Digital Filtering in the Frequency Domain

Document: 91
www.signal-processing.net
andreas_dsp@hotmail.com
Author: Andreas Schwarzinger
Date: November 12, 2019

Introduction

In reference one [1] we have discussed time-domain implementations of FIR filters, whose actions are based on and can be analyzed via the principle of convolution. We have further covered several techniques that provide us with proper time-domain filter coefficients. The three principle techniques that we have used in the past are the frequency sampling, the equiripple and the least-squares methods. However, with the advent of software-based radios, the need has arisen to find computationally more efficient techniques to perform filtering. As the following figure reminds us, time-domain filtering requires *N* potentially complex multiplication, where *N* is the number of coefficients, and a similar number of addition steps every time a new sample arrives at the next clock edge. Imagine a filter with *N* = 51 coefficients processing an input sequence $x[n]$ featuring 2048 samples. The complex multiplications alone number 2048·51 = 104448.

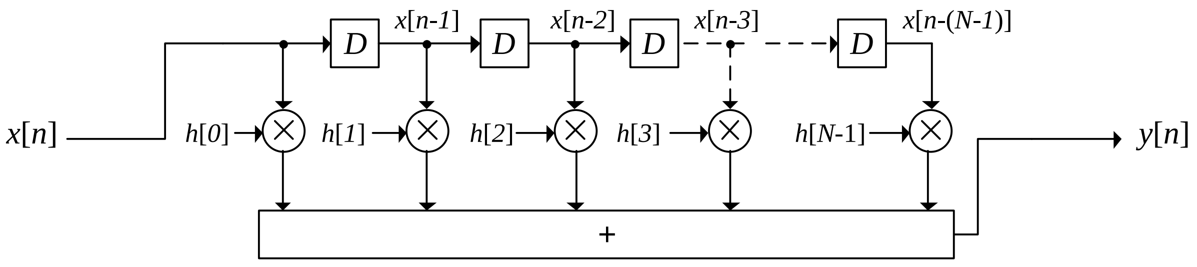

Figure 1: FIR Filter Structure Implemented as a Transversal Filter with Coefficient Vector H = [h[0], h[1], … h[N-1]]

Whereas this fact may not be a major problem for hardware implementations in FPGAs or digital ASICs, it is not the preferred method for software implementations. In this document, we will introduce the frequency domain method of digital filtering using I/FFT operations, which is far more efficient in software implementations of digital filtering.

Convolution (Digital Filtering) in the Time Domain

Convolution is the mathematical operation that computes the time domain output waveform, $y[n]$, of a LTI (linear time-invariant) system given an arbitrary time domain input signal, $x[n]$, and the impulse response, $h[n]$, of the system. LTI systems in the realm of discrete math are also called LSI (linear sample-invariant) systems.

At any one time instance *n*, the value $x[n]$ will reside at the input of our model, whereas $y[n]$ will reside at its output. Those samples that have appeared at the input in the past ($x[n-1]$, $x[n-2]$ …) have already traveled down the delay line and are multiplied by the model's coefficient vector $h = [\ h[0], h[1] \ldots h[N-1]\ ]$. As is evident in the figure above, $y[n]$ is calculated as follows.

$$y[n] = h[0]\cdot x[n] + h[1]\cdot x[n-1] + \cdots + h[N-1]\cdot x[n-(N-1)]$$

The simple expression above points us to the first formulation of the convolution operation for an LTI system with *N* model coefficients.

$$y[n] = x[n] \otimes h[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

As a reminder from your 'Signals and Systems' class in college, let us present the continuous time version of the convolution operation.

$$y(t) = x(t) \otimes h(t) = \int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau) d\tau \qquad (1)$$

You may also remember that the frequency content, *Y(f)*, of the filtered output signal, *y*(*t*), was simply the product of the Fourier transforms of the input signal *x*(*t*) and the filters impulse response *h*(*t*). The appendix shows the derivation of the formula below.

