementation

For 6-point, 5th-order Hermite, the total number of operations in the x-form routine is 50 (22 muls, 28 adds/subs), and in the z-form routine 47 (23 muls, 24 adds/subs). Depending on the platform, either is faster.

6.5 2nd-order-osculating

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ \frac{3}{2} & -\frac{9}{2} & \frac{9}{2} & -\frac{3}{2} \\ -\frac{5}{2} & \frac{15}{2} & -\frac{15}{2} & \frac{5}{2} \\ 1 & -3 & 3 & -1 \end{pmatrix}, \; Z = \begin{pmatrix} -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\ \frac{3}{16} & -\frac{25}{16} & \frac{25}{16} & -\frac{3}{16} \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ -1 & 3 & -3 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & -3 & 3 & -1 \end{pmatrix}$$

Formula: 4-point, 5th-order 2nd-order-osculating x- and z-form coefficient matrices

For 4-point, 5th-order 2nd-order-osculating, X requires 7 multiplications and Z requires 6.

```
// 4-point, 5th-order 2nd-order-osculating (x-form)
float y1my0 = y[1]-y[0];
float y2mym1 = y[2]-y[-1];
float c0 = y[0];
float c1 = 1/2.0*(y[1]-y[-1]);
float c2 = 1/2.0*(y[-1]+y[1]) - y[0];
float c3 = 9/2.0*y1my0 - 3/2.0*y2mym1;
float c4 = 5/2.0*y2mym1 - 15/2.0*y1my0;
float c5 = y[-1] + 3*y1my0 - y[2];
return ((((c5*x+c4)*x+c3)*x+c2)*x+c1)*x+c0;
```

Routine: 4-point, 5th-order 2nd-order-osculating x-form implementation

```
// 4-point, 5th-order 2nd-order-osculating (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], odd1 = y[-1]-y[2];
float even2 = y[0]+y[1],  odd2 = y[0]-y[1];
float c0 = 9/16.0*even2 - 1/16.0*even1;
float c1 = 3/16.0*odd1 - 25/16.0*odd2;
float c2 = 1/4.0*(even1-even2);
float c5 = odd1 - 3*odd2;
return (((c5*z*z-c5)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 5th-order 2nd-order-osculating z-form implementation

For 4-point, 5th-order 2nd-order-osculating, the total number of operations in the x-form routine is 26 (12 muls, 14 adds/subs), and in the z-form routine 24 (11 muls, 13 adds/subs). This makes the z-form more favorable.

$$X = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} & 0 \\ -\frac{1}{24} & \frac{2}{3} & -\frac{5}{4} & \frac{2}{3} & -\frac{1}{24} & 0 \\ -\frac{3}{8} & \frac{13}{8} & -\frac{35}{12} & \frac{11}{4} & -\frac{11}{8} & \frac{7}{24} \\ \frac{13}{24} & -\frac{8}{3} & \frac{21}{4} & -\frac{31}{6} & \frac{61}{24} & -\frac{1}{2} \\ -\frac{5}{24} & \frac{25}{24} & -\frac{25}{12} & \frac{25}{12} & -\frac{25}{24} & \frac{5}{24} \end{pmatrix}$$

$$Z = \begin{pmatrix} \frac{3}{256} & -\frac{25}{256} & \frac{75}{128} & \frac{75}{128} & -\frac{25}{256} & \frac{3}{256} \\ -\frac{13}{384} & \frac{27}{128} & -\frac{281}{192} & \frac{281}{192} & -\frac{27}{128} & \frac{13}{384} \\ -\frac{5}{96} & \frac{13}{32} & -\frac{17}{48} & -\frac{17}{48} & \frac{13}{32} & -\frac{5}{96} \\ \frac{3}{16} & -\frac{53}{48} & \frac{19}{8} & -\frac{19}{8} & \frac{53}{48} & -\frac{3}{16} \\ \frac{1}{48} & -\frac{1}{16} & \frac{1}{24} & \frac{1}{24} & -\frac{1}{16} & \frac{1}{48} \\ -\frac{5}{24} & \frac{25}{24} & -\frac{25}{12} & \frac{25}{12} & -\frac{25}{24} & \frac{5}{24} \end{pmatrix}$$

Formula: 6-point, 5th-order 2nd-order-osculating x- and z-form coefficient matrices

For 6-point, 5th-order 2nd-order-osculating, X requires 20 multiplications and Z requires 18.

```
// 6-point, 5th-order 2nd-order-osculating (x-form)
float c0 = y[0];
float c1 = 1/12.0*(y[-2]-y[2]) + 2/3.0*(y[1]-y[-1]);
float c2 = 2/3.0*(y[-1]+y[1]) - 1/24.0*(y[-2]+y[2]) - 5/4.0*y[0];
float c3 = 13/8.0*y[-1] - 35/12.0*y[0] + 11/4.0*y[1] -
    11/8.0*y[2] + 7/24.0*y[3] - 3/8.0*y[-2];
float c4 = 13/24.0*y[-2] - 8/3.0*y[-1] + 21/4.0*y[0] -
    31/6.0*y[1] + 61/24.0*y[2] - 1/2.0*y[3];
float c5 = 5/24.0*(y[3]-y[-2]) + 25/24.0*(y[-1]-y[2]) +
    25/12.0*(y[1]-y[0]);
return ((((c5*x+c4)*x+c3)*x+c2)*x+c1)*x+c0;
```

Routine: 6-point, 5th-order 2nd-order-osculating x-form implementation

