ELECTRONOTES
1016 Hanshaw Road
Ithaca, NY 14850

APPLICATION NOTE NO. 432

August 3, 2017

SIMPLIFIED DIGITAL NOTCH FILTER DESIGN

Recently [1] we have been involved with an issue of a so-called "Worldwide Hum" which is what seems like an actual sound of very low frequency (perhaps 50 Hz to 100 Hz) that is heard by a small percentage of people all around the world [2], but which seems to strongly evade a technical analysis. Any such low-frequencies (real or apparent) are in the range where the human ear does not hear well, and we have little experience even noticing acoustic energy down there. There are of course the frequencies of the electrical supply lines [60 Hz (US and Canada), 50 Hz (rest of world), along with a few harmonics], and when we do listen down there, we tend to hear these. Hence a search for the Hum may well involve electrical filtering to "notch out" these [3, 4].

Fig. 1 shows the (analog) circuit for one such notching scheme (60 and 120 Hz). Don't worry about whether or not you understand this schematic diagram. The "design" is in two parts: the exact arrangements of parts (typically called the "configuration") and the exact values of the parts (typically called the "characteristic"). The actual realization involves a circuit board space of perhaps 5 inches by 5 inches, containing perhaps 3 dozen components. Building and changing the filters can be tedious.

If we were to draw the circuit diagram for a digital filter, to a person unfamiliar with electronics it would superficially resemble Fig. 1, and be quite equally perplexing. The realization, however, would not be on a circuit board, but instead quite hidden inside a computer. Indeed, many and perhaps most electronic devices have digital filters inside. The "digital filter" would be instead a program, software, (corresponding to the analog configuration) with numerical parameters or coefficients (corresponding to the analog characteristic). You would (rarely) rewrite the program, but perhaps frequently change the coefficients. So instead of unsoldering and changing a capacitor, for example, you just change a number in a file. More likely, you would have software where you specify a change of the filter parameter (such as a new notch frequency) and the coefficient file is automatically updated and applied. Obviously this has many advantages, not the least of which is the possibility of using trial-and-error extensively.

The really good news here is likely that any audio software you have may already have notch filtering options which can be adapted to the cancelling of AC hum (fundamental and harmonics). A study below will show how this sort of thing works.

