ECG Signal Filtering 8

Electrical Engineering 20N
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Hsin-I Liu, Jonathan Kotker, Howard Lei, and Babak Ayazifar

1 Introduction

In this lab session, we will use LabVIEW to explore a practical application of the discrete-time filters that we have been studying in lectures, lab sessions, and discussion sections. Used in the medical fields, an **electrocardiogram (ECG)** is generated by an electrocardiograph, which measures electrical activity in the heart. ECGs are highly useful in studying heart behavior and in diagnosing potential heart problems. A common and well-documented problem in generating ECGs, however, is **power line interference**, which has a frequency of 60 Hz and arises due to the power lines connected to an electrocardiograph. There are other kinds of interferences in generating ECGs, such as white noise, but we shall only concern ourselves with the power line interference and white noise in the course of this laboratory session. Since power line interference occurs at a frequency of 60 Hz, a good choice to filter out this interference would be a notch filter, whose notch occurs at that frequency. We will use LabVIEW to simulate an ECG signal with the 60 Hz interference and operate upon that signal with a notch filter, and we will design this filter ourselves from scratch.

1.1 Lab Goals

- Implement a practical application of discrete-time filters in LabVIEW.
- Employ geometric reasoning to design a filter with the proper parameters to satisfy the required specifications.
- Analyze the magnitude and the phase of the frequency responses of practical discrete-time filters, and use LabVIEW to verify this analysis.
- Get acquainted with waveform manipulation, shift registers, and feedback nodes in LabVIEW.
- Get acquainted with the modularity of LabVIEW; more specifically, the usage of subVIs and LabVIEW LLBs.

1.2 Checkoff Points

2 Pre-Lab Section

2.1 Delay-Adder-Gain Block Diagrams

We have learned in lab 05 that discrete-time filters can be represented using **linear, constant-coefficient difference equations**, also known as **LCCDEs**. Given an LCCDE, we can construct a visual representation of a discrete-time filter through a **delay-adder-gain block diagram** that, as the name implies, employs delay elements, adder elements, and gain elements. We have achieved prior experience of this in lab 05 for simple LCCDEs. However, as we move onto systems with more complicated LCCDEs, we will now explore a more general method to draw delay-adder-gain block diagrams.

Consider the generalized LCCDE

$$y(n) = \sum_{k=0}^{M} \alpha_k x(n-k) + \sum_{l=1}^{N} \beta_l y(n-l), \tag{1}$$

where α_k and β_l represent scalar gains for the corresponding delayed signals. Note that the gain on the output signal is unity; it is possible for the $y(n)$ term to be preceded by a scalar nonzero gain, say β_0, but in that case, the entire LCCDE can be divided by β_0 to obtain an equivalent LCCDE.

One possible delay-adder-gain block diagram for this generalized LCCDE is shown in Figure 1. However, it assumes the existence of delay elements that can delay a signal by multiple samples. A more efficient delay-adder-gain block diagram is shown in Figure 2, born of the observation that delaying a signal by $k(> 1)$ samples is equivalent to delaying a signal by one sample, and then delaying the resulting signal by one more sample, and so on. The newer delay-adder-gain block diagram assumes that we have, at our disposal, only delay elements that can delay a signal by one sample. This version has the added advantage of being easily implemented in LabVIEW: instead of using the specialized `Y[i] = X[i-n] PtByPt` block, we can use regular shift registers to implement the newer delay-adder-gain block diagram; more specifically, we can use shift registers with cascading delays, as was seen in lab 04, the lab on discrete-time convolution.

Figure 1 Delay-adder-gain block diagram for LCCDE 1.