```
// 6-point, 5th-order 2nd-order-osculating (z-form)
float z = x - 1/2.0;
float even1 = y[-2]+y[3], odd1 = y[-2]-y[3];
float even2 = y[-1]+y[2], odd2 = y[-1]-y[2];
float even3 = y[0]+y[1],  odd3 = y[0]-y[1];
float c0 = 3/256.0*even1 - 25/256.0*even2 + 75/128.0*even3;
float c1 = 27/128.0*odd2 - 281/192.0*odd3 - 13/384.0*odd1;
float c2 = 13/32.0*even2 - 17/48.0*even3 - 5/96.0*even1;
float c3 = 3/16.0*odd1 - 53/48.0*odd2 + 19/8.0*odd3;
float c4 = 1/48.0*even1 - 1/16.0*even2 + 1/24.0*even3;
float c5 = 25/24.0*odd2 - 25/12.0*odd3 - 5/24.0*odd1;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order 2nd-order-osculating z-form implementation

For 6-point, 5th-order 2nd-order-osculating, the total number of operations in the x-form routine is 52 (25 muls, 27 adds/subs), and in the z-form routine 47 (23 muls, 24 adds/subs). This makes the z-form more favorable.

6.6 Watte tri-linear

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}, \; Z = \begin{pmatrix} -\frac{1}{8} & \frac{5}{8} & \frac{5}{8} & -\frac{1}{8} \\ 0 & -1 & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Formula: 4-point, 2nd-order Watte tri-linear x- and z-form coefficient matrices

For 4-point, 2nd-order Watte tri-linear, X requires 3 multiplications and Z requires 3.

```
// 4-point, 2nd-order Watte tri-linear (x-form)
float ym1py2 = y[-1]+y[2];
float c0 = y[0];
float c1 = 3/2.0*y[1] - 1/2.0*(y[0]+ym1py2);
float c2 = 1/2.0*(ym1py2-y[0]-y[1]);
return (c2*x+c1)*x+c0;
```

Routine: 4-point, 2nd-order Watte tri-linear x-form implementation

```
// 4-point, 2nd-order Watte tri-linear (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], even2 = y[0]+y[1];
float c0 = 5/8.0*even2 - 1/8.0*even1;
float c1 = y[1]-y[0];
float c2 = 1/2.0*(even1-even2);
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order Watte tri-linear z-form implementation

For 4-point, 2nd-order Watte tri-linear, the total number of operations in the x-form routine is 12 (5 muls, 7 adds/subs), and in the z-form routine 13 (5 muls, 8 adds/subs). This makes the x-form more favorable.

6.7 Parabolic 2x

$$X = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \end{pmatrix}, \; Z = \begin{pmatrix} \frac{1}{16} & \frac{7}{16} & \frac{7}{16} & \frac{1}{16} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

Formula: 4-point, 2nd-order parabolic 2x x- and z-form coefficient matrices

For 4-point, 2nd-order parabolic 2x, X requires 4 multiplications and Z requires 4.

```
// 4-point, 2nd-order parabolic 2x (x-form)
float y1mym1 = y[1]-y[-1];
float c0 = 1/2.0*y[0] + 1/4.0*(y[-1]+y[1]);
float c1 = 1/2.0*y1mym1;
float c2 = 1/4.0*(y[2]-y[0]-y1mym1);
return (c2*x+c1)*x+c0;
```

Routine: 4-point, 2nd-order parabolic 2x x-form implementation

```
// 4-point, 2nd-order parabolic 2x (z-form)
float z = x - 1/2.0;
float even1 = y[-1]+y[2], even2 = y[0]+y[1];
float c0 = 1/16.0*even1 + 7/16.0*even2;
float c1 = 1/4.0*(y[1]-y[0]+y[2]-y[-1]);
float c2 = 1/4.0*(even1-even2);
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order parabolic 2x z-form implementation

For 4-point, 2nd-order parabolic 2x, the total number of operations in the x-form routine is 13 (6 muls, 7 adds/subs), and in the z-form routine 16 (6 muls, 10 adds/subs). This makes the x-form more favorable.

6.8 2-point, 3rd-order optimal

Rather than making it a big deal, we presume that the z-form is the most favorable for the optimal interpolators. This is probably true, because the simplicity of the X matrices with the traditional interpolators has to do with that they were symbolically constructed. With the optimal interpolators, we are dealing with virtually unconstrained floating point constants, so it is very unlikely that any pairing could be done or other shortcuts could be made in the x-form. It seems that with some rounding of the coefficients, it is possible to do additional grouping in the Z matrix for some of these interpolators. Next, only the z-form implementations for the optimal interpolators are given.

```
// Optimal 2x (2-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float c0 = even1*0.50037842517188658;
float c1 = odd1*1.00621089801788210;
float c2 = even1*-0.004541102062639801;
float c3 = odd1*-1.57015627178718420;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 2-point, 3rd-order optimal 2x z-form implementation

```
// Optimal 4x (2-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float c0 = even1*0.50013034073688023;
float c1 = odd1*1.09617817497678520;
float c2 = even1*-0.001564088842561871;
float c3 = odd1*-1.32598918957298410;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 2-point, 3rd-order optimal 4x z-form implementation

```
// Optimal 8x (2-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float c0 = even1*0.50004007194083089;
float c1 = odd1*1.06397659072500650;
float c2 = even1*-0.000480863289971321;
float c3 = odd1*-0.73514591836770027;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 2-point, 3rd-order optimal 8x z-form implementation

```
// Optimal 16x (2-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float c0 = even1*0.50001096675880796;
float c1 = odd1*1.03585606328743830;
float c2 = even1*-0.000131601105693441;
float c3 = odd1*-0.38606621963374965;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 2-point, 3rd-order optimal 16x z-form implementation

```
// Optimal 32x (2-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float c0 = even1*0.50000286037713559;
float c1 = odd1*1.01889120864375270;
float c2 = even1*-0.000034324525627571;
float c3 = odd1*-0.19775766248673177;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 2-point, 3rd-order optimal 32x z-form implementation

The 2-point, 3rd-order optimal interpolation z-form implementations require total 13 (7 muls, 6 adds/subs) operations each.

6.9 4-point, 2nd-order optimal