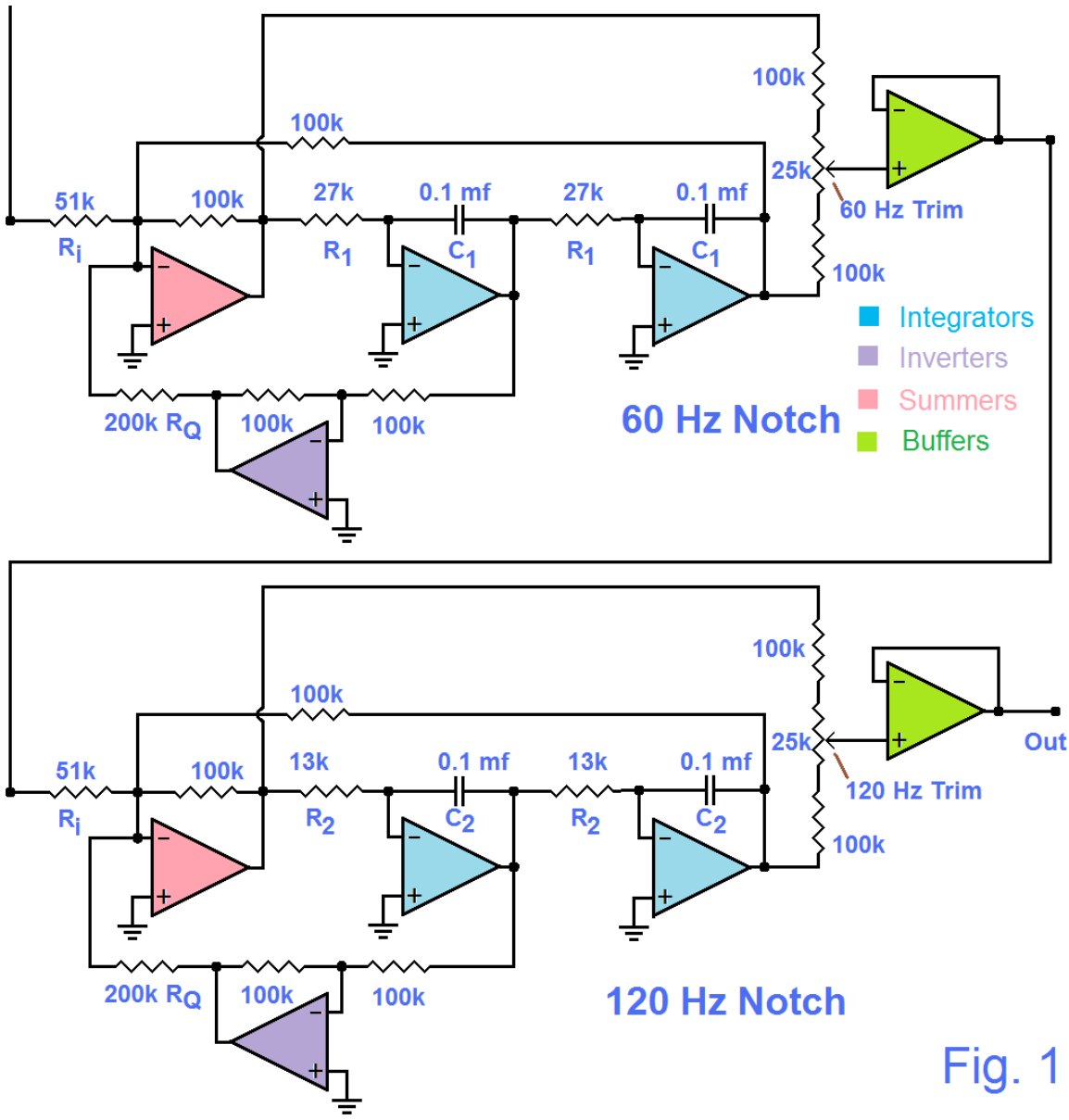
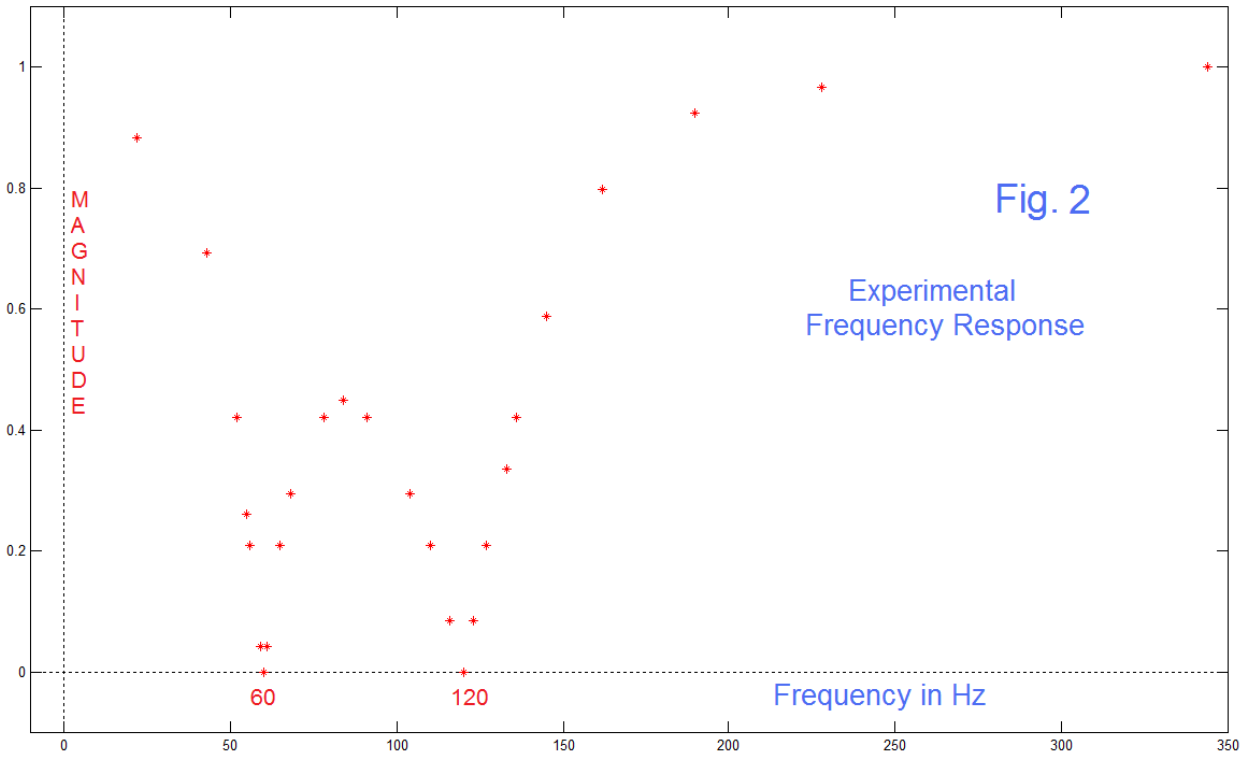


Fig. 1

Before getting to the actual nuts and bolts it may be necessary to say what we hope to accomplish and what is realistic. Fig. 2 shows the frequency response (experimental here) of the analog filter. Perhaps you are disappointed. It does take out 60 Hz and 120 Hz rather exactly, but takes out a lot of the spectrum around these. [Except for some very sophisticated adaptive techniques, these filters would not discriminate between something like 120 Hz from the power lines (don't want) and 120 Hz that might be there secondarily as a desired component.] In this application, we needed to knock down the overwhelming power line noise, and this was accomplished. The thought was that anything away from the notch could at least be observed. The measure of sharpness of a notch is called the Q. Here the Q is set to 2 and is quite low. Audio software may offer ranges of Q as well as notch frequency, so it is well to have an idea what this means. When we get to a custom design below, this notion of sharpness can be addressed more directly.