$$Y(f) = X(f) \cdot H(f) \qquad (2)$$

<u>Digital Filtering in the Frequency Domain</u>

In discrete time, the result above is suggesting the following,

$$Y[m] = DFT\,(x[n]) \cdot DFT\,(h[n]) = X[m] \cdot H[m]$$

and more importantly,

$$y[n] = IDFT\,(\,DFT\,(x[n]) \cdot DFT\,(h[n])\,)$$

$$y[n] = IDFT\,(\,DFT\,(x[n]) \cdot H[m]\,) \qquad (3)$$

The equations above lead us to a straight forward, but somewhat premature conclusion, which indicates the following course of action.

→ Take the DFT of the input signal *x*[*n*] to produce *X*[*m*].

→ Zero-pad the coefficient vector, *h*[*n*], until its length is equal to that of *x*[*n*], and then take the DFT to produce *H*[*m*].

→ Take the IDFT of product of *X*[*m*] and *H*[*m*] to arrive at the filtered signal, *y*[*n*].

This approach is basically correct, but there are a few alterations that we have to consider.

1. The input sequence, *x*[*n*], may be very long thus forcing us to break it up into convenient sections of a size that will allows us to compute the FFT. How to break up the sequence is not obvious and we will illustrate it in the next section.
2. There is no need to synthesize a coefficient sequence, *h*[*n*]. Whether we operate in the time or frequency domain, the goal is always to create a filter frequency response that suits our needs, and this response is always defined in the frequency domain. For time domain filters, we are stuck finding some way to get from that desired frequency response back to *h*[*n*], but as we are filtering in the frequency domain, this is no longer needed. We simply set *H*[*m*] to whatever we want. Thus, we only need to take the FFT of the current section of *x*[*n*] and the IFFT of the product of *X*[*m*] and *H*[*m*].

<u>Method 1</u>

The following figure illustrates a flawed method of dividing the input sequence, *x*[*n*], into equally long sections before computing the frequency domain filtering process.

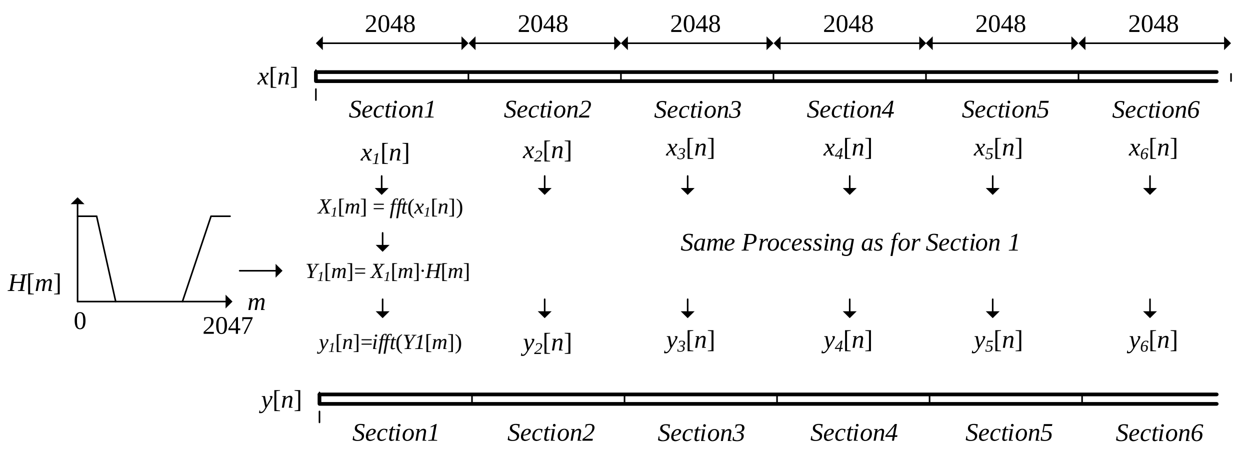

Figure 2: First Method of Generating Subsections of the Input Sequence, x[n]