```
// Optimal 2x (4-point, 2nd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.42334633257225274 + even2*0.07668732202139628;
float c1 = odd1*0.26126047291143606 + odd2*0.24778879018226652;
float c2 = even1*-0.213439787561776841 + even2*0.21303593243799016;
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order optimal 2x z-form implementation

```
// Optimal 4x (4-point, 2nd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.38676264891201206 + even2*0.11324319172521946;
float c1 = odd1*0.01720901456660906 + odd2*0.32839294317251788;
float c2 = even1*-0.228653995318581881 + even2*0.22858390767180370;
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order optimal 4x z-form implementation

```
// Optimal 8x (4-point, 2nd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.32852206663814043 + even2*0.17147870380790242;
float c1 = odd1*-0.35252373075274990 + odd2*0.45113687946292658;
float c2 = even1*-0.240052062078895181 + even2*0.24004281672637814;
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order optimal 8x z-form implementation

```
// Optimal 16x (4-point, 2nd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.20204741371575463 + even2*0.29795268253813623;
float c1 = odd1*-1.11855475338366150 + odd2*0.70626377291054832;
float c2 = even1*-0.245061178654743641 + even2*0.24506002360805534;
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order optimal 16x z-form implementation

```
// Optimal 32x (4-point, 2nd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*-0.04817865217726123 + even2*0.54817866412548932;
float c1 = odd1*-2.62328241292796620 + odd2*1.20778105913587620;
float c2 = even1*-0.247552438397138281 + even2*0.24755229501840223;
return (c2*z+c1)*z+c0;
```

Routine: 4-point, 2nd-order optimal 32x z-form implementation

The 4-point, 2nd-order optimal interpolation z-form implementations require total 18 (8 muls, 10 adds/subs) operations each.

6.10 4-point, 3rd-order optimal

```
// Optimal 2x (4-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.45868970870461956 + even2*0.04131401926395584;
float c1 = odd1*0.48068024766578432 + odd2*0.17577925564495955;
float c2 = even1*-0.246185007019907091 + even2*0.24614027139700284;
float c3 = odd1*-0.36030925263849456 + odd2*0.10174985775982505;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order optimal 2x z-form implementation

```
// Optimal 4x (4-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46209345013918979 + even2*0.03790693583186333;
float c1 = odd1*0.51344507801315964 + odd2*0.16261507145522014;
float c2 = even1*-0.24854033299029421 1 + even2*0.24853570133765701;
float c3 = odd1*-0.42912649274763925 + odd2*0.13963062613760227;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order optimal 4x z-form implementation

```
// Optimal 8x (4-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46360002085841184 + even2*0.03640000638072349;
float c1 = odd1*0.52776949859997280 + odd2*0.15746108253367153;
float c2 = even1*-0.24965812153579325 1 + even2*0.24965779466617388;
float c3 = odd1*-0.46789242171187317 + odd2*0.15551896027602030;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order optimal 8x z-form implementation

```
// Optimal 16x (4-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46436507349411416 + even2*0.03563492826010761;
float c1 = odd1*0.53463126553787166 + odd2*0.15512856361039451;
float c2 = even1*-0.24992354096715974 1 + even2*0.24992351991649797;
float c3 = odd1*-0.48601256046234864 + odd2*0.16195131297091253;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order optimal 16x z-form implementation

```
// Optimal 32x (4-point, 3rd-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46465589031535864 + even2*0.03534410979496938;
float c1 = odd1*0.53726845877054186 + odd2*0.15424449410914165;
float c2 = even1*-0.24998193095402910 1 + even2*0.24998192963009191;
float c3 = odd1*-0.49369595780454456 + odd2*0.16455902278580614;
return ((c3*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 3rd-order optimal 32x z-form implementation

The 4-point, 3rd-order optimal interpolation z-form implementations require total 23 (11 muls, 12 adds/subs) operations each.

6.11 4-point, 4th-order optimal

```
// Optimal 2x (4-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.45645918406487612 + even2*0.04354173901996461;
float c1 = odd1*0.47236675362442071 + odd2*0.17686613581136501;
float c2 = even1*-0.25367479420455852 1 + even2*0.25371918651882464;
float c3 = odd1*-0.37917091811631082 + odd2*0.11952965967158000;
float c4 = even1*0.04252164479749607 + even2*-0.04289144034653719;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 4th-order optimal 2x z-form implementation

```
// Optimal 4x (4-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46567255120778489 + even2*0.03432729708429672;
float c1 = odd1*0.53743830753560162 + odd2*0.15429462557307461;
float c2 = even1*-0.25194210134021744 1 + even2*0.25194744935939062;
float c3 = odd1*-0.46896069955075126 + odd2*0.15578800670302476;
float c4 = even1*0.00986988334359864 + even2*-0.00989340017126506;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 4th-order optimal 4x z-form implementation

```
// Optimal 8x (4-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46771532012068961 + even2*0.03228466824404497;
float c1 = odd1*0.55448654344364423 + odd2*0.14851181120641987;
float c2 = even1*-0.25058728369811012 1 + even2*0.25058765188457821;
float c3 = odd1*-0.49209020939096676 + odd2*0.16399414834151946;
float c4 = even1*0.00255074537015887 + even2*-0.00255226912537286;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 4th-order optimal 8x z-form implementation