LINEARITY TO THE RESCUE

Most readers here probably are familiar with the so-called “dual” descriptions of a sound signal in terms either in the time-domain (a waveform) denoted by something like x(t); or an alternative description in the frequency-domain (a “spectrum”), correspondingly denoted X(f). Further, probably at least the terminology “Fourier Transform” (FT) is known to relate the alternative descriptions, uniquely and completely. It is very important that the Fourier Transform (in its many forms) [5] is **LINEAR**. That is, the FT of a sum $x_1(t) + x_2(t)$ is the sum of the FTs: $X_1(f) + X_2(f)$ – the **principal of superposition**. This gives us permission to build up a desired frequency response which may be complicated from a set of more manageable pieces. This is much as we might build a wall from segments and blocks.

We will use linearity in our home-brew approach to calculation custom coefficients. Right here however we learn that if we have a notch in our software package, we can notch out two, three, or more frequencies **by successively running a signal through the program, sequentially for each notch. This may be all we need to know.**

So we see that the linearity of the FT is already potentially very useful, but what is the FT. Well the FT has four forms [5], of which most filter design involves the DTFT (Discrete Time Fourier Transform) which relates a filter's frequency response to its impulse response. The equations are generally defined/used in the DSP (Digital Signal Processing) literature [6]. For the DTFT, the frequency response is:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h(n)e^{-jn\omega} \qquad (1)$$

and the impulse response, the inverse DTFT is:

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(e^{j\omega})e^{jn\omega}d\omega \qquad (2)$$

Here h(n) is generally a real (not complex) time sequence, but $H(e^{j\omega})$ is generally complex as it includes a phase as well as a magnitude. For our simplified purposes here however, we will take H=1 or H=0 (over appropriate ranges), producing a non-causal linear-phase response, for all n from -∞ to +∞, from which we choose a working h(n) centered symmetrically about n=0. In return, H will be real, corresponding to the magnitude of $H(e^{j\omega})$, which approximates the input specification that H was 0 or 1 everywhere.

Equation (2) is kind of begging us to plug in some H, as since everyone can integrate an exponential (!) we get h(n), the coefficients of our finite-length filter (hence FIR or Finite Impulse Response). Thus H disappears from equation (2) and the integration limits now include only the range (or ranges) where H=1.

Let's consider an idealized low-pass filter (because of superposition, we can construct notches from that). We want to give specifications in actual frequencies (Hz), not in terms of the normalized digital frequency ω, where $\omega_s=2\pi$ is the sampling frequency.

Sampling frequency! Where did that come from? Well the complication is that while we can specify something like 120 Hz in the analog case, in the discrete-time (sampled or just "digital") case, frequency responses are periodic. Putting in for H, the value of 1 for $f=-f_c$ to $f=+f_c$ (f_c is low-pass cutoff), and H=0 elsewhere, adjusting the frequency scaling*, and doing the integration, we get:

$$h(n) = \left(\frac{2f_c}{f_s}\right)\frac{\sin\left(\frac{2n\pi f_c}{f_s}\right)}{2n\pi f_c/f_s} \qquad (3)$$

where f_s is the sampling frequency in Hz, f_s=1/T where T is the sampling interval. It computes h(n) for all values, and it is true than h(-n) = h(n) (even symmetry). Note that it is a sinc function. Two complications. First, we have to truncate this to a finite length. Possibly something like -100 to +100 (length 201). Second, when n=0, the calculation blows up. The correct value is found by limits – it is just ($2f_c/f_s$). Deal with this "error" by overwriting h(0), by using a since function if available, or calculate not at exactly n, but at n+0.000001 for all n – something like that.

While equation (3) is the result we need going forward, most users will likely want to compute the inverse of the impulse response to see how well we do with a truncated sequence h(n). As in the computation from equation (2) to equation (3), we desire to avoid exponentials. In plugging into equation (1), due to the symmetry of h(n) we can combine two complex exponentials into cosines (Euler's relationship**) and:

$$H(f) = h(0) + 2\sum_{n=1}^{N} h(n)\cos\left(\frac{2\pi nf}{f_s}\right) \qquad (4)$$

Here we have achieved a simple summation of cosines in frequency, a real (not complex) result, and have summed from n=0 to n=N (one side of the sequence is all that we need). Equation (4) gives H(f) for any f. Generally we would compute it on an array of values of f from 0 to half the sampling frequency. Something like 500 points is likely sufficient so that when we plot H(f) as a function of f, it looks like a continuous frequency response curve.

The two red equations (3 and 4) constitute our simplified design.

BUILDING FROM PIECES

So we know how to design and plot a low-pass filter. How do we get a notch or several notches? We use superposition.

