# ABOUT RUNNING TRANSFORMS[1]

James A. Moorer 2016-2025

*Please be notified that <u>everything</u> in this discussion is completely obvious and follows directly from principles everyone learns in DSP 101. There is nothing new here. Having said that, I still consider it important to repeat this material from time to time, since it appears to be routinely overlooked.*

Let us start with a definition of a zero-padded short-term discrete Fourier transform:

$$(1) \quad X_k(n) = \sum_{m=0}^{N-1} x(n-m)e^{2\pi jmk/M}$$

where

| | |
|---|---|
| $n$ | Time index |
| $k$ | Frequency index, $0 \leq k < M$ |
| $N$ | Number of input points per frame |
| $M$ | Transform size ($M \geq N$) |

Note that the sign of the exponent is positive, rather than the conventional negative sign used in the definition of the Fourier transform. This is because in this definition, the time index, $m$, actually represents *previous* points in time. Defining the short-term transform this way allows us to take an inverse FFT of $X_k(n)$ at any time $n$ and get the samples in the usual order. I will sometimes use the term "frame" (at time index $n$) as a shorthand for the inputs points $x(n-N+1) \ldots x(n)$.

*In this formulation, we can recover the original signal (scaled by M) simply by summing $X_k(n)$ over all values of k for each frame. That is, it is <u>not necessary</u> to take an explicit inverse transform. I will call this the*

[1] Many thanks to David Levine and Julius Smith for their help in doing this research and writing the paper.

*"direct sum" property*[2]. For complex input data, we must take the complex conjugate of the direct sum to get an identity.

Let us define the complex exponential weight factor as follows:

$$(2) \quad W_M \equiv e^{2\pi j/M}$$

We can then write the 1-step recursive update as follows:

$$(3) \quad X_k(n) = x(n) + X_k(n-1)(W_M)^k - x(n-N)(W_M)^{kN}$$

This defines a "running" transform that implements zero padding with (M-N) zeros. An unpadded version of relation was noted by Gold and Rader [**GR**], among others[3]. It can be viewed as a kind of comb filter with a complex coefficient and state. The direct sum property emphasizes the interpretation of the running transform as a bank of bandsplitting filters [**MB**], since we would expect that a properly-designed bandsplitting filter would have this property. Since we are not using the fast Fourier transform algorithm, there is no particular limitation on the choice of *M* and *N* – for instance, they might both be prime numbers.

This transform as defined above has only a Fourier window. It is interesting to see if we can incorporate the application of a window function to the recursive implementation.

We will take as a family of window functions those made by sums of cosines:

$$(4) \quad w_{ND}(m) = \sum_{d=0}^{D} a_d \cos(2\pi md/N)$$

---

[2] Of course, if $x(n)$ is a real signal, then we need only sum over the real part of $X_k(n)$

[3] The implementation in Gold & Rader involves a single real-valued comb filter. This can be done only when $M = N$. In the general case, there are *M* complex-valued comb filters as implied by equation (3).

This implements the common window functions, such as Hamming, Hann, and Blackman. Additionally, Rife and Vincent [**RV**] defined comprehensive families of windows functions using this same form with a variety of properties, including approximations to Chebychev windows. Moorer and Berger [**MB**] further noted that *any* window function can be implemented as a sum of sines and cosines. This is a direct consequence of the definition of Fourier's sine and cosine series. We start assuming that only cosines are used in the definition. The extension to sines and cosines is then straightforward.

**Window as Weighted Sum in Frequency Domain**

To derive the recursive form of a short-term transform where the input data is windowed by (4) at each point in time, we start by examining the effect of applying a single cosine in the equivalent form as a sum of two exponentials:

(5) $\frac{1}{2}(e^{2\pi jmd/N} + e^{-2\pi jmd/N})$

We now define two separate exponential weights as follows:

(6) $W_{NMd}^{+} = e^{2\pi j\left(\frac{d}{N}+\frac{k}{M}\right)}$

(7) $W_{NMd}^{-} = e^{2\pi j\left(-\frac{d}{N}+\frac{k}{M}\right)}$

The recursion can now be detailed as follows:

(8) $X_{kd}^{+}(n) = x(n) + X_{kd}^{+}(n-1)(W_{NMd}^{+})^{k} - x(n-N)(W_{NMd}^{+})^{kN}$

(9) $X_{kd}^{-}(n) = x(n) + X_{kd}^{-}(n-1)(W_{NMd}^{-})^{k} - x(n-N)(W_{NMd}^{-})^{kN}$

(10) $X_{kd}(n) = (X_{kd}^{+}(n)+X_{kd}^{-}(n))/2$

This is a recursive formulation of one term of a windowed transform where the window function is a weighted sum of cosines.

(11) $X_k(n) = \sum_{d=0}^{D} a_d X_{kd}(n)$

The extension to the use of both sines and cosines is straightforward. We can express the most general definition of a window function as follows:

(12) $w_{ND}(m) \equiv \sum_{d=0}^{D} a_d \cos\left(\frac{2\pi md}{N}\right) + \sum_{d=0}^{D} b_d \sin(\frac{2\pi md}{N})$

The revised version of equation (12) can then be expressed this way:

(13) $X_k(n) = 1/2 \sum_{d=0}^{D} (a_d + b_d) X_{kd}^{+}(n) + (a_d - b_d) X_{kd}^{-}(n)$

This is sufficient to generalize the window function to realize *any* desired function, symmetric or otherwise.

Now let us take the case where the transform size is an integral multiple of the number of non-zero input points.

(14) $M \equiv \alpha N$ where α is an integer

The weight functions then become these:

(15) $W_{NMd}^{+} = e^{2\pi j\left(\frac{\alpha d + k}{M}\right)}$

(16) $W_{NMd}^{-} = e^{2\pi j\left(\frac{-\alpha d + k}{M}\right)}$

We will recognize (8) and (9) as being the same as (3), shifted by $\alpha$ channels. Thus, when (14) holds, we do not need to run separate recursions as described by (8) and (9), but can simply use the iteration of (3) and take channels separated by $\alpha$.

Note what this is telling us. *We can implement any window function consisting of a sum of cosines by weighted sums of frequency channels separated by α.* When $M$ is not an integral multiple of $N$, implementation of a (time) window function in the frequency domain is not simple. In addition, if the window is summable, then the direct-sum property is available.

The equations of (15) and (16) comprise a well-known result in the case when $M=N$. That is, a multiplication in the time domain may be replaced by a convolution in the frequency domain. The common Hamming and Hann windows can be implemented by 3-point sums across the frequency bands. It is generally not done this way because it is more efficient to apply the window in the time domain. In the case of a running transform, there can be advantages to applying the window in the frequency domain. This property of cosine-based windows was noted as well in **[JAM1].**

As previously noted, Gold and Rader [**GR**] described what they termed "frequency-sampling filtering" which can be derived as a specialization of equation (1). They did not extend it to zero-padded transforms. Their implementation involved a single real comb filter followed by a bank of real-valued resonators. Ours uses a bank of $M$ complex comb filters so that we get complex channel outputs thus providing both amplitude and phase information directly. It is straightforward to specialize our formulation to that of [**GR**] by eliminating padding and ignoring the imaginary portion of the transform data.