```
// Optimal 16x (4-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46822774170144532 + even2*0.03177225758005808;
float c1 = odd1*0.55890365706150436 + odd2*0.14703258836343669;
float c2 = even1*-0.25015341189379603 1 + even2*0.25015343462990891;
float c3 = odd1*-0.49800710906733769 + odd2*0.16600005174304033;
float c4 = even1*0.00064264050033187 + even2*-0.00064273459469381;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 4th-order optimal 16x z-form implementation

```
// Optimal 32x (4-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float c0 = even1*0.46835497211269561 + even2*0.03164502784253309;
float c1 = odd1*0.56001293337091440 + odd2*0.14666238593949288;
float c2 = even1*-0.25003875982623369 1 + even2*0.25003876124297131;
float c3 = odd1*-0.49949850957839148 + odd2*0.16649935475113800;
float c4 = even1*0.00016095224137360 + even2*-0.00016095810460478;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 4-point, 4th-order optimal 32x z-form implementation

The 4-point, 4th-order optimal interpolation z-form implementations require total 28 (14 muls, 14 adds/subs) operations each.

6.12 6-point, 4th-order optimal

```
// Optimal 2x (6-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.37484203669443822 + even2*0.11970939637439368
    + even3*0.00544862268096358;
float c1 = odd1*0.19253897284651597 + odd2*0.22555179040018719
    + odd3*0.02621377625620669;
float c2 = even1*-0.15402600647565307 1 + even2*0.10546111301131367
    + even3*0.04856757454258609;
float c3 = odd1*-0.06523685579716083 + odd2*-0.04867197815057284
    + odd3*0.04200764942718964;
float c4 = even1*0.03134095684084392 + even2*-0.04385804833432710
    + even3*0.01249475765486819;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 4th-order optimal 2x z-form implementation

```
// Optimal 4x (6-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.26148143200222657 + even2*0.22484494681472966
    + even3*0.01367360612950508;
float c1 = odd1*-0.20245593827436142 + odd2*0.29354348112881601
    + odd3*0.06436924057941607;
float c2 = even1*-0.022982104451679701 + even2*-0.09068617668887535
    + even3*0.11366875749521399;
float c3 = odd1*0.36296419678970931 + odd2*-0.26421064520663945
    + odd3*0.08591542869416055;
float c4 = even1*0.02881527997393852 + even2*-0.04250898918476453
    + even3*0.01369173779618459;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 4th-order optimal 4x z-form implementation

```
// Optimal 8x (6-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.07571827673995030 + even2*0.39809419102537769
    + even3*0.02618753167558019;
float c1 = odd1*-0.87079480370960549 + odd2*0.41706012247048818
    + odd3*0.12392296259397995;
float c2 = even1*0.18688371835645290 1 + even2*-0.40535151498252686
    + even3*0.21846781431808182;
float c3 = odd1*1.09174419992174300 + odd2*-0.62917625718809478
    + odd3*0.15915674384870970;
float c4 = even1*0.03401038103941584 + even2*-0.05090907029392906
    + even3*0.01689861603514873;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 4th-order optimal 8x z-form implementation

```
// Optimal 16x (6-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*-0.30943127416213301 + even2*0.75611844407537543
    + even3*0.05331283006820442;
float c1 = odd1*-2.23586327978235700 + odd2*0.66020840412562265
    + odd3*0.25104761112921636;
float c2 = even1*0.625420761014402691 + even2*-1.06313460380183860
    + even3*0.43771384337431529;
float c3 = odd1*2.57088518304678090 + odd2*-1.36878543609177150
    + odd3*0.30709424868485174;
float c4 = even1*0.03755086455339280 + even2*-0.05631219122315393
    + even3*0.01876132424143207;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 4th-order optimal 16x z-form implementation

```
// Optimal 32x (6-point, 4th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*-1.05730227922290790 + even2*1.45069541587021430
    + even3*0.10660686335233649;
float c1 = odd1*-4.87455554035028720 + odd2*1.12509567592532630
    + odd3*0.49985370215839708;
float c2 = even1*1.47937043582311210 1 + even2*-2.34405608915933780
    + even3*0.86468565335070746;
float c3 = odd1*5.42677291742286180 + odd2*-2.79672428287565160
    + odd3*0.59267998874843331;
float c4 = even1*0.03957507923965987 + even2*-0.05936083498715066
    + even3*0.01978575568000696;
return (((c4*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 4th-order optimal 32x z-form implementation

The 6-point, 4th-order optimal interpolation z-form implementations require total 40 (19 muls, 21 adds/subs) operations each.

6.13 6-point, 5th-order optimal

```
// Optimal 2x (6-point, 5th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.40513396007145713 + even2*0.09251794438424393
    + even3*0.00234806603570670;
float c1 = odd1*0.28342806338906690 + odd2*0.21703277024054901
    + odd3*0.01309294748731515;
float c2 = even1*-0.19133768254035194 1 + even2*0.16187844487943592
    + even3*0.02946017143111912;
float c3 = odd1*-0.16471626190554542 + odd2*-0.00154547203542499
    + odd3*0.03399271444851909;
float c4 = even1*0.03845798729588149 + even2*-0.05712936104242644
    + even3*0.01866750929921070;
float c5 = odd1*0.04317950185225609 + odd2*-0.01802814255926417
    + odd3*0.00152170021558204;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order optimal 2x z-form implementation