The method shown above divides the input sequence, $x[n]$, into sections each featuring 2048 samples ready to be processed by the FFT. We would simply take the FFT of each section, multiply the result by $H[m]$ and then take the IFFT to get to the output sections that when concatenated yield $y[n]$. The problem with this approach is that it will produce discontinuities at the boundaries of the output sections. In part *a*) of the figure below, observe the real continuous waveform, $x[n]$, as it progresses from section 1 through section 3. Once we isolate section 2 to form a vector $x_2[n]$, we lose the knowledge of the waveform $x[n]$ as it existed at the end of section 1 and beginning of section 3. When we take the FFT of $x_2[n]$, the frequency information of section 2 represents a periodic waveform that repeats every 2048 samples and will look like part *b*) of the figure below. Notice how the samples at index 2048 and up are equal to those at 0 and above. Likewise the samples approaching index 2047 are the same as those samples below n = 0. If we multiply $X_2[m]$ and $X_3[m]$ by $H[m]$ and then take the IFFT to move back into the time domain, there is no guarantee that the section $y_2[n]$ and $y_3[n]$ will connect continuously at the section boundary.

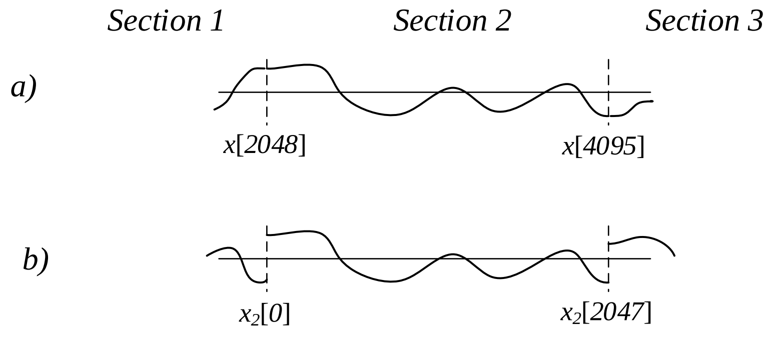

Figure 3: The Periodic Nature of a Section Once Isolated and Transformed by the FFT

In a time-domain filter implementation, the situation is very different, as filtering any sample in $x[n]$ requires that both past samples ($x[n\text{-}1]$, $x[n\text{-}2]$, ..) and future samples ($x[n+1]$, $x[n+2]$, …) are accessible in the FIR filters shift register.

Method 2

The next figures illustrate the proper way of solving the discontinuity problem associated with frequency domain filtering. This method still divides the input waveform $x[n]$ into sections of length 2048, but this time there are overlapping regions of length 256 samples. Sections $x_1[n]$, $x_2[n]$, and $x_3[n]$ span samples $x[0 \dots 2047]$, $x[1792 \dots 3839]$, $x[3584 \dots 5631]$ and so forth. We multiply a mask onto each section to smoothly force the sample values at the start and end of each section to zero. There are different possibilities for this mask and we show just two of them below.