## Window Functions and Direct Sum Failure

The intrepid programmer might be a bit startled upon noticing that the direct sum for certain window functions produces all zeros. For instance, the first sample of Hann window is zero. Consequently, the direct sum using the Hann window will always be zero, since the point-by-point product of the input signal with the Hann window will always start with a zero sample. There are workarounds for this issue. First, the transform

can be shifted (see equations (20) to (22) below) to select a sample where the window function is non-zero. For the Hann window, we can simply reverse the window so the zero sample is at the end of the window rather than at the beginning. For $N = M$, this can still be implemented as a 3-point frequency-domain weighted sum, but the coefficients will be complex. For instance, the Hann window coefficients are normally (-.5, 1, -.5). To reverse the Hann window, placing the zero as the last point of the window rather than the first, the coefficients will be this:

$$(17) \quad \left(-\sqrt{{}^1/_4 - ({}^\pi/_N)^2} - {}^\pi/_N j,\ 1,\ -\sqrt{{}^1/_4 - ({}^\pi/_N)^2} + {}^\pi/_N j\right).$$

**There are Windows then there are Windows**

It is useful to ask what the point of a window function is. The main reason is to reduce the "edge effects" of taking a finite number of points. A discontinuity at the beginning or ending of the frame will reduce the rate of decay of the sidebands of the frequency response of the window function. Figure 1 compares the frequency responses of the Hamming window and the Hann window:

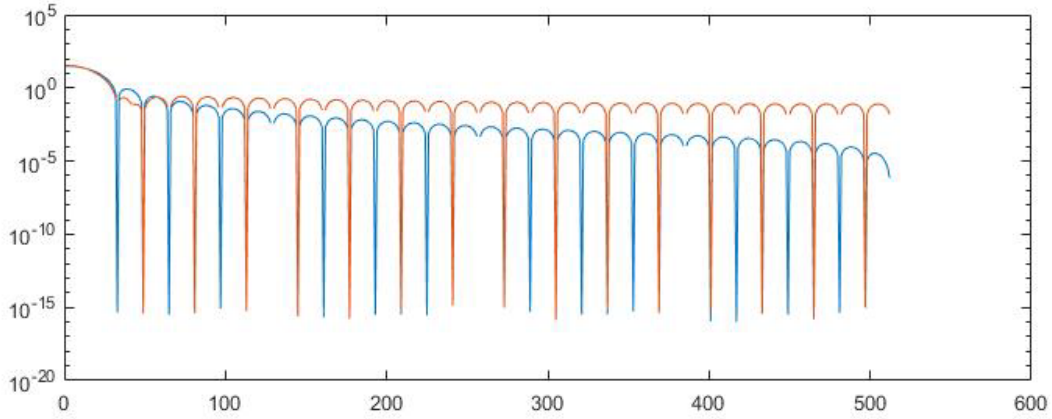


**Figure 1:** Comparison of roll-off rate of Hamming (red) versus Hann (blue) windows.

The Hamming window is often selected because it suppresses the first sideband, but this is at the cost of allowing a discontinuity at the endpoints of the data. The Hann window is zero on the first (or last) point of the window, and thus has a more rapid decay as we move away from the center frequency. Signals such as audio exhibit tremendous dynamic range from low frequencies to high frequencies. In general, most of the energy in an audio signal is concentrated in the low frequencies – maybe 1500 Hz or less. Without the suppression of "leakage" from low frequencies to high frequencies, it is possible for the high-frequency information to become swamped by this leakage from the lower frequency bands.

Note that the only reason equations (5)-(16) work is because the data length is $N$ and is padded with $(N - M)$ zeros. That is, there is *no* data in $(N - M)$ points. This may seem tautological, since it is true *by construction*. That is, we started by declaring this as the definition of the problem to be solved. I mention it because people (myself included) sometimes assume that a window function using a frequency-domain convolution as described by equations (5)-(16) can be used to *time-limit* a sequence of more than $N$ data points down to $N$ data points. It cannot. It is possible to define a frequency-domain convolution that will implement an approximation to what is called a "brick-wall window" in **JAM4** that will have the effect of attenuating the data before and after the frame. It will involve somewhat more computation than described above. The 1-step recursion in (3) forces the data length to be exactly $N$ data points, so the relatively simple equations (5)-(16) suffice for implementing a window function.

## Running Transform is a (Complex) Linear Filter

We can view equation (1) as $M$ complex linear filters. It is instructive to calculate the response to a pure sinusoid by starting with a signal that is a complex exponential:

$$(18)\quad x(n) = e^{jn\theta}$$

The response can be written as follows:

$$(19)\quad X_k(n) = e^{jn\theta}\left(\frac{1-e^{j(2\pi\frac{k}{\alpha}-N\theta)}}{1-e^{j(2\pi\frac{k}{M}-\theta)}}\right)$$

The input sinusoid will be scaled by a complex transfer gain that depends on *M, N, k,* and $\theta$, and does not depend on *n.* This differs from the usual formula by the inclusion of zero-padding. Note the absence of "heterodyning" in equation (19). That is, the original frequency of the source sinusoid is present in *all* channels. It is not shifted by the center frequency of the channel. This is a direct result of the formulation given in equation (1).

**Time-Shifting a Running Transform**

Note that equation (1) is defined with the data as the last *N* points in an *M*-point analysis window. We may want to locate the data anywhere in the window. Let us define σ as the desired sample offset such that $0 \leq \sigma < M - N$. We can rewrite equation (1) as follows:

$$(20)\quad \tilde{X}_k(n) = \sum_{m=-\sigma}^{N-\sigma-1} x(n-m)e^{2\pi jmk/M}$$

It follows then that the relation between equations (20) and (1) is as follows:

$$(21)\quad \tilde{X}_k(n) = e^{-2\pi\sigma jk/M}X_k(n) = (W_M)^{-\sigma k}X_k(n)$$

We can then center the data in the analysis window by setting σ to $(M - N)/2$. We can window the data, if needed, using equations (16) and (17), but this constrains the choice of *M* and *N.* We must set α in equation (15) to an *odd* integer to apply a window.

Note that equation (21) can be used to shift the data to any position in the window. For example, we can center the data around zero by setting σ to *M*/2. Additionally, we can incorporate shifting directly into the recursion of equation (3) as follows:

$$(22)\quad \tilde{X}_k(n) = \tilde{X}_k(n-1)(W_M)^k + (W_M)^{-\sigma k}(x(n) - x(n-N)(W_M)^{kN})$$

This iteration yields a shifted running transform. It does preserve the direct sum property. The direct sum selects the sample that was originally at $m = \sigma$. Note that the useful range for the shift amount is between 0 and *N-1*. The direct sum of a transform with a shift outside this range will produce only zeros.

One particularly simple shift is *M/2* samples. This has a very simple form. We can think of the direct-sum property as calculating the dot product of the unit vector {1, 1, 1, . . .} with the current frame. If we instead form the dot product with {-1, 1, -1, 1, . . .}, we realize a shift of *M/2* points, and allows the use of the Hann window thus implementing a form of the direct sum property in a very efficient manner. One side effect, however, is that it introduces a delay of *M/2* samples when viewing the running transform as a filter bank. This may be a problem for some applications.