```
// Optimal 4x (6-point, 5th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.41496902959240894 + even2*0.08343081932889224
    + even3*0.00160015038681571;
float c1 = odd1*0.31625515004859783 + odd2*0.21197848565176958
    + odd3*0.00956166668408054;
float c2 = even1*-0.20327189654887537 1 + even2*0.17989908432249280
    + even3*0.02337283412161328;
float c3 = odd1*-0.20209241069835732 + odd2*0.01760734419526000
    + odd3*0.02985927012435252;
float c4 = even1*0.04100948858761910 + even2*-0.06147760875085254
    + even3*0.02046802954581191;
float c5 = odd1*0.06607747864416924 + odd2*-0.03255079211953620
    + odd3*0.00628989632244913;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order optimal 4x z-form implementation

```
// Optimal 8x (6-point, 5th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.41660797292569773 + even2*0.08188468587188069
    + even3*0.00150734119050266;
float c1 = odd1*0.32232780822726981 + odd2*0.21076321997422021
    + odd3*0.00907649978070957;
float c2 = even1*-0.205219993961471501 + even2*0.18282942057327367
    + even3*0.02239057377093268;
float c3 = odd1*-0.21022298520246224 + odd2*0.02176417471349534
    + odd3*0.02898626924395209;
float c4 = even1*0.04149963966704384 + even2*-0.06224707096203808
    + even3*0.02074742969707599;
float c5 = odd1*0.07517133281176167 + odd2*-0.03751837438141215
    + odd3*0.00747588873055296;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order optimal 8x z-form implementation

```
// Optimal 16x (6-point, 5th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.41809989254549901 + even2*0.08049339946273310
    + even3*0.00140670799165932;
float c1 = odd1*0.32767596257424964 + odd2*0.20978189376640677
    + odd3*0.00859567104974701;
float c2 = even1*-0.206944618112960001 + even2*0.18541689550861262
    + even3*0.02152772260740132;
float c3 = odd1*-0.21686095413034051 + odd2*0.02509557922091643
    + odd3*0.02831484751363800;
float c4 = even1*0.04163046817137675 + even2*-0.06244556931623735
    + even3*0.02081510113314315;
float c5 = odd1*0.07990500783668089 + odd2*-0.03994519162531633
    + odd3*0.00798609327859495;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order optimal 16x z-form implementation

```
// Optimal 32x (6-point, 5th-order) (z-form)
float z = x - 1/2.0;
float even1 = y[1]+y[0], odd1 = y[1]-y[0];
float even2 = y[2]+y[-1], odd2 = y[2]-y[-1];
float even3 = y[3]+y[-2], odd3 = y[3]-y[-2];
float c0 = even1*0.42685983409379380 + even2*0.07238123511170030
    + even3*0.00075893079450573;
float c1 = odd1*0.35831772348893259 + odd2*0.20451644554758297
    + odd3*0.00562658797241955;
float c2 = even1*-0.217009177221292431 + even2*0.20051376594086157
    + even3*0.01649541128040211;
float c3 = odd1*-0.25112715343740988 + odd2*0.04223025992200458
    + odd3*0.02488727472995134;
float c4 = even1*0.04166946673533273 + even2*-0.06250420114356986
    + even3*0.02083473440841799;
float c5 = odd1*0.08349799235675044 + odd2*-0.04174912841630993
    + odd3*0.00834987866042734;
return ((((c5*z+c4)*z+c3)*z+c2)*z+c1)*z+c0;
```

Routine: 6-point, 5th-order optimal 32x z-form implementation

The 6-point, 5th-order optimal interpolation z-form implementations require total 47 (23 muls, 24 adds/subs) operations each.

7. Summary

The following table lists the modified SNR values along with other useful information of the interpolators.