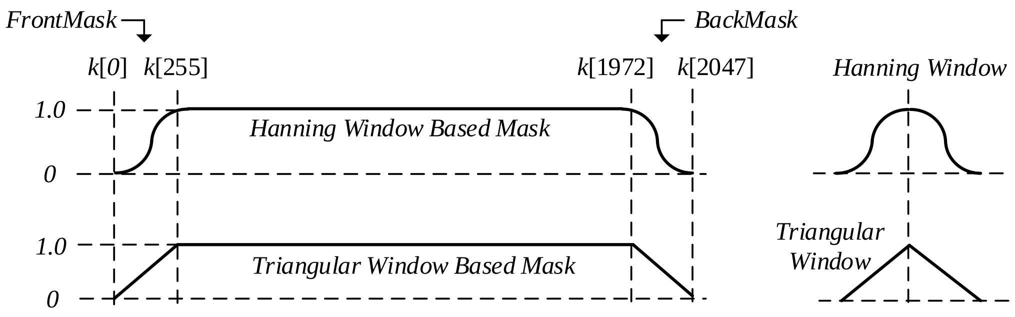

Figure 3: Possible Masks Applied to Each Section

The total mask, $k[0]$ through $k[2047]$, consists of a front and back portion as well as the center section whose samples are always equal to 1.0. If you add the front and back portions of the mask, all samples must also add to 1.0. The front and back masks can be the first and seconds halves of a *Hanning* or triangular window as shown in the figure above. The equation for the Hanning window is provided below. The total length of the window would be N = 512, which splits up into two halves each with a length of 256 samples (see figure 7 at the end of this document).

$$Hanning[n] = 0.5 - 0.5\cos(\frac{2\pi(n+1)}{N+1})$$

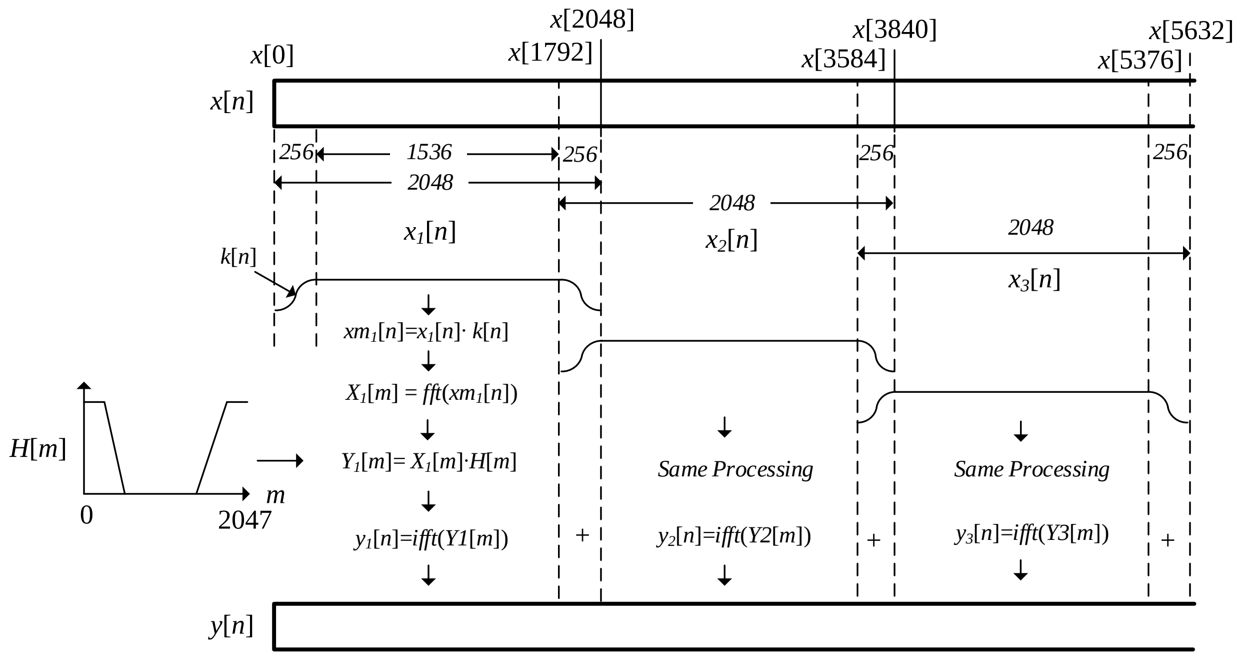

Figure 4: Second Method of Generating Subsections of the Input Sequence, x[n]

The figure above illustrates the processing steps for sections $x_1[n]$, $x_2[n]$, and $x_3[n]$, which produce output sections $y_1[n]$, $y_2[n]$, and $y_3[n]$. Take care to overlap these output sections as well to yield the final output waveform $y[n]$.