It is also interesting to examine the interpretation of $\tilde{X}_k(n-1)(W_M)^k$. In general, multiplication by $(W_M)^k$ will advance the time index of the data window by one sample. We might ask then what is the direct sum of $\tilde{X}_k(n-1)(W_M)^k$? It cannot be $\tilde{X}_k(n)$, because $x(n)$ has not been introduced. The answer is that the direct sum is equal to $x(n-N)$. It "picks off" the last data point in the window because of the circularity of the transform. You advance the transform one point into the future and the last point gets rotated around into the position of the first sample of the sequence. The construction of (1) and thus of (3) produces a frequency transform of the *periodic* signal consisting of repeating copies

of the current frame. However you think of the sequence $x(n)$, after you take this transform, you have the transform of a strictly periodic sequence. This was pointed out, for instance, by Stockham (**TS**) and many others.

## Frequency Offset in a Running Transform

Gold and Rader note that the transform can be shifted by ½ bin by changing the sign of the comb filter in their implementation. We can define a shifted transform as follows:

(23) $\hat{X}_k(n) = \sum_{m=0}^{N-1} x(n-m)e^{2\pi jm(k+\beta)/M}$

We set $\beta = 1/2$ to get the case described by Gold and Rader. This can be computed with the following recursion formula:

(24) $\hat{X}_k(n) = x(n) + \hat{X}_k(n-1)(W_M)^{k+\beta} - x(n-N)(W_M)^{N(k+\beta)}$

Note that if $\beta$ is not either 0 or ½, then we lose the convenient property that the real and imaginary parts of the transform of a real sequence are even and odd, respectively.

An odd thing happens with the direct sum.

(25) $\sum_{k=0}^{M-1} \hat{X}_k(n) = \sum_{k=0}^{M-1} \sum_{m=0}^{N-1} x(n-m)e^{2\pi jm(k+\beta)/M}$

Reverse the order of summation and extract the term with $\beta$ to get:

(26) $\sum_{k=0}^{M-1} \hat{X}_k(n) = \sum_{m=0}^{N-1} x(n-m)e^{2\pi jm\beta/M} \sum_{k=0}^{M-1} e^{2\pi jmk/M}$

(27) $\sum_{k=0}^{M-1} \hat{X}_k(n) = \sum_{m=0}^{N-1} x(n-m)e^{2\pi jm\beta/M} \frac{1-e^{2\pi jm}}{1-e^{2\pi jm/M}}$

(28) $\sum_{k=0}^{M-1} \hat{X}_k(n) = M\, x(n)$

That is, the direct sum is independent of $\beta$.

## Time-Shift Direct Sum

We can use time-shifting in combination with direct sum to recover any input point from $X_k(n)$. We start by multiplying by $(W_M)^{-\sigma k}$, then summing over *k*.

$$(29)\quad \sum_{k=0}^{M-1}(W_M)^{-\sigma k}X_k(n) = \sum_{k=0}^{M-1}\sum_{m=0}^{N-1} x(n-m)(W_M)^{-\sigma k}(W_M)^{km}$$

Reverse order of summation:

$$(30)\quad \sum_{k=0}^{M-1}(W_M)^{-\sigma k}X_k(n) = \sum_{m=0}^{N-1} x(n-m)\sum_{k=0}^{M-1}(W_M)^{-\sigma k}(W_M)^{km}$$

$$(31)\quad = Mx(n-\sigma)$$

Just to remind us of the obvious, equation (31) for $\sigma = 0$ is just the direct sum of $X_k(n)$. Equation (30) is the generalization of the direct sum to recover any point of the input sequence. Or, if you prefer, it represents the dot product of one row of the (implied) inverse DFT matrix and $X_k(n)$. The direct sum just represents the dot product of $X_k(n)$ with the first row of the inverse DFT matrix (all ones).

## Combination Time and Frequency Shift

For completeness, we should derive a formula for the combination of time shift and direct sum to extract an input sample from a frequency shifted transform. We will start as before with the basic equation:

(32) $\widetilde{X_k}(n) = \sum_{m=0}^{N-1} x(n-m)(W_M)^{m(k+\beta)}$

Now take the direct sum of both sides and exchange order of summation.

(33) $\sum_{k=0}^{M-1} (W_M)^{-\sigma k}\widetilde{X_k}(n) = \sum_{m=0}^{N-1} x(n-m) \sum_{k=0}^{M-1} (W_M)^{-\sigma k}(W_M)^{m(k+\beta)}$

(34) $\sum_{k=0}^{M-1} (W_M)^{-\sigma k} \widetilde{X_k}(n) =$

$$\sum_{m=0}^{N-1} x(n-m) (W_M)^{m\beta} \sum_{k=0}^{M-1} (W_M)^{-\sigma k}(W_M)^{mk}$$

The last summation will take on the value $M$ at the point $m = \sigma$, simplifying the RHS of equation (34) as follows:

(35) $\sum_{k=0}^{M-1} (W_M)^{-\sigma k} \widetilde{X_k}(n) = Mx(n-\sigma)(W_M)^{\sigma\beta}$

To extract sample $x(n-\sigma)$ from a transform that has been frequency-shifted by $\beta$, you have to scale the direct sum as follows:

(36) $x(n-\sigma) = \frac{1}{M}(W_M)^{-\sigma\beta} \sum_{k=0}^{M-1} (W_M)^{-\sigma k} \widetilde{X_k}(n)$

**Filtering a Running Transform**

The direct sum yields the sample at *m=0*, where *m* is defined in equation (1). We can shift the transform to place any sample at *m=0*, and thus we can cause any sample to be produced by the direct sum. If we shift the midpoint of the data to zero, then we can multiply the transform by the transform of any FIR filter of length less than *N*. Application of such a filter will allow us to use the direct sum to produce the output point – no explicit inverse transform is necessary. This is especially interesting for time-varying filters, where each point in time may have a different filter. We have shown that the only requirement is that the impulse response of the filter be *N* points or less in length and the direct sum will be exactly the convolution of the input signal with that filter.

We might point out that the usual rules of FIR filter design apply here. That is, we may want to window the impulse response of the desired filter to eliminate discontinuities at the edges that might excite Gibbs phenomenon ripples in the frequency response. This windowing can be applied as a convolution in the frequency domain [**MB**]. This basic principle is well-known, but it is generally not mentioned in the discussion of frequency-sampling filters or running transforms generally.

For instance, Gold and Rader designed their bandpass filter by manually adjusting the edge coefficient. We can design a very similar bandpass filter by starting with five adjacent channels of unity gain, then applying a Hamming window. This produces the gains (0.23, 0.77, 1, 1, 1, 0.77, 0.23), as opposed to the gains reported by Gold and Rader (0.221, 0.707, 1, 1, 1, 0 .707,  0.221). Figure 2 shows a comparison of these two designs plus one derived by using the Blackman window.


