ABOUT RUNNING TRANSFORMS

James A. Moorer 2016

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, $0 \le k < M$
N	Number of input points per frame
M	Transform size ($M \ge N$)

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

"direct sum" property[1]. 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. 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:

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

This implements the common window functions, such as Hamming, Hanning, and Blackman. Additionally, Rife and Vincent [**RV**] defined

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

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.

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 (11) 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 α 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 α.

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.

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 Hanning 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.

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:

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

The response can be written as follows:

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

The input sinusoid will be scaled by a complex transfer gain that depends on *M*, *N*, *k*, and θ. This differs from the usual formula by the inclusion of zero-padding.

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:

$$(19)\quad \tilde{X}_k(n) = \sum_{m=-\sigma}^{N-\sigma-1} x(n-m)e^{2\pi jmk/M}$$

It follows then that the relation between equations (19) and (1) is as follows:

$$(20)\quad \tilde{X}_k(n) = e^{-2\pi\sigma \mathrm{j}k/M}X_k(n) = (W_M)^{-\sigma k}X_k(n)$$

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 (15) and (16), but this constrains the choice of *M* and *N*. We must set α in equation (14) to an *odd* integer to apply a window.

Note that equation (20) 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:

$$(21)\quad \begin{aligned}\tilde{X}_k(n) = {} & \tilde{X}_k(n-1)(W_M)^k \\ & +(W_M)^{-\sigma k}(x(n) - x(n-N)(W_M)^{kN})\end{aligned}$$

This iteration yields a shifted running transform. It does preserve the direct sum property. The direct sum selects the sample that was originally at $m = \sigma$.

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:

$$(22)\quad \hat{X}_k(n) = \sum_{m=0}^{N-1} x(n-m)e^{2\pi jm(k+\beta)/M}$$

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

$$(23)\quad \hat{X}_k(n) = x(n) + \hat{X}_k(n-1)(W_M)^{k+\beta} - x(n-N)(W_M)^{N(k+\beta)}$$

Curiously enough, the direct sum property is preserved for any value of β. Note that if β 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.

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 1 shows a comparison of these two designs plus one derived by using the Blackman window.

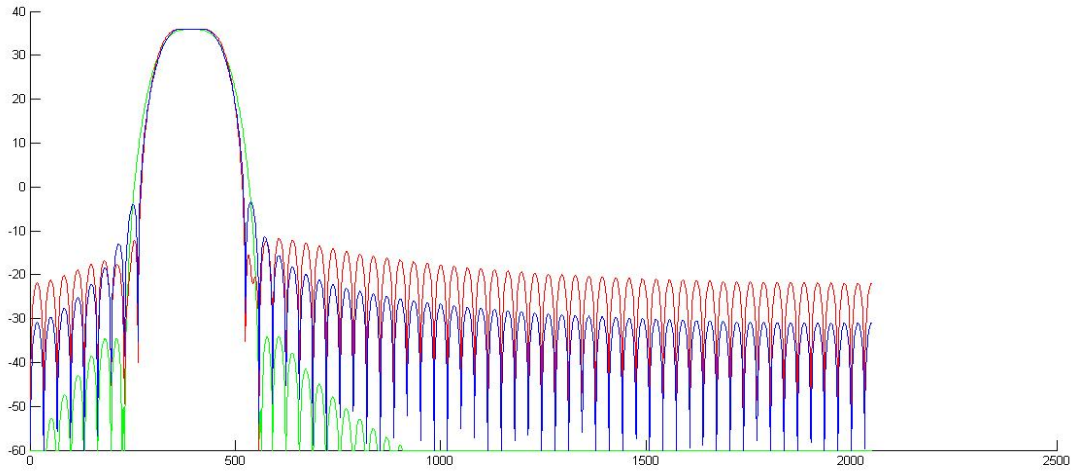


Figure 1 – Comparison of frequency-sampling filters. Blue is original Gold and Rader [GR] design. Red is design using Hamming window. Green is using Blackman window.

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 (15) and (16).

Note also that any frequency offset as defined in equation (22) must be taken into account when designing the filter in the frequency domain.

An interesting result of this formulation is that if we product 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-in to 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 Hanning 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.

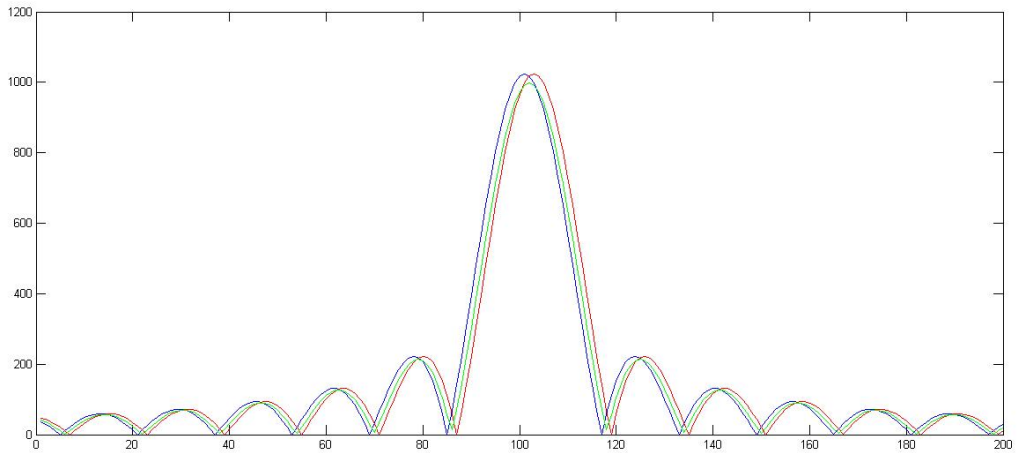
One issue with combining channels this way is that the band edge frequencies, center frequencies, and band widths are quantized by N. In this case we can pad the transform by M-N zeros. We can increase the number of channels by the factor of α (equation (14)). These additional channels will be redundant, 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 (22) and (23). 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* β. 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 $\beta = 1/2$[2]. 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 2 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.



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

Figure 2 – 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 3 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.

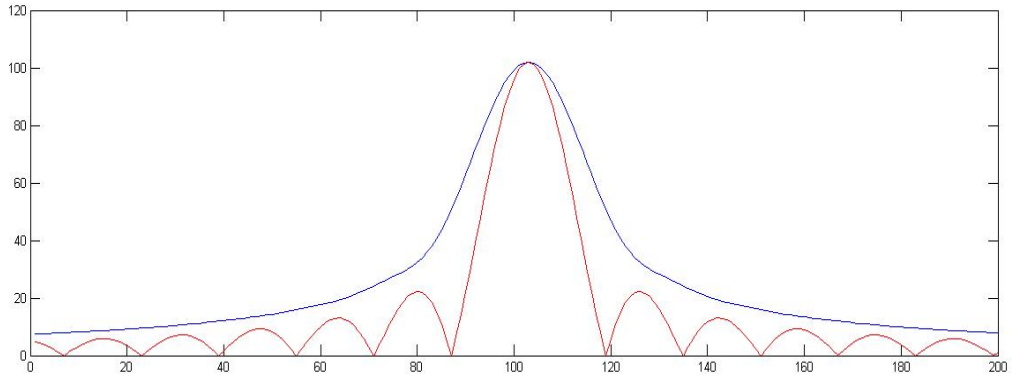


Figure 3 – Comparison of the unmodified magnitude response of a half-integer channel filter (red) and a combination of this channel filter minus ½ of the adjacent integer channel filters.

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 pad 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.

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

[**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