Example

In the following example, we will write MatLab code that executes both methods of frequency domain filtering. The sample rate of the signal is set to Fs = 1 MHz, and the input signal, $x[n]$, and filter magnitude response are defined as follows. Note further, that in this example the sections are 4096 samples long, and the filter will eliminate the tone at 75KHz.

$$x[n] = \cos\left(\frac{2\pi \cdot n \cdot 3000}{Fs}\right) + \cos\left(\frac{2\pi \cdot n \cdot 10000}{Fs}\right) + \cos\left(\frac{2\pi \cdot n \cdot 75000}{Fs}\right)$$

$$H(f) = \begin{cases} 1 \ from - 39KHz \ to + 39KHz \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

```
% -----------------------------------------------------------------------------
% 0. Simulation Setup
Fs  = 1e6;                  % Sample Rate in Hz
Len = 4*4096;               % Number of samples in test waveform
n   = 0:Len-1;              % The sample indices
disp(['Nyquist range spans: ' num2str(-Fs/2) ' to ' num2str(Fs/2) ' Hz.']);


% ------------------------------------------------------------------------------
% 1a. Generate input signal 1 -> cos(2*pi*n*F1/Fs)
F1      = 3e3;                   % Frequency of test signal 1
Signal1 = cos(2*pi*n*F1/Fs);     % Signal1

% 1b. Generate input signal 2 -> cos(2*pi*n*F2/Fs)
F2      = 10e3;                  % Frequency of test signal 2
Signal2 = cos(2*pi*n*F2/Fs);     % Signal2

% 1c. Generate input signal 3 -> cos(2*pi*n*F3/Fs)
F3      = 75e3;                  % Frequency of test signal 3
Signal3 = cos(2*pi*n*F3/Fs);     % Signal3

% 1d. The final composite signal x[n]
x_n = Signal1 + Signal2 + Signal3;

% In this exercise we will be filtering away Signal3. To evaluate how well
% the filtering process worked, we synthesize the ideal filtered signal.
x_n_Filtered = Signal1 + Signal2;

% Let's take a look at the composite waveform, x[n]
figure(1);
DisplayRange = 3900:4200;
subplot(3,1,1);
plot(DisplayRange, x_n(1, DisplayRange), 'k'); grid on; hold on;
plot(DisplayRange, x_n_Filtered(1, DisplayRange), 'k.-', 'LineWidth', 2);
title('Input Sequences');
legend('Input Waveform, x[n]', 'Ideal Filtered Waveform');
```

Observe the first subplot on the next page, which illustrates a small section of what would otherwise be a very busy plot. We see a certain number of samples of the original waveform $x[n]$ as well as what we would be expecting the filtered waveform to look like. The sample range was chosen to span the boundary between input sections 1 and 2. Whereas we haven't shown the rest of the code, subplot 2 illustrates the output of filtering method 1 and 2, and the discontinuity at the section boundary around sample 4086 is clearly visible for the filtered waveform produced by method 1. The third subplot illustrates the very small error between the ideally filtered signal, that only consists of sinusoids at 3KHz and 10KHz, and the filtered signal of method 2, thus proving its effectiveness. Clearly the error is tiny compared to the one from method 1.