**Figure 2** – Comparison of frequency-sampling filters. Blue is original Gold and Rader [GR] design. Red is design using Hamming window. Green is using Blackman window.

This example is a linear-phase filter. It need not be. By use of the time shift, we can realize filters with other phase properties, such as minimum or maximum phase filters.

Gold and Rader's definition of frequency-sampling filter is based on an unpadded frequency analysis ($M=N$). When using a padded transform, the application of the window function to a prototype or ideal frequency response must take this into account, as implied by equations (16) and (17).

Note also that any frequency offset as defined in equation (23) must be taken into account when designing the filter in the frequency domain.

An interesting result of this formulation is that if we produce the output by direct sum, time-aliasing is not possible *by construction*. *Note that this is independent of padding and is true whether padding is used or not.* Equivalently, any time-aliasing is necessarily already "baked-into" the filter by the choice of frequency weights. This is really a discussion about filter design. As noted above, we will surely want to taper the edges of the bands in some smooth manner so as to reduce the discontinuities at the edges of the $N$-point analysis window.

**Combining Channels**

It is worth repeating some comments in [**MB**] relating to combining channels. It follows immediately from the direct sum property that we can make any desired grouping (weighted sums) of the channels as long as we respect the direct sum property (that is, the sum of the gains for each channel across all groups should be unity. Channel gains are otherwise unconstrained – they may be negative or even complex). For example, we may wish to combine the channels into a perceptually-relevant grouping, such as a Bark or ERB scale [**JOS**]. Of course, it is not limited to just simply summing down to, say, 24 or 26 discrete bands – we might wish to use overlapping, redundant bands with 4, 5 or more overlays, each of width of 1 Bark at the center frequency. Some window

functions, such as the Hann window, can be used to reduce edge effects without disturbing the direct sum property. We might wish to do this kind of redundant overlapping to better mimic the kind of frequency analysis done in the inner ear.

Similarly, we may want to group these channels into constant-Q groups at any desired number of divisions per octave. Again, window functions may be used to taper the band edges seamlessly. Please note one interpretation of this result – *A constant-Q transform may be implemented by a Discrete Fourier transform, either of fixed frame or as a "running" transform.* Note also that this definition of a constant-Q transform is, by construction, an identity. We may start with the direct sum property and work backwards to a set of channel group coefficients that preserve this property, and thus implement an identity. With many kinds of processing, it is of some comfort to start with a transform that is guaranteed to be an identity (in the absence of modification).

These groupings may be used to direct either filtering or analysis in any desired way. Note that any grouping of these channels will be *complex*, yielding both amplitude and phase (or, equivalently, I-Q quadrature components). In the case of real data, the imaginary portion may be either retained or ignored, depending on whether phase information is important.

One issue with combining channels this way is that the band edge frequencies, center frequencies, and band widths are quantized by $N$. In this case we can pad the transform by $M$-$N$ zeros. We can increase the number of channels by the factor of $\alpha$ (equation (13)). These additional channels will be redundant, but they can be used to reduce the band grouping quantization to any desired level. Increasing the number of channels by increasing $M$ does not, by itself, increase the selectivity of the windowing function, but it does increase the *accuracy* of the band edge frequencies, center frequencies, and band widths.

## Integer and Half-Integer Transforms

There is an interesting consequence of the frequency shifting described by equations (23) and (24). This consequence is not noted in most discussions of frequency-sampling filters. In designing a frequency-sampling filter, *one may freely intermix channels from any value of* $\beta$. Of course, if the result is to be, say, a bank of bandsplitting filters that should sum to unity, then all the channel gains for any particular value of β must also sum to unity (or negative unity). This gives us another degree of freedom in adjusting the filters produced by grouping (weighted summing) channels together. For instance, to produce constant-Q filters, we can increase the precision of the band edges by increasing *M*, or by adding in channels with $\beta = 1/2$[4]. Note that these two methods of increasing the precision of the band edges are not directly interchangeable – there is a fundamental difference. With $\beta = 0$, the channels are centered on half-integer values of *k*. With $\beta = 1/2$, the channels are centered on integer values of *k*. A sinusoid that is exactly between two adjacent channels with $\beta = 0$ will still be exactly between two adjacent channels for any integer multiple of *M,* but will be centered in a channel with $\beta = 1/2$. This may make a difference in some cases.

Figure 3 shows the result of combining adjacent channels from integer and half-integer transforms, showing that the result is a peak between the two adjacent peaks at a somewhat lower amplitude.

[4] I do not claim this as an original observation. I'm sure it has been noted elsewhere – I just haven't found it.

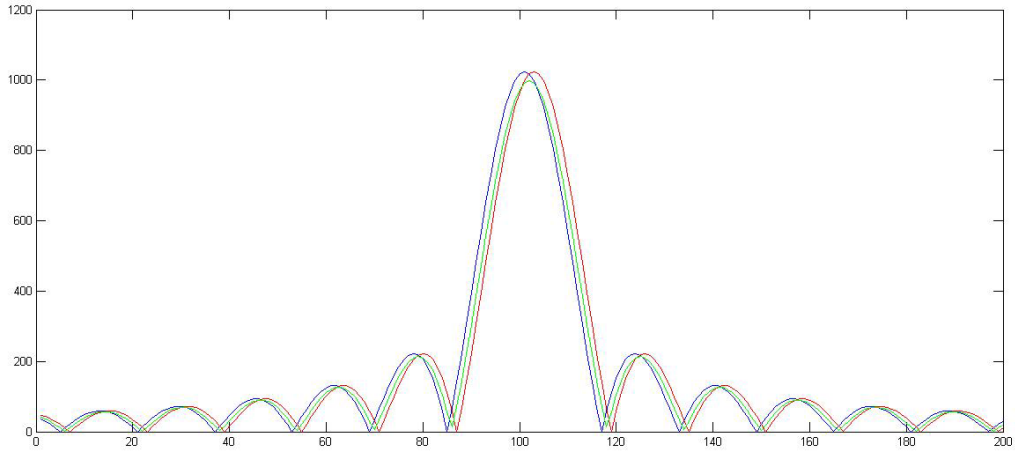


**Figure 3** – Comparison of frequency response of a channel with $\beta = 0$ (blue), a channel with $\beta = 1/2$ (red), and the sum of the two (green).

There is "crosstalk" between channels of integer and half-integer transforms, so designing a filter with both integer and half-integer elements would have to take this into account to get a specific transfer function.

There are a number of observations about half-integer transforms that should be kept in mind. First, half-integer filtering is not zero-phase. It introduces a ½-sample time-shift in the signal. Next, integer and half-integer channel responses show an exchange of zeros and extrema – that is, the zeros of one occur at the extrema of the other. This can be used to eliminate zeros in the frequency response of a filter. Figure 4 shows an unmodified channel filter and a sum of that half-integer filter minus ½ of the two adjacent integer filters. Note that the response is very smooth and without zeros. There doesn't seem to be another way to accomplish this except by summing many channels with nonzero coefficients.