Mod. SNR (dB)	N	$\times$	$\times$ + −	Type	Interpolator
19.1	2	**1**	**3**	2p **1o**	Linear
22.1	2	11	24	4p 5o	2o-osculating
23.5	2	9	19	4p 3o	Hermite
27.7	2	9	20	4p 3o	Lagrange
27.9	2	**5**	**12**	4p 2o	Watte tri-lin.
28.0	2	7	13	**2p** 3o	Optimal 2x
28.6	2	**6**	**13**	4p 2o	Parabolic 2x
29.9	2	23	47	6p 5o	2o-osculating
30.5	2	14	30	6p 3o	Hermite
31.0	2	22	47	6p 5o	Hermite
35.2	2	22	47	6p 5o	Lagrange
38.2	2	9	19	4p 3o	B-spline
45.1	2	**8**	**18**	4p **2o**	Optimal 2x
57.3	2	20	42	6p 5o	B-spline
65.9	2	**11**	**23**	4p **3o**	Optimal 2x
69.8	2	**14**	**28**	**4p** 4o	Optimal 2x
89.8	2	**19**	**40**	6p **4o**	Optimal 2x
111.4	2	**23**	**47**	**6p 5o**	Optimal 2x
33.8	4	**1**	**3**	2p **1o**	Linear
34.9	4	**5**	**12**	4p 2o	Watte tri-lin.
39.1	4	7	13	**2p** 3o	Optimal 4x
41.9	4	11	47	4p 5o	2o-osculating
44.2	4	9	19	4p 3o	Hermite
50.7	4	**6**	**13**	4p 2o	Parabolic 2x
52.8	4	9	20	4p 3o	Lagrange
60.2	4	14	30	6p 3o	Hermite
60.4	4	23	47	6p 5o	2o-osculating
62.3	4	22	47	6p 5o	Hermite
64.6	4	**8**	**18**	4p **2o**	Optimal 4x
67.6	4	**9**	**19**	4p 3o	B-spline
70.9	4	22	47	6p 5o	Lagrange
89.0	4	**11**	**23**	4p **3o**	Optimal 4x
101.1	4	**14**	**28**	**4p** 4o	Optimal 4x
101.4	4	20	42	6p 5o	B-spline
120.6	4	**19**	**40**	6p **4o**	Optimal 4x
149.3	4	**23**	**47**	**6p 5o**	Optimal 4x
46.8	8	5	12	4p 2o	Watte tri-lin.
47.0	8	**1**	**3**	2p **1o**	Linear
49.7	8	7	13	**2p** 3o	Optimal 8x
61.1	8	11	24	4p 5o	2o-osculating
64.0	8	9	19	4p 3o	Hermite
70.6	8	**6**	**13**	4p 2o	Parabolic 2x
77.7	8	9	20	4p 3o	Lagrange
83.5	8	**8**	**18**	4p **2o**	Optimal 8x
89.1	8	14	30	6p 3o	Hermite
91.4	8	23	47	6p 5o	2o-osculating
93.7	8	22	47	6p 5o	Hermite
94.1	8	**9**	**19**	4p 3o	B-spline
107.5	8	22	47	6p 5o	Lagrange
112.9	8	**11**	**23**	4p **3o**	Optimal 8x
126.4	8	**14**	**28**	**4p** 4o	Optimal 8x
141.1	8	20	42	6p 5o	B-spline
151.2	8	**19**	**40**	6p **4o**	Optimal 8x
185.4	8	**23**	**47**	**6p 5o**	Optimal 8x
59.3	16	5	12	4p 2o	Watte tri-lin.
59.7	16	**1**	**3**	2p **1o**	Linear
61.0	16	7	13	**2p** 3o	Optimal 16x
79.9	16	11	24	4p 5o	2o-osculating
83.1	16	9	19	4p 3o	Hermite
89.5	16	**6**	**13**	4p 2o	Parabolic 2x
101.9	16	**8**	**18**	4p **2o**	Optimal 16x
102.2	16	9	20	4p 3o	Lagrange
116.3	16	14	30	6p 3o	Hermite
119.3	16	**9**	**19**	4p 3o	B-spline
122.1	16	23	47	6p 5o	2o-osculating
124.7	16	22	47	6p 5o	Hermite
136.9	16	**11**	**23**	4p **3o**	Optimal 16x
144.1	16	22	47	6p 5o	Lagrange
150.7	16	**14**	**28**	**4p** 4o	Optimal 16x
179.0	16	20	42	6p 5o	B-spline
181.6	16	**19**	**40**	6p **4o**	Optimal 16x
221.5	16	**23**	**47**	**6p 5o**	Optimal 16x
71.8	32	5	12	4p 2o	Watte tri-lin.
72.0	32	**1**	**3**	2p **1o**	Linear
72.7	32	7	13	**2p** 3o	Optimal 32x
98.3	32	11	24	4p 5o	2o-osculating
101.8	32	9	19	4p 3o	Hermite
108.0	32	**6**	**13**	4p 2o	Parabolic 2x
120.2	32	**8**	**18**	4p **2o**	Optimal 32x
126.6	32	9	20	4p 3o	Lagrange
142.3	32	14	30	6p 3o	Hermite
143.9	32	**9**	**19**	4p 3o	B-spline
152.6	32	23	47	6p 5o	2o-osculating
155.4	32	22	47	6p 5o	Hermite
161.0	32	**11**	**23**	4p **3o**	Optimal 32x
174.9	32	**14**	**28**	**4p** 4o	Optimal 32x
180.5	32	22	47	6p 5o	Lagrange
212.0	32	**19**	**40**	6p **4o**	Optimal 32x
215.9	32	**20**	**42**	6p 5o	B-spline
257.8	32	**23**	**47**	**6p 5o**	Optimal 32x
84.3	64	1	3	2p 1o	Linear
96.6	128	1	3	2p 1o	(estimated)
108.9	256	1	3	2p 1o	
121.2	512	1	3	2p 1o	
133.5	1024	1	3	2p 1o	
145.8	2048	1	3	2p 1o	

Table: Modified SNR values of the compared interpolators in ascending order, for various oversampling ratios (N). The $\times$ column gives the total number of multiplications, and the $\times+-$ column the total number of mul/add/sub operations, required by the version of the interpolation routine that minimizes that count. The winning choices for each criterion (within the oversampling ratio) are marked with ***bold blue***

The optimal interpolators are superior (especially with low N) within interpolators of same order and samplepoints – no surprise, they were designed to be. They also often require the least operations for a given SNR.

For example, if one wants at least 100dB modified SNR with the least processing (total number of operations), the table suggests the following choices for the interpolator.

6-point, 5th-order optimal 2x with 2x oversampling (111.4dB);
4-point, 4th-order optimal 4x with 4x oversampling (101.1dB);
4-point, 3rd-order optimal 8x with 8x oversampling (112.9dB);
4-point, 2nd-order optimal 16x with 16x oversampling (101.9dB);
Parabolic 2x with 32x oversampling (108.0dB).

For higher oversampling ratios (N), linear interpolation is the best choice because of its implementation simplicity. At high N, the oversampled signal is usually not wholly generated, but evaluated only at points needed. In this light, the property of linear interpolation of operating only on two points is an important quality.

It is outside the scope of this paper to make guesses of the most profitable oversampling ratio. Also, it shall only be commented that using polynomial interpolators with unoversampled input is a choice that can only be made when the quality is not that important but speed is essential, the most useful interpolators in that case being linear and 4-point Hermite, and Watte tri-linear, which is somewhere between those two in both quality and computational complexity.

8. Pre-emphasis

Compensation for an interpolator passband attenuation can be done with a pre-emphasis filter. Doing the compensation after the whole resampling process would generally be harder, because the required filter would need to be different for different (possible varying) target samplerates. In addition to being harder, post-emphasis would also be less predictable, so pre-emphasis is presumed throughout this paper.