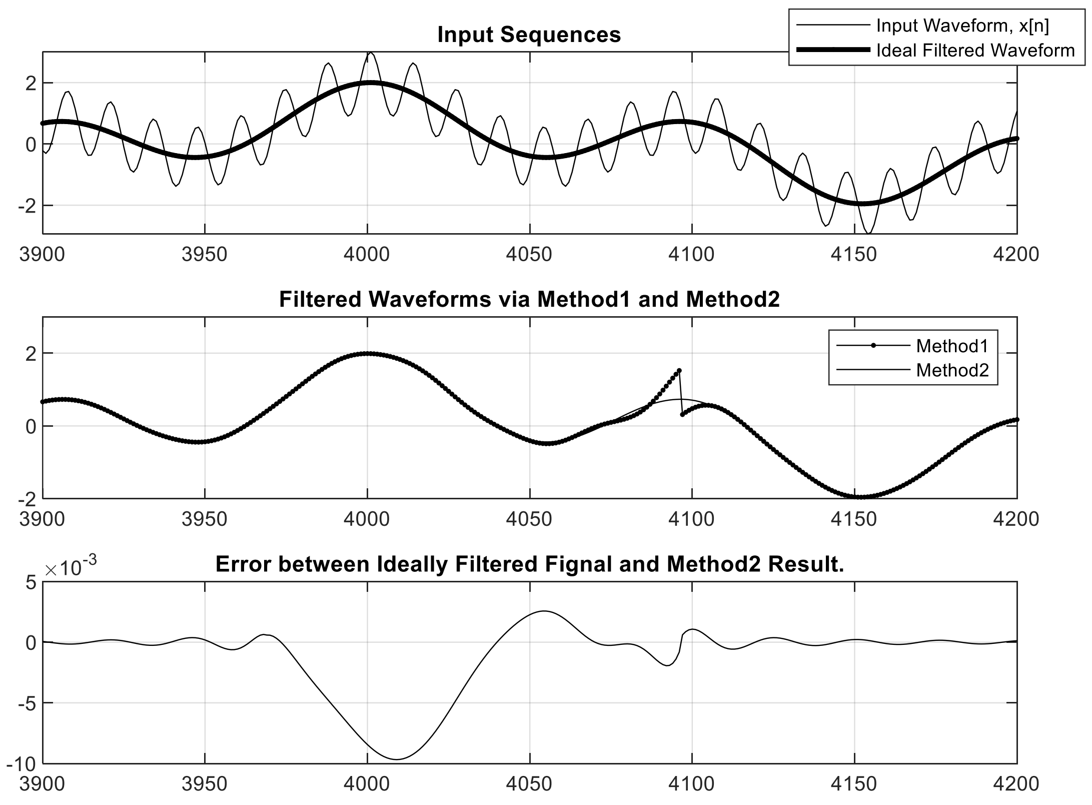

Figure 5: Performance Comparison between Filtering Method 1 and 2

The figure below illustrates the positive portion of the spectrum of the input waveform, *x*[*n*], which we generated in section 1 of the MatLab code. The filter is set up in section 2 of the MatLab code. Note how we define the frequency response, *H*[*m*], for both positive and negative frequencies in the code.

H[0:159] = 1.0 corresponds to frequencies 0 through 159·1MHz/4096 = 0 through 38.818Khz.

H[3937:4095] = 1.0 corresponds to frequencies [-159 to -1] ·1MHz/4096 = -38.818KHz through -244.14Hz.