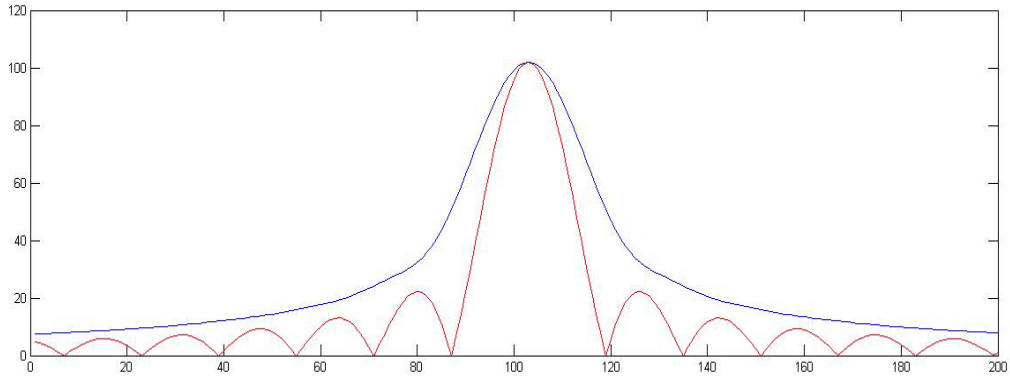


**Figure 4 –** Comparison of the unmodified magnitude response of a half-integer channel filter (red) and a combination of this channel filter minus ½ of the adjacent integer channel filters.

## About Audio Applications

As noted in [**MB**], the running transform can be used to make a graphic equalizer that has some properties not always found in such devices:

- Setting the gain of all the bands to the same boost/cut amount produces a *flat* frequency response. Traditional designs show ripple at any value of boost/cut different from zero.
- Once the iteration of (3) is done, the channels can be grouped in any manner – we are not limited to a fixed number of groups. The grouping of the channels can even be *time-varying*.
- The individual groups do not need to resemble second-order resonances – they can be bandpass filters, overlapping filters, mirror filters, harmonically-related filters, or any desired shape, as long as the direct sum property is preserved.
- As noted above, the band spacing can be constant-Q, constant-Bark, or any other grouping.

In [**MB**], we also noted that signal enhancement such as noise reduction can be implemented by applying dynamics processing to each channel group. This implementation, using equation (3), has some advantages over the block-based formulation generally in use – the time resolution is at the sampling rate. For instance, there is no "anticipation" of a gain change such as is inherent in block-based processing. The channel gain algorithm can be set to fast-attack down to single sample resolution (there are other reasons why this might not generally be a good idea, but it is, at least, possible by use of a running transform).

## About Compute Requirements

Some people will surely complain about the amount of compute power involved in the above iterations. For this, I have two answers. First, the amount of compute power required is trivial when run on a modern GPU. Many channels of running transforms can easily be computed in real time, even with large amounts of padding. Next, every time I have been appalled by the amount of compute time a particular technique takes, a few years later I am running multiple channels of it in real time. Nothing demonstrates this more clearly than some of my experiments with concert-hall reverberation simulation [**JAM2, JAM3**]. A calculation that took 10 hours of computer time per second of monaural audio in 1977 ran two channels in real time in 2000. I am now running more than 20 channels of the same algorithm in real time on a tablet computer. As the compute power becomes available, it seems important to have some algorithms in the portfolio ready to take advantage of new platforms as they arise.

I might note, however, that the real problem with implementing running transforms is where to put the resulting data. It explodes the amount of data by a factor of $M$. It is suitable for real-time processing where we can discard the data after use, but storing the transform data will remain problematic for some time.

## Numerical Issues

As noted as early as Gold and Rader [**GR**], the iteration of equation (3) and its descendants is unstable. Roundoff error accumulates exponentially. One suggestion was to stabilize the calculation by multiplying the state by a number close to one, such as 0.999. This does stabilize the calculation, but it also means that the calculation of the discrete Fourier transform is not exact. That is, it never compares exactly to the block transform of the same points. As it turns out, there is a simple feedback scheme that absolutely stabilizes the iteration. This can be derived by noting that we can form an approximation to $x(n-N)$ by extracting that point from $X_k(n-1)$ by a time-shifted direct sum as described above. That is,

$$(37)\quad \hat{x}(n-N) = \frac{1}{M}\sum_{k=0}^{M-1} X_k(n-1)(W_M)^{-(N-1)k}$$

In the absence of roundoff error in the arithmetic, equation (37) is *exact.* In the presence of roundoff error, this value of $\hat{x}(n-N)$ will be complex, even with real-valued input. We know that $X_k(n-1)$ contains information from the input points $\mathrm{x}(n-1{:}\,n-N)$. The factor $(W_M)^{-(N-1)k}$ shifts the point at $n-N$ to the first position. The direct sum then produces an approximation to the input point $x(n-N)$. This can be viewed as the dot product of the transform vector ($X_k$) and a complex harmonic vector ($(W_M)^{-(N-1)k}$). A complex harmonic vector, among other things, has the property that every component has magnitude 1. We may then use this calculated value in place of the actual input point $(n-N)$ in equation (3). This creates a negative feedback loop. A number of authors have noted that this absolutely stabilizes the calculation and reduces the error to a very, very low level **[KK, JL, SSP, KPT**].

.

This has been explained in a number of ways. I will provide a definitive proof of the stability. I am sure this proof has been discovered a number of times, but I have not seen a reference to it in the literature, so just to make it explicit, let me repeat it here.

Start with equation (3) but substitute $\hat{x}(n-N)$ for $x(n-N)$.

$$(38)\quad \tilde{X}_k(n) = x(n) + \tilde{X}_k(n-1)(W_M)^k - \hat{x}(n-N)(W_M)^{kN}$$

Now substitute the direct sum for $\hat{x}(n-N)$:

$$(39)\quad \tilde{X}_k(n) = x(n) + \tilde{X}_k(n-1)(W_M)^k - (W_M)^{kN}\frac{1}{M}\sum_{p=0}^{M-1}\tilde{X}_p(n-1)(W_M)^{-(N-1)p}$$

We may formulate this as a $Z$-transform by replacing the time delay with the delay operator $Z^{-1}$ and dropping the time indices.

$$(40)\quad \tilde{X}_k = x + Z^{-1}\tilde{X}_k(W_M)^k - Z^{-1}(W_M)^{kN}\frac{1}{M}\sum_{p=0}^{M-1}\tilde{X}_p(W_M)^{-(N-1)p}$$

We can extract the term in the summation where $p = k$.

$$(41)\quad \tilde{X}_k = x + Z^{-1}\tilde{X}_k(W_M)^k - Z^{-1}(W_M)^{kN}\frac{1}{M}\tilde{X}_k(W_M)^{-(N-1)k} - Z^{-1}(W_M)^{kN}\frac{1}{M}\sum_{p\neq k}\tilde{X}_p(W_M)^{-(N-1)p}$$

Note that $(W_M)^{kN}(W_M)^{-(N-1)k}$ collapses to just $(W_M)^k$

$$(42)\quad \tilde{X}_k - Z^{-1}\left(1-\frac{1}{M}\right)\tilde{X}_k(W_M)^k$$

$$= x - Z^{-1}(W_M)^{kN}\frac{1}{M}\sum_{p\neq k}\tilde{X}_p(W_M)^{-(N-1)p}$$

This is recognized as a first-order difference equation in $\tilde{X}_k$. The magnitude of the feedback coefficient is $(M-1)/M$. The iteration of equations (33), or equivalently equation (34), is thus unconditionally stable for each value of *k*. Thus the overall iteration is unconditionally stable.