Consider the frequency range of f=0 to f=f_s/2. We restrict our designs to this range – obeying the famous "sampling theorem". Normally we design low-pass filters with a cutoff above f=0 but short of f=f_s/2. Nothing prevents us from setting the cutoff f_c to f_s/2. This would mean, equation (3), that $h_{all}(0)$ = 1 and all other h(n) are zero – a direct path through which makes sense. If we then design a second low-pass with a cutoff f_{c1}, and **subtract** this from the first, we get a high-pass with cutoff f_{c1}. Next we design a third low-pass with cutoff f_{c2}, where $f_{c2}<f_{c1}$, and add this. We end up with a notch (Fig. 3). That is, we have:

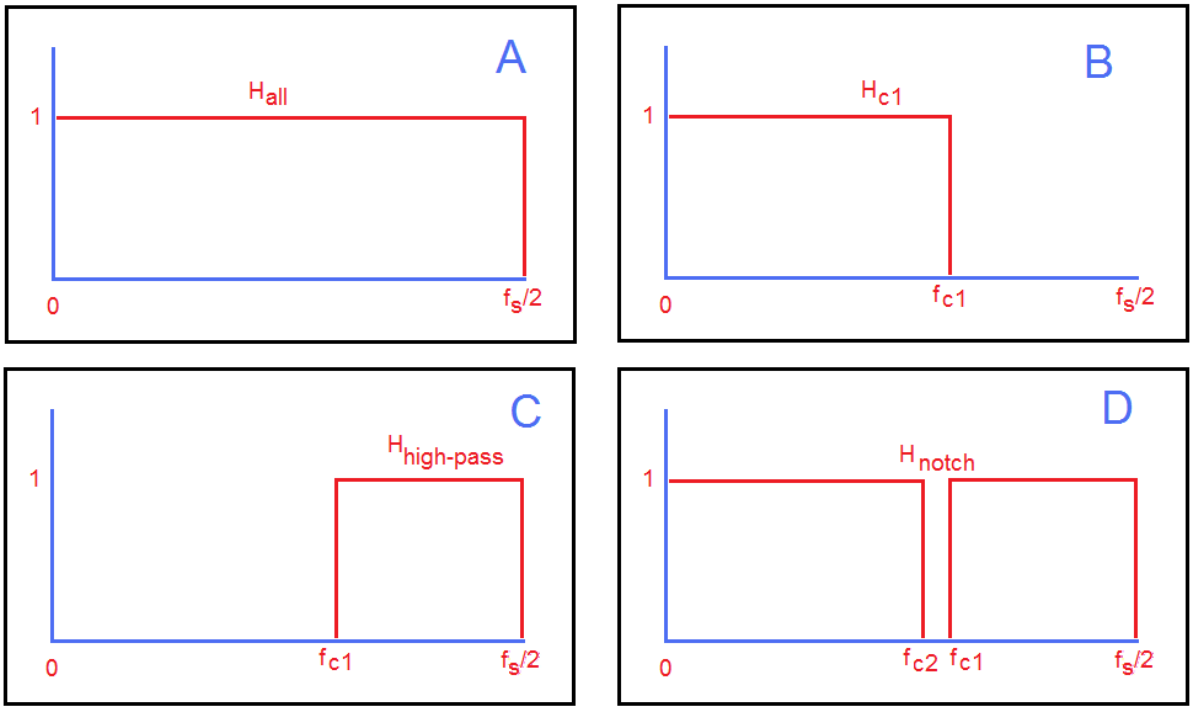


Fig 3 Notch as Linear Sum of Three Responses

$$H_{notch} = H_{all} - H_{c1} + H_{c2} \quad (5)$$

and by the linearity of the DTFT as mentioned above:

$$h_{notch} = h_{all} - h_{c1} + h_{c2} \quad (6)$$

Additional notches can be added in exactly the same way. Each additional notch adds two terms to equations (5) and (6).

A TWO-NOTCH EXAMPLE

Here we will do a two-notch example, which will illustrate several things: (a) how to do additional notches (b) some examples of coding with equations (3) and (4), and (c) how a truncated result is imperfect due to the necessary truncation. We remind the reader that we are doing this coding with the supposition that we are going to use the impulse response for a digital filtering operation. This may be as an input to audio software that accepts “guest” digital filters. Or it may be the case that we intend to write our own implementation filter. This is not hard at all (if you are a reasonably good coder) even lacking significant experience with DSP.