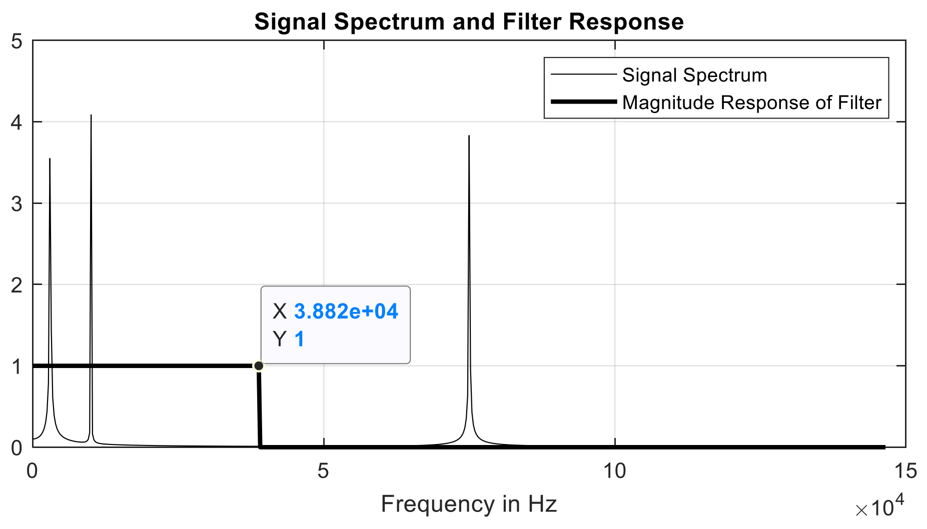

Figure 6: The Input Signal Spectrum as well as the Filter Magnitude Response for Positive Frequencies

```
% ----------------------------------------------------------------------------
% 2. Defining the Filter frequency Response. Note, the frequency response will
% be entirely real valued.
% Our goal is the reject the signal with Frequency F3.
% Remember, that the frequency step is equal to Fs/SectionLength
NumSections        = 4;
SectionLength      = 4096;
FStep              = Fs/SectionLength;
disp(['Frequency Step: ' num2str(FStep)]);

H_m                = zeros(1, SectionLength);
H_m(1,1:160)        = ones(1, 160);   % Define passband for positive frequencies
H_m(1,end-158:end)  = ones(1, 159);   % Define passband for the negative frequencies


% ----------------------------------------------------
% 3a. Method 1: Non-Overlapping sections

y1_n = zeros(1, length(x_n));
for SectionIndex  = 0:(NumSections-1)
   Range          = (1:SectionLength) + SectionIndex*SectionLength;
   Section        = x_n(1, Range);
   DFT_Section    = fft(Section);
   Y_m            = H_m .* DFT_Section;
   y1_n(1, Range) = ifft(Y_m);
   if(SectionIndex == 0)
     figure(2);
     m = 0:600;
     f = m*FStep;
     % The division by 500 is only there so that we can easily compare
     % the signal spectrum and the filter's magnitude response.
     plot(f, abs(DFT_Section(1, m+1))/500, 'k'); grid on; hold on;
     plot(f, abs(H_m(1, m+1)), 'k-', 'LineWidth', 2);
     title('Signal Spectrum and Filter Response');
     xlabel('Frequency in Hz');
     legend('Signal Spectrum', 'Magnitude Response of Filter');
   end
end


% ----------------------------------------------------
% 3b. Method 2: Overlapping Sections

HannLength    =  256;                % 256 samples long - try 512 or 128 if you like
HanningWindow = hann(HannLength)';
FrontMask   = HanningWindow(1, 1:HannLength/2);                % 128 samples long
BackMask    = HanningWindow(1, (1+HannLength/2):HannLength);   % 128 samples long
TotalMask   = [FrontMask, ones(1,SectionLength-HannLength), BackMask]; % 4096 samples
y2_n        = zeros(1, length(x_n));
for SectionIndex  = 0:(NumSections-1)
   Range          = (1:SectionLength) + SectionIndex*(SectionLength - HannLength/2);
   Section        = x_n(1, Range).*TotalMask;
   DFT_Section    = fft(Section);
   Y_m            = DFT_Section .* H_m ;
   OutputSection  = ifft(Y_m);
   y2_n(1, Range) = y2_n(1, Range) + OutputSection;
end

figure(1);
subplot(3,1,2);
plot(DisplayRange, y1_n(1, DisplayRange), 'k.-');  grid on; hold on;
plot(DisplayRange, y2_n(1, DisplayRange), 'k');
axis([3900 4200 -2 3]);
```

```
title('Filtered Waveforms via Method1 and Method2');
legend('Method1', 'Method2');

subplot(3,1,3);
plot(DisplayRange, y2_n(1, DisplayRange) - x_n_Filtered(1, DisplayRange), 'k'); grid
on; hold on;
%plot(DisplayRange, y2_n(1, DisplayRange) - x_n_Filtered(1, DisplayRange), 'k'); grid
on; hold on;
title('Error between Ideally Filtered Fignal and Method2 Result.');


figure(3);
subplot(2,1,1);
stem(FrontMask, 'k'); grid on;
title('The front part of the Hanning Window Magnitude Mask');
subplot(2,1,2);
stem(BackMask, 'k'); grid on;
title('The back part of the Hanning Window Magnitude Mask');
```