Design of the oversampling and pre-emphasis filter(s) has been left for the reader. To make this job easier, minmax-error polynomial approximations of the passband magnitude frequency responses of the interpolators are given in the following table, with a maximum error of $\pm$0.001dB. The x variable is the frequency in radians. $x = 0$ corresponds to 0Hz and $x = \pi$ to the passband edge frequency, i.e. the Nyquist frequency of the original sampledata before upsampling by N.

Type	Interpolator	N	Polynomial approximation of passband magnitude response
2p 1o	Linear	1	1 + x^2*-0.08322989624857348 + x^4*0.00271934406067139 + x^6*-0.00003971672040791
		2	1 + x^2*-0.02080088884033381 + x^4*0.00016313313160313
		4	1 + x^2*-0.00520782546507489 + x^4*0.00001068495970745
		8	1 + x^2*-0.00129654618588707
		16	1 + x^2*-0.00032517439633740
		32	1 + x^2*-0.00008135855031023
4p 3o	B-spline	1	1 + x^2*-0.16640502255773582 + x^4*0.01228017737024591 + x^6*-0.00050027796209637 + x^8*0.00000963132802828
		2	1 + x^2*-0.04165500855002408 + x^4*0.00077433197774282 + x^6*-0.00000758254328763
		4	1 + x^2*-0.01041096020557994 + x^4*0.00004696830873626
		8	1 + x^2*-0.00257931742447139
		16	1 + x^2*-0.00064948375683407
		32	1 + x^2*-0.00016266297211713
6p 5o	B-spline	1	1 + x^2*-0.24971133684507330 + x^4*0.02883604361622073 + x^6*-0.00198230583493742 + x^8*0.00008180902955303 + x^10*-0.00000158312808302
		2	1 + x^2*-0.06242909544523705 + x^4*0.00178139138425410 + x^6*-0.00002582908012259
		4	1 + x^2*-0.01560372802650468 + x^4*0.00010700525591829
		8	1 + x^2*-0.00390591284558800 + x^4*0.00000701051188486
		16	1 + x^2*-0.00097292994241284
		32	1 + x^2*-0.00024394341875457
4p 3o	Lagrange	1	1 + x^2*-0.00009488096453117 + x^4*-0.01514677142173045 + x^6*0.00146109576356452 + x^8*-0.00006419369304970 + x^10*0.00000123063707050
		2	1 + x^2*-0.00004518371420257 + x^4*-0.00092664677942848 + x^6*0.00001866341219874
		4	1 + x^2*-0.00001484719751166 + x^4*-0.00005475309363511
		8	1 + x^2*-0.00003007293026739
		16	1 + x^2*-0.00000189939505604
		32	1 + x^2*-0.00000011902044674
6p 5o	Lagrange	1	1 + x^2*0.00031096876457228 + x^4*-0.00061518445014183 + x^6*-0.00274624322037590 + x^8*0.00042005143809930 + x^10*-0.00002565329098802 + x^12*0.00000060672095234
		2	1 + x^2*0.00023630204175958 + x^4*-0.00015542700665419 + x^6*-0.00001887712089898
		4	1 + x^2*0.00002716600105685 + x^4*-0.00000934958595234
		8	1 + x^2*-0.00000085807324002
		16	1 + x^2*0.00000001124268487
		32	1 + x^2*-0.00000003076348541
4p 3o	Hermite	1	1 + x^2*-0.00005333203257415 + x^4*-0.01242578041093850 + x^6*0.00109014040055239 + x^8*-0.00004417926195795 + x^10*0.00000079906102798
		2	1 + x^2*-0.00003062028167362 + x^4*-0.00076214500513774 + x^6*0.00001412073481103
		4	1 + x^2*-0.00001102344703875 + x^4*-0.00004517529147094
		8	1 + x^2*-0.00002463837055619
		16	1 + x^2*-0.00000155457581194
		32	1 + x^2*-0.00000009738219240

Table: Polynomial approximations of interpolator passband responses (. . .)

Type	Interpolator	N	Polynomial approximation of passband magnitude response
6p 3o	Hermite	1	`1 + x^2*0.00018259576650771 + x^4*-0.00175056161346116 + x^6*-0.00194042192557450 + x^8*0.00029325481059871 + x^10*-0.00001735249351243 + x^12*0.00000039974321014`
		2	`1 + x^2*0.00016315772984362 + x^4*-0.00019365493638948 + x^6*-0.00001330449384056`
		4	`1 + x^2*0.00001886323801592 + x^4*-0.00001190843952919`
		8	`1 + x^2*-0.00000337009968184`
		16	`1 + x^2*-0.00000018280736434`
		32	`1 + x^2*-0.00000001096502600`
6p 5o	Hermite	1	`1 + x^2*0.00020114336420553 + x^4*-0.00039890572473931 + x^6*-0.00229556927556134 + x^8*0.00033156072749299 + x^10*-0.00001939518424822 + x^12*0.00000044492173307`
		2	`1 + x^2*0.00018449827272179 + x^4*-0.00012076518704688 + x^6*-0.00001656712931157`
		4	`1 + x^2*0.00002231671196743 + x^4*-0.00000765304111907`
		8	`1 + x^2*-0.00000069755852979`
		16	`1 + x^2*-0.00000003814496217`
		32	`1 + x^2*-0.00000005120943160`
4p 5o	2o-osculating	1	`1 + x^2*-0.00003810731746664 + x^4*-0.01125635108311614 + x^6*0.00093446715934087 + x^8*-0.00003596438901587 + x^10*0.00000062492114797`
		2	`1 + x^2*-0.00002472867620723 + x^4*-0.00069142841070279 + x^6*0.00001220808924164`
		4	`1 + x^2*-0.00000942733874677 + x^4*-0.00004105609626839`
		8	`1 + x^2*-0.00002230797701725`
		16	`1 + x^2*-0.00000086708780524`
		32	`1 + x^2*0.00000015624344717`
6p 5o	2o-osculating	1	`1 + x^2*0.00016728745643872 + x^4*-0.00033180499542066 + x^6*-0.00214224354648566 + x^8*0.00030186179619621 + x^10*-0.00001731224751541 + x^12*0.00000039135210540`
		2	`1 + x^2*0.00016735375130858 + x^4*-0.00010930555207958 + x^6*-0.00001577254673146`
		4	`1 + x^2*0.00002069410150064 + x^4*-0.00000708601491651`
		8	`1 + x^2*-0.00000032041779301`
		16	`1 + x^2*0.00000047022487877`
		32	`1 + x^2*0.00000011755621969`
4p 2o	Watte tri-lin.	1	`1 + x^2*0.08310376973158973 + x^4*-0.03579156791607385 + x^6*0.00334209365628805 + x^8*-0.00014883957926161 + x^10*0.00000289107920861`
		2	`1 + x^2*0.02073112144393299 + x^4*-0.00219214872746441 + x^6*0.00004267336546731`
		4	`1 + x^2*0.00517460473851293 + x^4*-0.00012980516326426`
		8	`1 + x^2*0.00130153324464790 + x^4*-0.00000863514074959`
		16	`1 + x^2*0.00032103211425380`
		32	`1 + x^2*0.00008109888849582`
4p 2o	Parabolic 2x	2	`1 + x^2*-0.06246359855072939 + x^4*0.00154144387052042 + x^6*-0.00001835446617397`
		4	`1 + x^2*-0.01561067013990179 + x^4*0.00009300879607293`
		8	`1 + x^2*-0.00385657590779811`
		16	`1 + x^2*-0.00097344704925316`
		32	`1 + x^2*-0.00024394574160357`