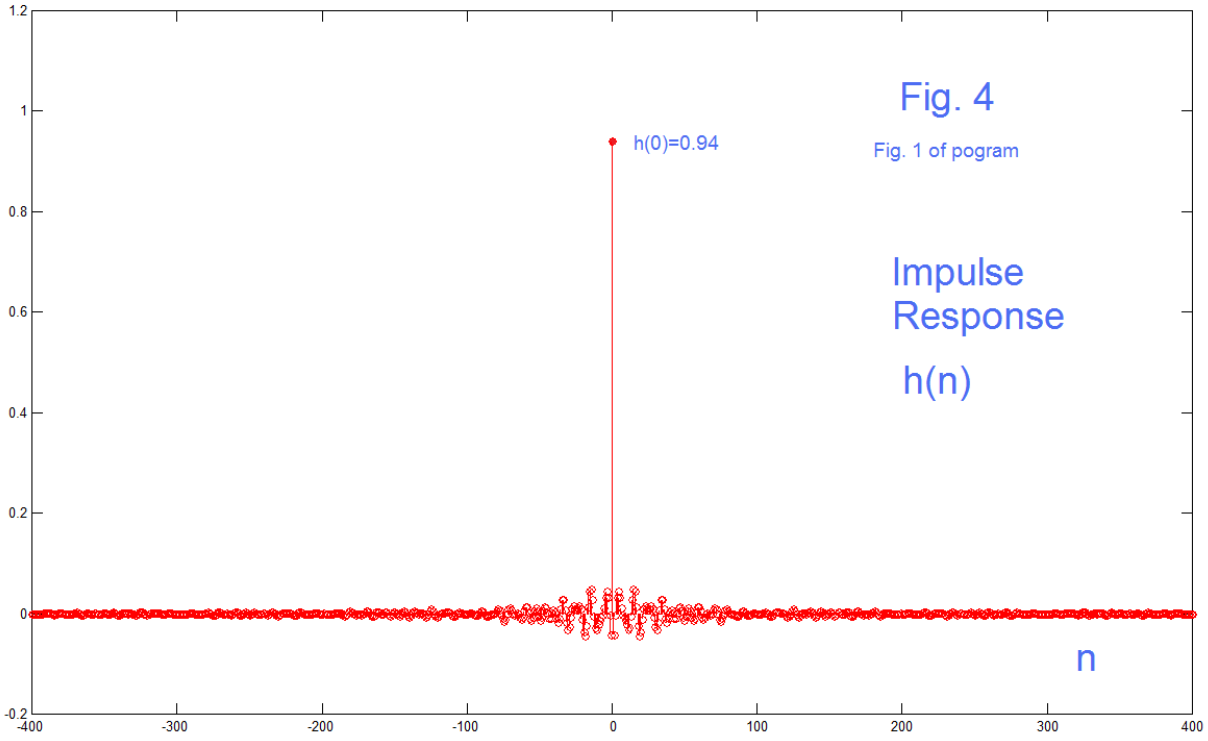
Here is the code:

```
% AN432.m
N=400
n=(-N:N);
fs=10000
f0=fs/2
f1=1650
f2=1550
f3=1100
f4=900

h0 = (2*f0/fs)*sin(2*n*pi*f0/fs)./(2*n*pi*f0/fs)
h0(N+1)= 2*f0/fs
h1 = (2*f1/fs)*sin(2*n*pi*f1/fs)./(2*n*pi*f1/fs);
h1(N+1)= 2*f1/fs;
h2 = (2*f2/fs)*sin(2*n*pi*f2/fs)./(2*n*pi*f2/fs);
h2(N+1)= 2*f2/fs;
h3 = (2*f3/fs)*sin(2*n*pi*f3/fs)./(2*n*pi*f3/fs);
h3(N+1)= 2*f3/fs;
h4 = (2*f4/fs)*sin(2*n*pi*f4/fs)./(2*n*pi*f4/fs);
h4(N+1)= 2*f4/fs;

h =  h0  - h1 + h2 -h3 + h4

figure(1)     % Matlab plot of h
stem(-N:N,h,'r')
figure(2)     % Matlab plot of absolute H
plot([0:fs/1000:fs/2-fs/1000],abs(freqz(h,1,500)),'r')
hold on
plot([0 5000],[0,0],'k')
plot([0,0],[0,1.5],'k')
hold off
axis([-fs/20 fs/20+fs/2 -.05 1.2])

f=[0:fs/1000:fs/2];
ff=pi*f/fs;

% direct plot of H (Fourier series with f and t interchanged)
H=h(N+1);
for k=1:N
   H=H+2*h(N+1-k)*cos(2*k*ff);
end
figure(3)
plot(f,H,'r')
hold on
plot([0 5000],[0,0],'k')
plot([0,0],[-0.2,1.5],'k')
hold off
axis([-fs/20 fs/20+fs/2 -.25 1.2])
```