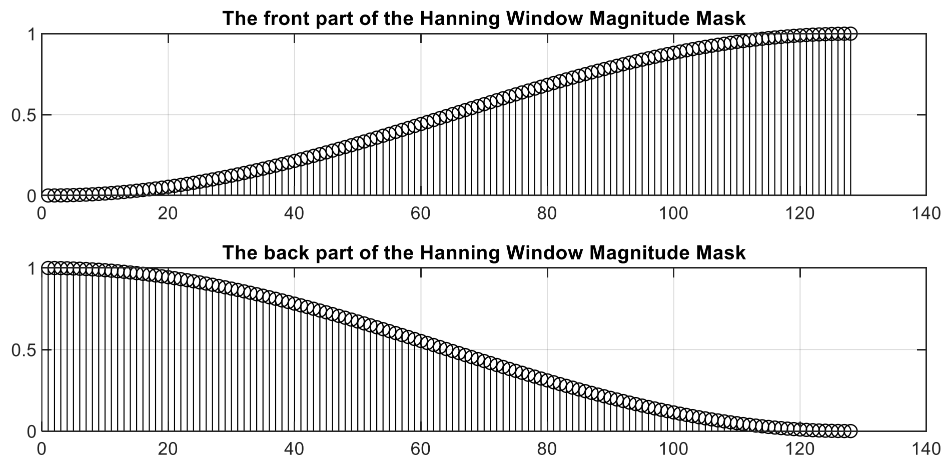

Figure 7: The Front and Back Portion of the Magnitude Mask Applied to Each Input Section

Appendix

For those of you who are a little rusty on the topic of convolution, the process of computing the frequency domain expression of the time domain convolution integral is shown below.

$$y(t) = x(t) \otimes h(t) = \int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau) d\tau$$

However, before we begin, let us first review the equation of the time-domain Fourier transform as well as the time shifting property of the Fourier Transform, as we will need both in the coming derivation. The Fourier transform of a time domain waveform *x*(*t*) is as follows.

$$FT\big(x(t)\big) = X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi f t}\, dt$$

The time shifting property of the Fourier transform is as follows.

$$FT\big(x(t-t_o)\big) = e^{-j2\pi f t_o} \cdot X(f)$$

Let us now take the Fourier transform of the convolution integral.

$$FT(y(t)) = \int_{-\infty}^{-\infty} [\int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau) d\tau] \cdot e^{-j2\pi f t} dt$$

Notice that the expression in the parenthesis below is the Fourier transform of *x*(*t*-*τ*).

$$= \int_{-\infty}^{-\infty} h(\tau) \cdot [\int_{-\infty}^{\infty} x(t-\tau) \cdot e^{-j2\pi f t} dt] d\tau$$

Using the time shifting property of the Fourier transform, the expression reduces to the following.

$$= \int_{-\infty}^{-\infty} h(\tau) \cdot X(f) \cdot e^{-j2\pi f \tau} d\tau$$

As we are not integrating over frequency, we can move the expressions *X*(*f*) to the front.

$$= X(f) \cdot \int_{-\infty}^{-\infty} h(\tau) \cdot e^{-j2\pi f \tau} d\tau$$

The integral that remains is simply the Fourier transform of *h*(*τ*), which we rechristen as *H*(*f*).

$$FT\big(y(t)\big) = X(f) \cdot H(f)$$

References:

[1] Schwarzinger, Andreas O, *Digital Signal Processing in Modern Communication Systems*, Lake Mary, FL 2013, Chapters 2.4, 3.1.3 and 3.2.2.