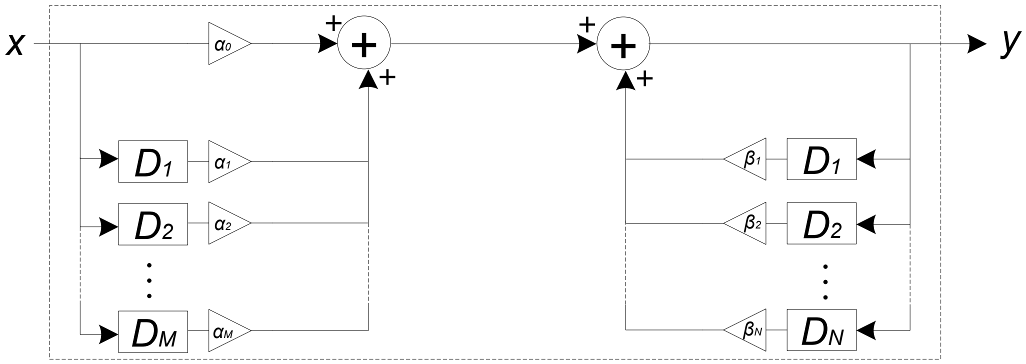
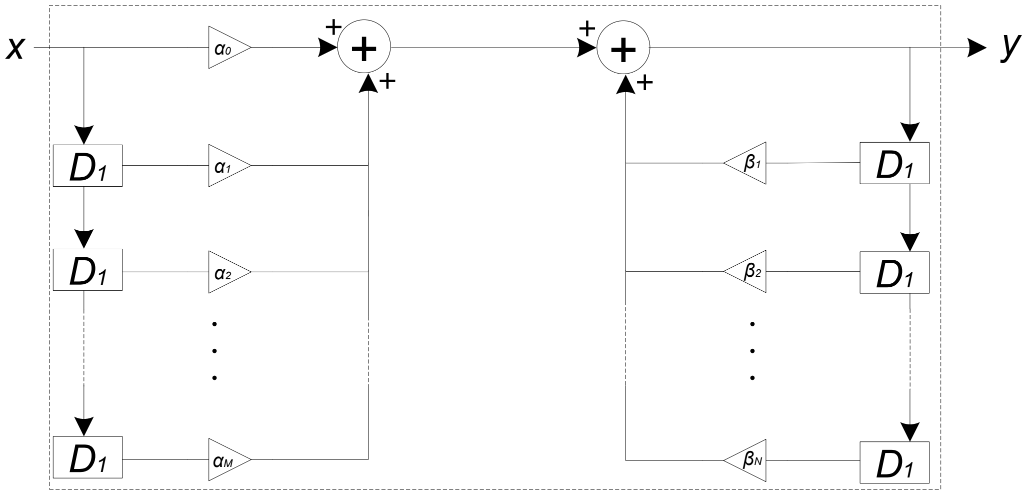


Figure 2 Another delay-adder-gain block diagram for LCCDE 1.



2.2 Geometric Interpretation of Frequency Responses

In this class, and in a lot of real-life applications, we concern ourselves with discrete-time filters whose frequency responses are rational in $e^{i\omega}$. This means that both the numerator and the denominator of the frequency response (for, say, a system F) are polynomials in $e^{i\omega}$. In fact, it can be shown that such a frequency response is of the form

$$F(\omega) = \frac{(e^{i\omega} - z_1)(e^{i\omega} - z_2)\cdots(e^{i\omega} - z_N)}{(e^{i\omega} - p_1)(e^{i\omega} - p_2)\cdots(e^{i\omega} - p_M)}. \tag{2}$$

We are interested in plotting such frequency responses; through these plots, we can understand, among other things, which frequencies are favored, which are attenuated, and which are amplified, allowing us to tweak necessary parameters to get the filter that we need. Unfortunately, since frequency responses are complex functions in general, we cannot plot these responses on paper, since we would require three axes (why?). As a result, we plot the magnitude and the phase of the frequency responses separately, and in doing so, we get a better idea as to how systems affect the magnitudes and the phases of signals with different frequencies.

Providing such plots, however, is not a trivial task. If we needed to generate a precise plot, we would look to a computer to do the plotting for us. For relatively simple frequency responses, however, we can sketch out the general shape of these plots, which is usually enough to afford us an idea of how our filter works.

2.2.1 Low-Pass Filter

For example, we consider the frequency response of a low-pass filter,

$$F_L(\omega) = \frac{e^{i\omega}}{e^{i\omega} - \alpha}, \quad 0 < \alpha < 1.$$

Rewriting the function slightly differently,

$$F_L(\omega) = \frac{e^{i\omega} - 0}{e^{i\omega} - \alpha}, \quad 0 < \alpha < 1. \tag{3}$$

Our main aim now is to draw vectors for each of the terms in the numerator and in the denominator. Now, comparing Equation 3 with Equation 2, we find that $z_1 = 0$ and that $p_1 = \alpha$. On the unit circle in the complex plane, we mark an `O` for every $z_i, 1 \leq i \leq N$ and we mark an `X` for every $p_i, 1 \leq i \leq M$[1]. We then draw the vector $e^{i\omega}$, whose tip rotates on the unit circle (why?). We draw vectors for the other terms, starting from each `X` and `O` and ending at the tip of the vector $e^{i\omega}$. Using laws of vector addition, we know that each of these vectors represents one term in Equation 3. A potential final result is shown in Figure 3.