Table: Polynomial approximations of interpolator passband responses (. . .)

Type	Interpolator	N	Polynomial approximation of passband magnitude response
2p 3o	Optimal 2x	2	`1 + x^2*-0.02554619677843794 + x^4*0.00024013379174846`
	Optimal 4x	4	`1 + x^2*-0.00574208273496733 + x^4*0.00001315216245712`
	Optimal 8x	8	`1 + x^2*-0.00135618642176540`
	Optimal 16x	16	`1 + x^2*-0.00033233240729770`
	Optimal 32x	32	`1 + x^2*-0.00008223446937757`
4p 2o	Optimal 2x	2	`1 + x^2*-0.06789983241024726 + x^4*0.00187396896433240 + x^6*-0.00002473496367671`
	Optimal 4x	4	`1 + x^2*-0.02170313823333328 + x^4*0.00015782661814829`
	Optimal 8x	8	`1 + x^2*-0.00728405982348531 + x^4*0.00001497426641413`
	Optimal 16x	16	`1 + x^2*-0.00279971126703223`
	Optimal 32x	32	`1 + x^2*-0.00119086264779340`
4p 3o	Optimal 2x	2	`1 + x^2*-0.05172434622344999 + x^4*0.00118064718929264 + x^6*-0.00001377425931894`
	Optimal 4x	4	`1 + x^2*-0.01252685934677141 + x^4*0.00006772267827786`
	Optimal 8x	8	`1 + x^2*-0.00305463592329826`
	Optimal 16x	16	`1 + x^2*-0.00076445772774392`
	Optimal 32x	32	`1 + x^2*-0.00019096370951489`
4p 4o	Optimal 2x	2	`1 + x^2*-0.05288134845267101 + x^4*0.00123174456487270 + x^6*-0.00001459486840119`
	Optimal 4x	4	`1 + x^2*-0.01209875686296254 + x^4*0.00006330069200420`
	Optimal 8x	8	`1 + x^2*-0.00293014841707120`
	Optimal 16x	16	`1 + x^2*-0.00073452725013589`
	Optimal 32x	32	`1 + x^2*-0.00018375385546417`
6p 4o	Optimal 2x	2	`1 + x^2*-0.09926567184099210 + x^4*0.00436972559074357 + x^6*-0.00011145259389693 + x^8*0.00000150855440787`
	Optimal 4x	4	`1 + x^2*-0.04104326668505586 + x^4*0.00054795446283906 + x^6*-0.00000365948234725`
	Optimal 8x	8	`1 + x^2*-0.01684372350477612 + x^4*0.00006032275579243`
	Optimal 16x	16	`1 + x^2*-0.00764482449535352 + x^4*0.00000744158606948`
	Optimal 32x	32	`1 + x^2*-0.00357265950908555`
6p 5o	Optimal 2x	2	`1 + x^2*-0.08084154876655289 + x^4*0.00293944745701822 + x^6*-0.00005171508704785`
	Optimal 4x	4	`1 + x^2*-0.01880393544082998 + x^4*0.00015408893922917`
	Optimal 8x	8	`1 + x^2*-0.00465275707182741 + x^4*0.00000997264487197`
	Optimal 16x	16	`1 + x^2*-0.00114503636967797`
	Optimal 32x	32	`1 + x^2*-0.00026762900966793`

Table: Polynomial approximations of interpolator passband responses

9. Conclusion

The presented optimal interpolators make it possible to do transparent-quality resampling for even the most demanding applications with only 2x or 4x oversampling before the interpolation. However, in most cases simple linear interpolation combined with a very high-ratio oversampling (perhaps 512x) is the optimal tradeoff. The computational costs depend on the platform and the oversampling implementation.

Therefore, which interpolator is the best is not concluded here. You must first decide what quality you need (for example around 90dB modified SNR for a transparency of 16 bits) and then see what alternatives the table given in the summary has to suggest for the oversampling ratios you can afford.