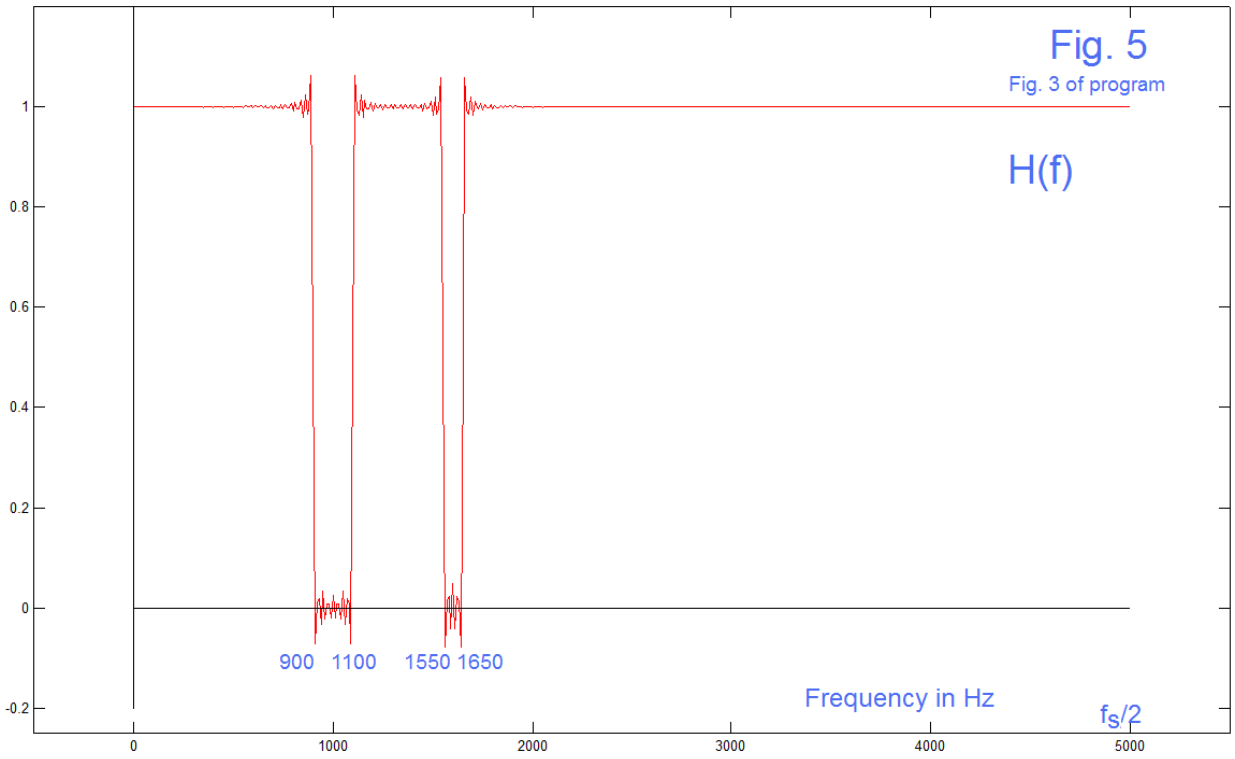
Our code here is Matlab. We give the code as an example, and because often it is very useful to use (inherently unambiguous) computer code in the event that equations and descriptions may remain unclear. At least you recognize that the code listed produced the figures displayed. Nothing in this note requires Matlab.

Here the program is first fed specifications. The length of the filter is set to 801 (n=-400 to n=+400) and the sampling rate is 10,000 Hz. One notch is centered at 1600 Hz with width 100 Hz (1550 to 1650) and the other is centered at 1000 Hz with width 200 Hz (900 to 1100). Pretty easy to say exactly what you want.

Then five impulse responses (the low-pass blocks), h0 to h4, are computed using equation (3). Trivially we sum these as `h = h0 - h1 + h2 - h3 + h4` to get the notches. The code for Fig. 1 (in the program) just displays h, because you are probably curious about it. We used Matlab for the plots here, but any program that plots a data vector should work. Note that h(n) looks a lot like h0: the h_{all} of our discussion. After all, there is relatively little taken out by the two notches, so it should look like that.



Moving on, Fig. 2 (program) is a Matlab calculation of the magnitude of the frequency response and uses the Matlab ***freqz*** function. This we do not plot here as it was used to verify that the hand-code of Fig. 3 (program) is correct. Fig. 3 uses equation (4) so should be the route for the general user. It's just a sum of cosines, the DTFT being a Fourier series in frequency. In this note, Fig. 5 is the plot under discussion.



How well did we do? Pretty good. The edges of the notches are where they belong and are reasonably sharp. There is some "fuzziness" in the vicinities of the transitions. This is the expected "Gibbs Phenomenon", which can be adjusted if necessary***.