One might argue that energy from the other channels, represented by the summation in equation (42), could contribute destabilizing energy to $\tilde{X}_k$. There is cross-coupling of energy among all the channels. We can ask if other channels can excite channel *k* in a manner that would create instability. Equation (19) shows that all the channels other than channel *k* have a zero of transmission at the frequency of band *k*. That is, the summation in equation (42) *cannot* pass (steady-state) energy at the frequency of band *k* because of the zero of transmission.

We can derive the equivalent relation for the frequency-shifted transform. We start with equation (24):

(43) $$\hat{X}_k(n) = x(n) + \hat{X}_k(n-1)(W_M)^{k+\beta} - x(n-N)(W_M)^{N(k+\beta)}$$

We substitute the time-shifted form from equation (36) to get the following:

(44) $$\hat{X}_k(n) = x(n) + \hat{X}_k(n-1)(W_M)^{k+\beta} - \frac{1}{M}(W_M)^{Nk+\beta}\sum_{p=0}^{M-1}(W_M)^{-(N-1)p}\,\hat{X}_p(n-1)$$

We collect the terms involving $\hat{X}_k$ and move them to the LHS, leaving everything else on the RHS to get this:

(45) $$\hat{X}_k(n) - (1-\tfrac{1}{M})\hat{X}_k(n-1)(W_M)^{k+\beta}$$

$$= x(n) - \frac{1}{M}(W_M)^{Nk+\beta} \sum_{p \neq k} (W_M)^{-(N-1)p}\, \hat{X}_p(n-1)$$

Again, the feedback coefficient for each value of *k* is less than one. Thus the recursion is absolutely stable with or without frequency-shift.

I find this result most remarkable. Do recall that a frequency offset of ½ bin changes the sign of the $x(n-N)$ term from negative to positive. One would expect that would lead to noise accumulation, no matter how $x(n-N)$ is computed. It does not. The recursion is stable with noise at a very low level. Most remarkable.

One interesting side-effect of replacing $x(n-N)$ with a direct sum is that we no longer need an explicit *N*-point delay to produce $x(n-N)$, We recover $\hat{x}(n-N)$ directly from the state $\hat{X}_k(n-1)$.

**Roundoff Noise**

We should say a few words about roundoff error. Equation (44) has a large summation. The arithmetic implied by this summation will have roundoff error. Without roundoff error, the iteration is unconditionally stable. In the absence of error, Equation (37) computes exactly x(n-N). It is a dot product. By definition, the dot product has no "state" and no internal feedback. It cannot "blow up" exponentially by itself. Moreover, the weight vector $(W_M)^{-(N-1)k}$ has elements that are all magnitude one. The magnitude of the dot product can not exceed $\frac{1}{M}\sum_{p=0}^{M-1}\left|\hat{X}_p(n-1)\right|$. This is recognized as the *average* of the magnitudes of the elements of the calculated DFT vector.

If we include roundoff error, we can model the error as an *additive independent random process* that is included as a driving function (along with x(n)) for the iteration of Equation (44). As with any part of the driving function, each new error sample will excite the impulse response of the iteration.

Equation (44) describes a multi-band FIR filter. The impulse response of each band is exactly M samples in length. In the absence of error, the impulse response will be zero after N samples. When error is included, we can understand that M-1 samples of pure error signal will be applied to the iteration after the impulse through the feedback path. This, of course will provoke M more samples of error (second-order error, if you will). Since the error in any practical implementation is much smaller than the original signal (or impulse), we can expect the impulse response to consist of an impulse followed by an error signal that "steps down" in amplitude every M samples.

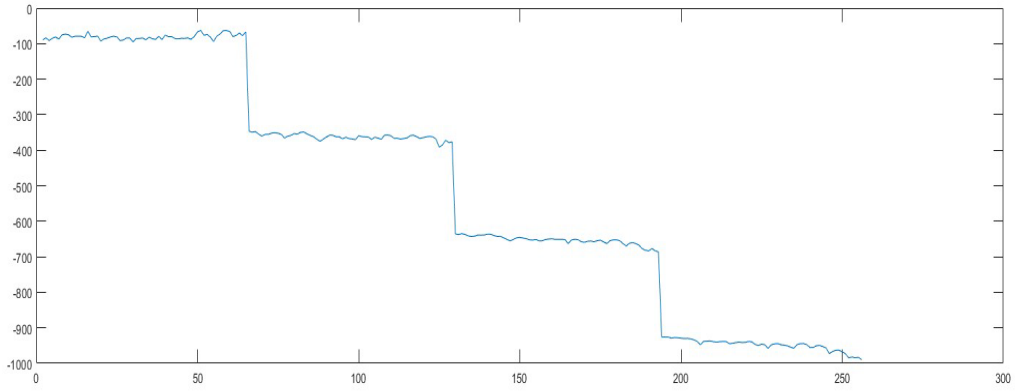


**Figure 5** – Error output energy from equation (44) using single impulse as input with M=64 and N=16.

Every input sample will generate a curve with output energy like that shown in Figure 5. To calculate the overall response, we have to note that the error response for each sample will itself generate more roundoff error, but at an increasingly lower level. The plot in Figure 5 shows the roundoff error for an isolated impulse. This shows how the influence of the error always decays and does not "explode" or become unstable. In 32-bit floating point, we can expect each step in Figure 5 to be maybe 10^-6, or about 120 dB down. With 64-bit floating point, each step would be maybe 10^-15, or about 300 dB down (see Figure 5). Roundoff error, even when accumulated over all time, can not be a factor in

stability. Given the decay, the total accumulated roundoff error will converge to a stable, hopefully small value compared to the signal.

Let me emphasize the main point – the noise error produced by the algorithm is *added* into the input signal and decays with time in exactly the same manner as any other input signal. Equation (45) shows that the iteration is stable *for any input signal*. Figure 5 shows that the roundoff error decays away by itself and cannot cause any instability in the algorithm.

One might ask why the impulse response in the absence of noise is of length $N$, whereas the impulse response in the presence of roundoff noise is of length $M$. The answer is that the roundoff noise can be considered to be uncorrelated with its environment. After the application of an impulse, some amount of noise will be present in all components of the running transform. The noise will continue to propagate to subsequent samples by the multiplication by a complex harmonic vector followed by summation into the state. The noise in the feedback loop can be considered to be independent of anything that has come before. That is, the process of generating and propagating the roundoff noise destroys information about where the $M$ samples of noise came from. If we mentally take the inverse transform of the state of the iteration, we will get a sequency of $M$ random numbers that would exactly generate that state. If we feed the iteration with zeros and run the iteration followed by the direct sum, we will get exactly those $M$ random numbers out that would have produced that state.