Awesome! With this in hand, we can provide a qualitative sketch of the magnitude and phase responses of a low-pass filter. We know, for instance, that the magnitude of the frequency response is given by

$$|F_L(\omega)| = \left|\frac{e^{i\omega} - 0}{e^{i\omega} - \alpha}\right| = \frac{|e^{i\omega} - 0|}{|e^{i\omega} - \alpha|} = \frac{1}{|e^{i\omega} - \alpha|}.$$

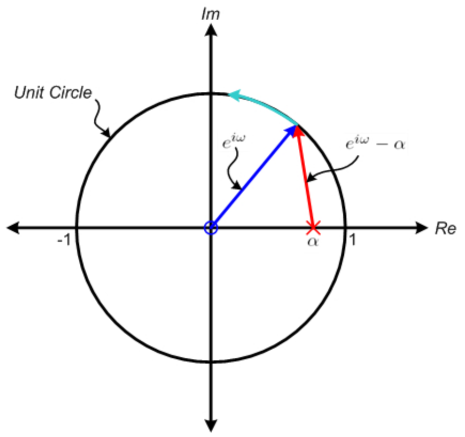
Then, as the vector $e^{i\omega}$ moves about the unit circle, we need only determine how the magnitude (or length) of the vector $e^{i\omega} - \alpha$ changes. By looking at our diagram, we find that the length of the vector $e^{i\omega} - \alpha$ is minimum when $\omega = 0$, and is maximum when $\omega = \pm\pi$. As a result, the magnitude of the frequency response is maximum when $\omega = 0$, and is minimum when $\omega = \pm\pi$. A quick evaluation tells us that

$$|F_L(\omega)||_{\omega=0} = \frac{1}{1-\alpha}, \qquad |F_L(\omega)||_{\omega=\pm\pi} = \frac{1}{1+\alpha}.$$

We collate all of this information and make a rough sketch of the magnitude response, similar to the computer-generated plot shown in Figure 4. Notice that the decrease in the plot, as ω goes from 0 to $\pm\pi$, is not a linear decrease. If we refer back to our diagram, we find that this is because the length of the

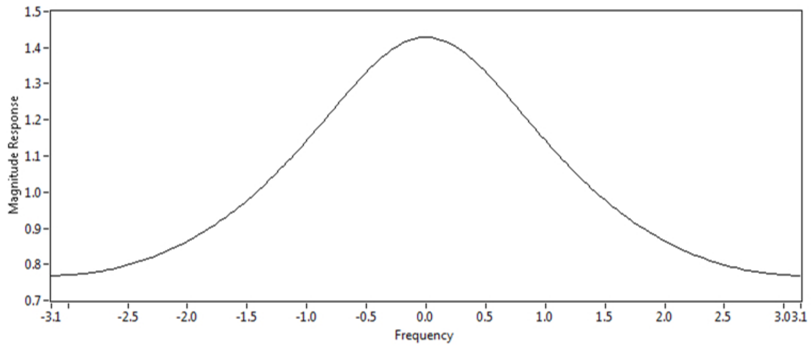
[1]In technical terms, the points we mark as `O` are called *zeros*, while the points we mark as `X` are called *poles*. However, in this class, we will not use these terms; for this class, we use the `X`s and `O`s merely to keep track of points, allowing us to remember which points are in the numerator, and which points are in the denominator.

Figure 3 Geometric Interpretation of a Low-Pass Filter.



$e^{i\omega} - \alpha$ vector does not change linearly either; initially, its length increases quickly, but later on, its length increases relatively slowly. We can use either calculus or a computer to figure out at precisely which point this change in concavity happens, but since we only care about the qualitative shape of the graph, we will not bother. However, we can now clearly see why this filter is a low-pass filter.

Figure 4 Magnitude of the Frequency Response of a Low-Pass Filter.