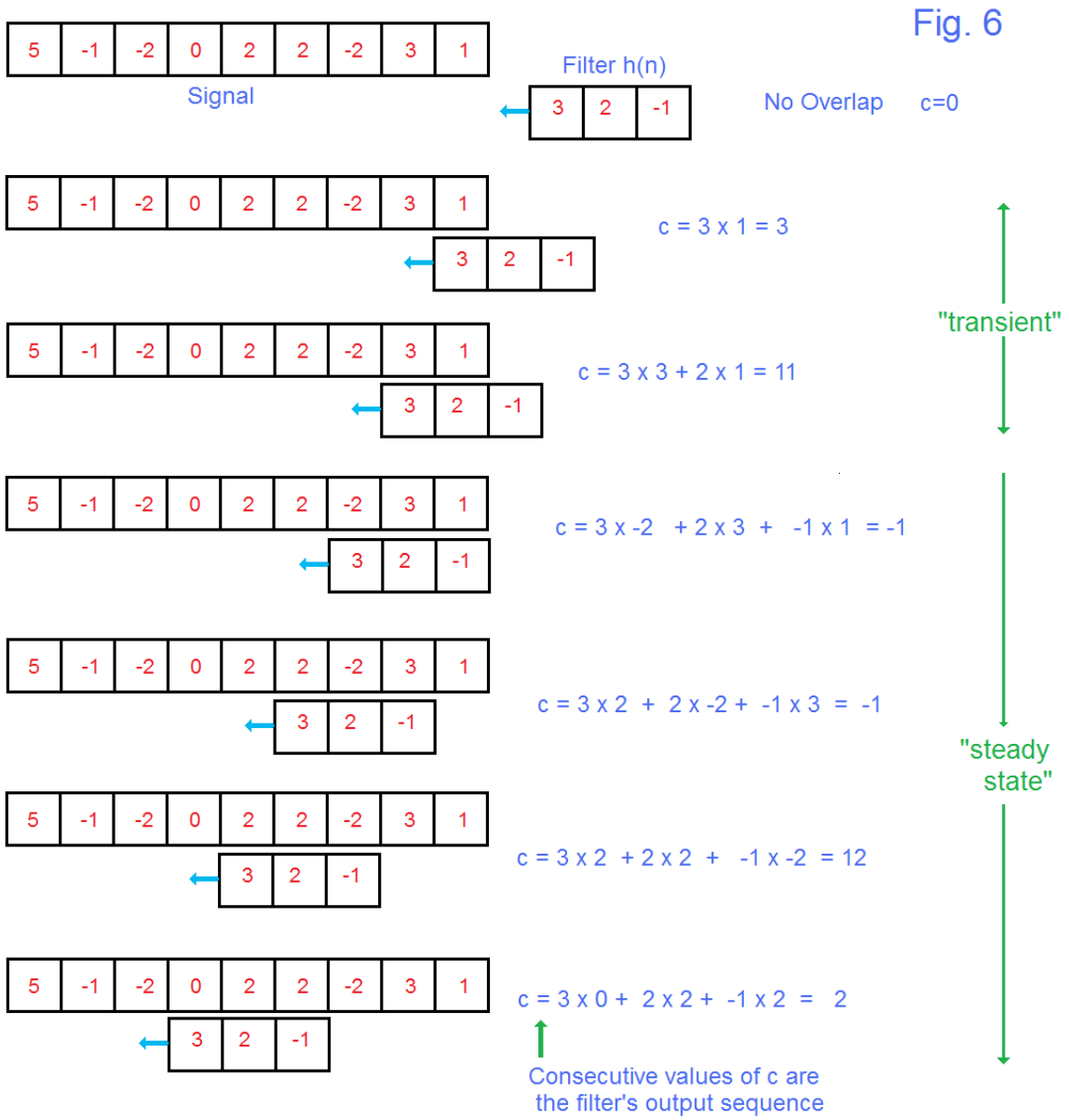
WHAT IF YOU NEED TO PROGRAM THE ACTUAL FILTERING?

Our first choice was probably that some existing audio software that we already have has a suitable notch option, in which case linearity suggested we could use multiple passes of that. A second choice might be that we custom design the necessary impulse response (as we have just done) and the resulting h(n) can be input as a guest to some filtering software. Here we suppose that we have no suitable software, but we can program in some language that handles signals (sequences of numbers). After, all, the boxes we all used are still called computers.

The filtering procedure with an impulse response (FIR filter) is called "convolution". Convolving (please – not "convoluting"!) two sequences is a matter or running one by the other, step-by-step, multiplying together adjacent values for that time interval, and summing all the multiplies in that overlap. Then the data are shifted by one position and the next convolution output value, and so on. Study of Fig. 6 will give the correct understanding. Two points: (1) The filter length will generally be much longer that 3

(perhaps hundreds) and the signal much much longer than the filter (perhaps tens or hundreds of thousands). (2) Also, note that the outputs may be untypical until the signal data fully overlap the filter length. This is called a “transient” and usually is not a large problem as long as it is recognized. The main filtering is called “steady state”, and an ending transient is expected as the signal exits the filter.

It may be the case that a data-processing program has a built-in convolution function. IF you need to write your own, it is not too difficult and even though your code may not be highly efficient****, it will likely run fast enough to accommodate a few signal.



Before ending, it may be instructive to show exactly why a filter can notch out (null) a particular frequency. A "moving average" is a simple low-pass filter which just adds up M adjacent values of a sequence and divides by M (hence the name). A length-4 moving average is thus h(n) = [1/4 1/4 1/4 1/4] as seen in Fig. 7. Frequency analysis of this impulse response shows a "zero" at $f_s/4$. We see this in the time domain (Fig. 7) in that if we have a sine wave of frequency $f_s/4$ (exactly 4 samples per cycle), regardless of phase (30º, 120º, 210º, and 300º in the figure) the average is 0 in all cases.