Having made the above argument for absolute stability of the iteration of Equation (44), we should also point out where this argument breaks down: *the relative error increases with M.* That is, if you increase $M$, at some point the error will swamp the signal. The iteration is always stable, but the error in the iteration or in the signal itself may ultimately become unacceptable. For a given level of arithmetic precision, there

will be some value of *M* where the error accumulation will become unacceptable. A rule-of-thumb might be that the maximum value of *M* would be the ratio between the accuracy the user requires of the result over the error accumulation implied by Equation (44), which would be roughly M times the roundoff error of the resonator array. Clearly, with 32-bit floating point, this limit on *M* will have to be carefully considered. With 64-bit floating point, considerable freedom is available for the choice of *M*. Figure 6 shows the absolute error of the running transform followed by a direct sum for different values of *M*.

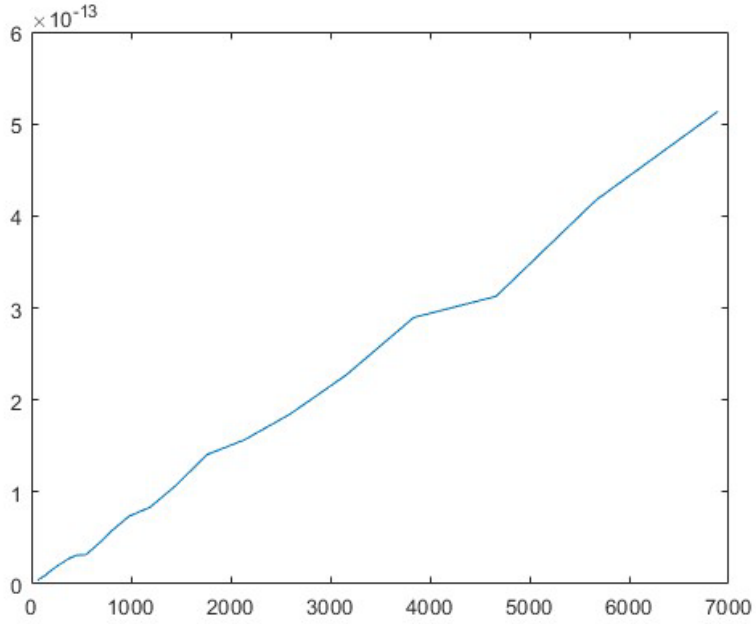


**Figure 6** – RMS error in output of equation (39) using unit white noise as input for various values of M. This line can be approximated by (0.73e-16)M.

This error curve has a value of roughly 0.73e-16 for each additional order. This is about what you would expect for summing M random values using IEEE double-precision arithmetic. The amplitude of the noise equals the amplitude of the signal at a bit over 10^16. Needless to say, using shorter precision arithmetic, such as 32-bit IEEE floating

point, leads to even more compressed dynamic range. M is probably limited to 256 or less for high-quality audio applications in 32-bit floating point.

**Why Does Feedback Work?**

One might wonder why replacing $x(n-N)$ with $\hat{x}(n-N)$ in equation (37) has such a profound effect. It is because of the form of the error in each resonator state. Since each resonator has infinite Q, the roundoff error is spectrally-shaped. Energy is concentrated at the resonant frequency of each resonator. The form of the error is that of a noisy sinusoid at the resonator frequency. An example of this is shown in Figure 7.

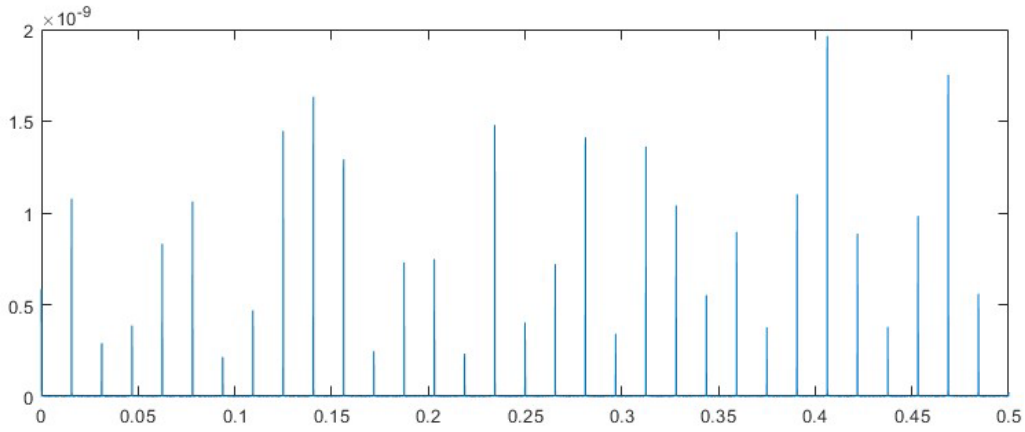


**Figure 7** – Spectrum of error in output of equation (3) using white noise as input with M=64 and N=16.

Subtracting $\hat{x}(n-N)$ has the effect of subtracting off the *actual* noisy sinusoid present in each point of the state, thus cancelling out the sinusoidal component of the state. Other non-sinusoidal noise will be largely unaffected as shown in Figure 8.

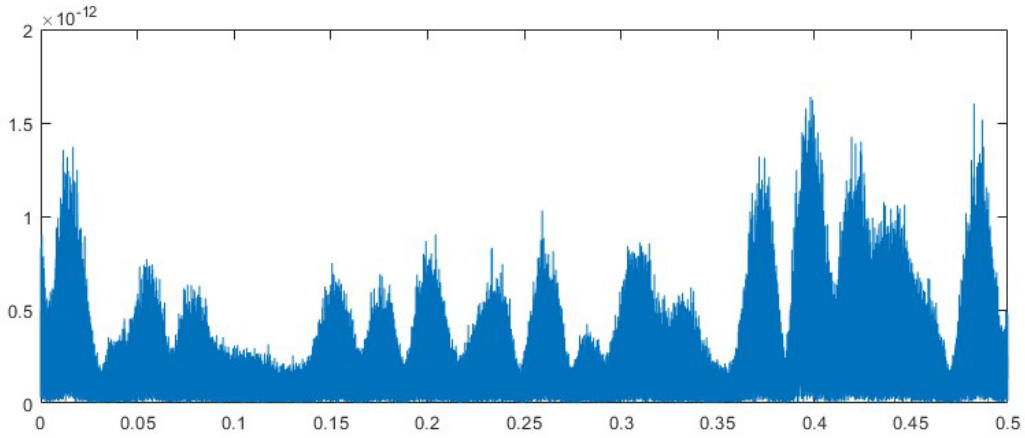


**Figure 8** – Spectrum of error in output of equation (39) using white noise as input with M=64 and N=16.

Note that the sinusoidal component is completely cancelled, leaving a noise floor several orders of magnitude lower than the error without feedback. In this test, the noise was lowered more than 60 dB. The cancellation of the sinusoidal components of the roundoff error is the principal stabilizing influence of the feedback. Without feedback, the error increases without limit, and the form of the error is a noisy complex harmonic series.