Correspondingly, the phase of the frequency response is given by

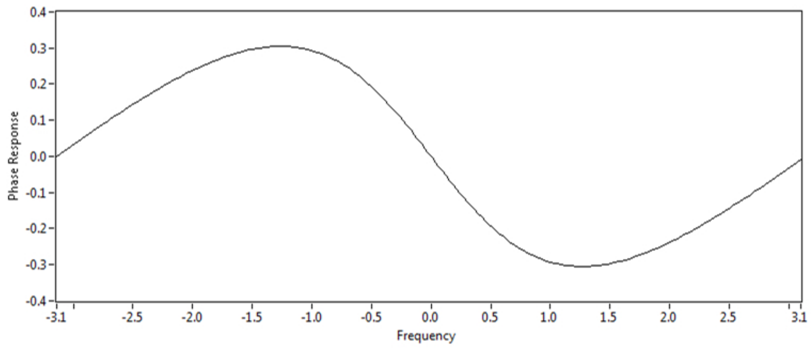
$$\angle F_L(\omega) = \angle\left(\frac{e^{i\omega} - 0}{e^{i\omega} - \alpha}\right) = \angle(e^{i\omega} - 0) - \angle(e^{i\omega} - \alpha).$$

This implies that we should look at the difference between the angles that the vectors $e^{i\omega}$ and $e^{i\omega} - \alpha$ make, respectively, with the real axis. We find that, as ω goes from 0 to π, the angle made by $e^{i\omega} - \alpha$ is always greater than the angle made by $e^{i\omega}$. Thus, the difference is negative, as ω goes from 0 to π, and by symmetry, the difference is positive, as ω moves from 0 to $-\pi$. A quick evaluation tells us that

$$\angle F_L(\omega)|_{\omega=0} = 0, \qquad \angle F_L(\omega)|_{\omega=\pm\pi} = 0.$$

Again, as ω goes from 0 to π, we notice that the angle made by the vector $e^{i\omega}$ changes at a steady rate (how much?), but the speed of the change in the angle made by the vector $e^{i\omega} - \alpha$ is really fast initially, but slows down as ω approaches π. As a result, the phase of the frequency response starts off at 0 and decreases fast, but it eventually evens out and starts to increase as it goes back to 0. A computer-generated plot for the phase of the frequency response is shown in Figure 5.

Figure 5 Phase of the Frequency Response of a Low-Pass Filter.



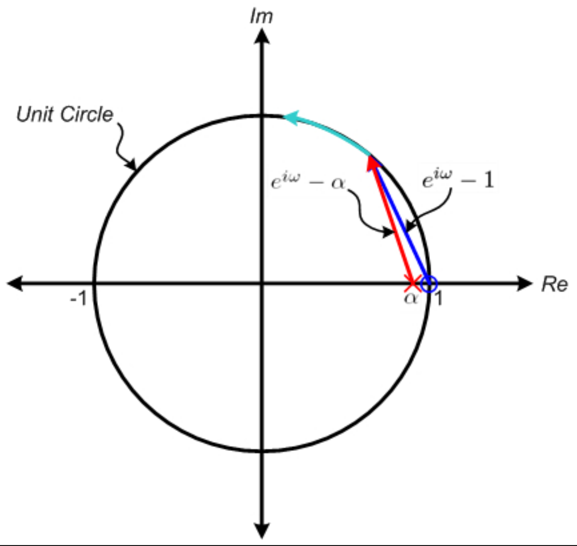
2.2.2 Notch Filter

As another example more relevant to this lab, we consider a different kind of filter, known as a *notch filter*. The motivation behind a notch filter is to attenuate several singular frequencies while preserving the rest. The frequency response of one such filter is given by