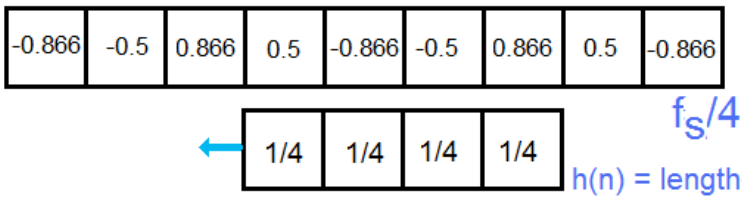


Fig. 7
Notching of $f_s/4$

$$c = 1/4 \times 0.5 + 1/4 \times -0.866 + 1/4 \times -0.5 + 1/4 \times 0.866 = 0$$

SUMMARY

We have shown here we can handle notch filter design and implementation with simple procedure and need not take a full course in DSP or obtain a specific program. In the simplest instance, we understand that existing audio software may be made to work. Moving up, we can custom design an impulse response with simple equations/ procedures. Finally, we may write our own convolution procedures to implement the digital filtering.

NOTES: (if interested)

* For many decades in EE, the lower case omega (ω) was an ordinary frequency expressed in radians/second instead of cycles/second (renamed Hertz of just Hz). That is, $\omega=2\pi f$ where f is frequency in Hz. When DSP came along in the late 1960's, the symbol ω was appropriated for a normalized discrete-time frequency in radians (not radians/sec). This was confusing at first and still is! In this view, the sampling frequency is 2π radians. This notion of ω has the one advantage of representing angle around the unit circle. Old timers (and those who no longer study analog – "continuous time") have gotten used to this. When it is necessary to consider a frequency in radians/second, the uppercase omega (Ω) is often found.

Steiglitz [N1] has usefully pointed out that the "units" of the discrete-time ω should be radians/sample. These are not "physical dimensions" but allow us to keep track of proper frequency scaling. In the DSP case, there is a sampling frequency f_s as we have used above, and the corresponding sampling time is $T = 1/f_s$. A sine wave of frequency ω advances an angle of ω radians between consecutive samples. In summary:

$$\boldsymbol{\omega}\left(\frac{\boldsymbol{radians}}{\boldsymbol{sample}}\right) = \boldsymbol{\Omega}\left(\frac{\boldsymbol{radians}}{\boldsymbol{second}}\right)\boldsymbol{T}\left(\frac{\boldsymbol{seconds}}{\boldsymbol{sample}}\right) = \boldsymbol{2\pi}\left(\frac{\boldsymbol{radians}}{\boldsymbol{cycle}}\right)\boldsymbol{f}\left(\frac{\boldsymbol{cycles}}{\boldsymbol{second}}\right)\frac{\boldsymbol{1}}{\boldsymbol{f_s}}\left(\frac{\boldsymbol{second}}{\boldsymbol{samples}}\right)$$

$$= \frac{\boldsymbol{2\pi f}}{\boldsymbol{f_s}}\left(\frac{\boldsymbol{radians}}{\boldsymbol{sample}}\right)$$

** Euler's Relationships (after Leonard Euler – pronounced "Oiler") relates complex exponentials to sines and cosines. Extremely useful.

$$e^{j\theta} = \cos(\theta) + j\sin(\theta) \qquad and \qquad e^{-j\theta} = \cos(\theta) - j\sin(\theta)$$

which are solved for

$$\cos(\theta) = \frac{e^{j\theta} + e^{-j\theta}}{2} \qquad and \qquad \sin(\theta) = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

When integrating exponentials between symmetric limits (as in the main text) these greatly simplify results.

*** Generally if you just say "Gibbs Phenomenon" (J. Willard Gibbs) people will stop bugging you about the "fuzzy ears"! If you apply a Hamming window [N2] to the impulse response, the overshooting is nearly completely reduced – at the trade-off that the transition rates are somewhat reduced.

**** Convolution is traditionally slow, but the first thing is to get the right answer. May be fast enough and good enough. Because convolution in time can be achieved by jumping to the frequency domain and multiplying, the FFT (Fast Fourier Transform algorithm), even with the overhead, can greatly speed things up.

REFERENCES

[1] "Evidence For Internal Source Of 'The Hum' ", Electronotes Webnote ENWN-47 2/26/2017 http://electronotes.netfirms.com/ENWN47.pdf

[2] Glen MacPherson, " World Hum Map And Database Project " http://thehum.info/

[3] "Notching to Try to Display 'The Hum' ", Electronotes Webnote ENWN-38, 4/11/2016 http://electronotes.netfirms.com/ENWN38.pdf

[4] "Calculating/Measuring the Notch", Electronotes Webnote ENWN-39 4/27/2016 http://electronotes.netfirms.com/ENWN39.pdf

[5] "Fourier Map", Electronotes Application Note No. 410 May 6, 2014 http://electronotes.netfirms.com/AN410.pdf

[6] A extensive series of notes on digital filter design appeared in ***Electronotes*** http://electronotes.netfirms.com/EN197.pdf http://electronotes.netfirms.com/EN198.pdf http://electronotes.netfirms.com/EN198.pdf

[N1] Ken Steiglitz, ***A Digital Signal Processing Primer***, Addison-Wesley (1996), pp65-66

[N2] Hamming window is discussed in many DSP books. See also EN#197, 2b-4 (Reference 6) and http://electronotes.netfirms.com/AN362.pdf