## Real-Valued Data

It is well known that there is redundancy in the transform when the input is restricted to real-valued data. The upper half of the transform will simply the complex conjugate of the lower half of the transform (in reverse frequency order). With no frequency shift ($\beta = 0$), there will be $\frac{M}{2} + 1$ independent components, with components at 0 and $M/2$ both being real. With a frequency shift of $\beta = 1/2$, there will be $M$ independent (complex) components. We might ask if it is possible to compute only the first $\frac{M}{2} + 1$ (or $M/2$) components in equation (3)?

Indeed, in the absence of error, this does give the correct answer for real-valued input data.

We might ask what happens when we try to use feedback, as in equation (37)? It still works, but the formula must be modified. The general form of $\hat{x}(n-N)$ can be written as follows:

(46) $\hat{x}(n-N)$

$$= Re\left(\frac{1}{M}(W_M)^{Nk+\beta}\sum_{p=0}^{L} c(p)\,(W_M)^{-(N-1)p}\,\hat{X}_p(n-1)\right)$$

The upper-bound, *L,* and the coefficient $c(p)$ need a bit of explanation. There are four different cases corresponding to two choices of shift and of *M* odd or even. These may be summarized in the following table, taking $c(p) = 2$ where not specified.

| | $\boldsymbol{\beta = 0}$ | $\boldsymbol{\beta = 1/2}$ |
|---|---|---|
| *M* EVEN | $L = \frac{M}{2}$ <br> $c(0) = c(L) = 1$ | $L = \frac{M}{2} - 1$ <br> $c(p) = 2$ |
| *M* ODD | $L = \frac{M-1}{2}$ <br> $c(0) = 1$ | $L = \frac{M-1}{2}$ <br> $c(L) = 1$ |

**Table 1** – Details of limit *L* and coefficient $c(p)$ in equation (46).

These limits are a bit clumsy, but they represent the fold-down of the upper half of the transform.

This value of $\hat{x}(n-N)$ is real by construction. As noted before, it does contain information about the sinusoidal component of the state, and can cancel out the that sinusoidal error component, but only directly in the real part of the state. It influences the imaginary part of the state only indirectly by the complex multiplication that advances the state by one sample. The result is that equation (46) exhibits a higher average error than equation (37). Some informal experiments suggest that the noise floor can be 30 dB higher when taking advantage of the real-valued input. This makes for an interesting tradeoff for the user.

**Notes on Implementation**

The calculation for a new incoming sample starts with *N* complex resonators. The resonator for a particular channel is calculated entirely locally, without reference to other channels. This means that the entire row of N resonators may be performed entirely in parallel. After that, there is, of course, a direct sum across the outputs to provide the feedback term for the next sample. This channel-to-channel independence on each sample makes this algorithm ideal for GPU implementation. The number of channels in this implementation is only limited by the local memory on the GPU. Do note that since the resonators are complex, it takes two GPU shaders to compute one channel for the real and imaginary parts of the output value. There will be some redundant calculation between the two shaders due to the GPU architecture. GPU implementation even on the smallest device on the market today will be many times faster than CPU implementations. The astute reader will recall that the GPU implementation of double-precision floating point is generally 9 or 10 times slower than single-precision.

In the section on roundoff error, we note that the error accumulation increases with *N*. This is ultimately the limit on the viability of running transforms. A simple and easy-to-program way to extend floating-point precision was described by Dekker in 1971 [**DEKK**]. These are

somewhat informally called “Dekker doubles”. One doubling of an IEEE double-precision floating point number gives a huge factor of reduction in the accumulated noise. Note that the Dekker doubling algorithm can be recursively applied to allow for arbitrary precision floating-point arithmetic. A single step of the resonator using Dekker doubles does require a few dozen floating-point operations, so the compute time does rise sharply with each doubling. Note that the calculation for the Dekker double can again be pipelined, since it involves no conditionals – just a sequence of multiplies and adds. This means that a GPU implementation using Dekker doubles would not only be quite fast, but could be highly accurate as well.

**References:**


[**GR**] Gold and Rader “*Digital Processing of Signals*” McGraw-Hill, 1969

[**JAM1]** Moorer “*Algorithm Design for Real-Time Audio Signal Processing”* ICASSP Conference Proceedings, San Diego, 1984

[**JAM2**] Moorer “*About This Reverberation Business”* Computer Music Journal, Volume 3, Number 2, June 1979

[**JAM3**] Moorer “*Audio in the New Millennium”* AES 108th Convention Heyser Lecture http://www.aes.org/technical/heyser/aes108.cfm

[**JAM4**] Moorer “*A Note on the Implementation of Audio Processing by Short-Term Fourier Transform”* 2017 IEEE Workshop on Applications of Signal Processing in Audio (WASPAA

[**JOS**] Julius Orion Smith "*Bark and ERB Bilinear Transforms"* https://ccrma.stanford.edu/~jos/bbt/

[**MB**] Moorer and Berger "*Linear Phase Bandsplitting*" AES 76th Conference, New York, 1984

[**RV**] Rife and Vincent "*Use of the Discrete Fourier Transform in the Measurement of Frequencies and Levels of Tones*" Bell System Technical Journal, February 1970, pp197-228

[**TS**] Thomas G. Stockham, Jr., "High-Speed Convolution and Correlation", *1966 Spring Joint Computer Conference, AFIPS Conference Proceedings*, Volume 28, pp 229-233

[**DEKK**] T.J. Dekker "*A Floating-Point Technique for Extending the Available Precision*" Numerische Mathematica 18, 224-242, 1971

[**KK**] Kovács, Kollár "*Software Implementation of the Recursive Discrete Fourier Transform"* 27th International Conference Radioelektronika 2017

[**JL**] Eric Jacobsen, Richard Lyons "*The Sliding DFT"* IEEE Signal Processing Magazine, March 2003 pp74-80

[**SSP**] László Sujbert, Gyula Simon, Gábor Peceli "*An Observer-Based Adaptive Fourier Analysis*" IEEE Signal Processing Magazine, July 2020 pp134-143

[**KPT**] Zsalt Kollár, Ferenc Plesznik, Simon Trumpf „*Observer-Based Recursive Sliding Discrete Fourier Transform*" IEEE Signal Processing Magazine, November 2018, pp 101-106

## Appendix 1 – Response to Sinusoidal Input

This is equation (18) above:

$$\text{(A1)} \qquad X_k(n) = e^{jn\theta}\left(\frac{1-e^{j(2\pi\frac{k}{\alpha}-N\theta)}}{1-e^{j(2\pi\frac{k}{M}-\theta)}}\right)$$

Let $\theta$ be an offset from the center of a band. That is, $\theta = 2\pi\frac{k+\delta}{M}$. This reduces to the following:

$$\text{(A2)} \qquad X_k(n) = e^{2\pi j\frac{\delta}{2M}(N-1)}\frac{\cos\left(2\pi{}^{\delta}/_{2\alpha}\right)}{\cos\left(2\pi{}^{\delta}/_{2M}\right)}$$

The right side is a polar decomposition consisting of a phase term of unit magnitude and a real amplitude term. It shows that the phase change is *linear* in $\delta$ as the frequency of the input deviates from the center of the band ($\delta = 0$).