$$F_N(\omega) = \frac{e^{i\omega} - 1}{e^{i\omega} - \alpha},$$

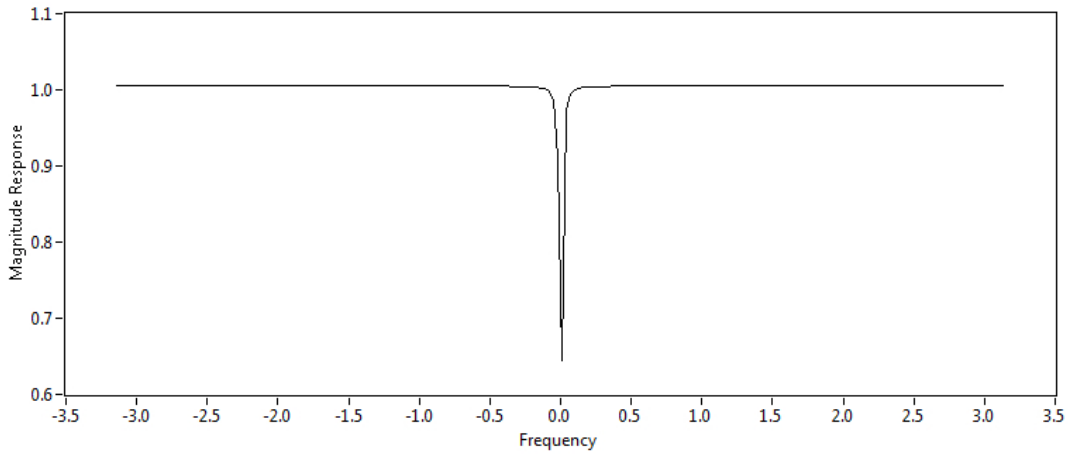
where α is really close to, but not equal to, 1. Following the algorithm presented in section 2.2, we draw the relevant geometric interpretation of the frequency response, as shown in Figure 6.

Figure 6 Geometric Interpretation of a Notch Filter, notching out the Zero Frequency.



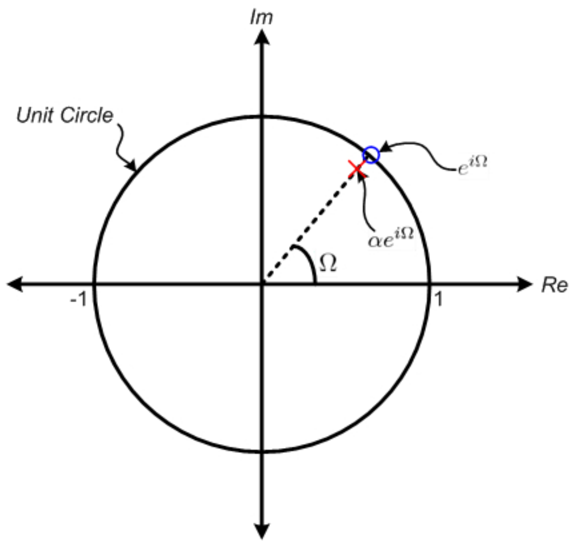
We notice that, for angles beyond a small neighborhood around 0, the vectors $e^{i\omega} - 1$ and $e^{i\omega} - \alpha$ are approximately the same length; this approximation holds better as α gets closer to 1. However, in that small neighborhood around 0, the vector $e^{i\omega} - 1$ is of negligible length; in fact, at $\omega = 0$, the vector $e^{i\omega} - 1$ is precisely zero. As a result, the magnitude of the frequency response $F_N(\omega)$ is approximately 1 for all angles except around zero, where the magnitude is negligible. This filter is said to *notch out* the zero frequency. A computer-generated plot for the magnitude of the frequency response is shown in Figure 7.

Figure 7 Magnitude of the Frequency Response of a Notch Filter.



What would you have to do if you wanted to notch out another frequency – say, Ω – instead of the zero frequency? Where would you place the X and the O? Extrapolating from the case for zero frequency, we deduce that the O should be placed on the unit circle and at angle Ω. Thus, the O should be placed on the complex number $e^{i\Omega}$. Again, the X should be placed slightly away, but not too far away, from the O, and should also be at angle Ω. Thus, the X should be placed on the complex number $\alpha e^{i\Omega}$, where $|\alpha| < 1$, but α is very close to 1. The relevant geometric interpretation is shown in Figure 8.

Figure 8 Geometric Interpretation of a Notch Filter, notching out the Frequency Ω.



What would be the frequency response of such a filter? Recall, from section 2.2.1, that if the frequency response is of the form

$$F(\omega) = \frac{(e^{i\omega} - z_1)(e^{i\omega} - z_2) \cdots (e^{i\omega} - z_N)}{(e^{i\omega} - p_1)(e^{i\omega} - p_2) \cdots (e^{i\omega} - p_M)},$$

we mark an O for every $z_i, 1 \le i \le N$ and we mark an X for every $p_i, 1 \le i \le M$.

3 In-Lab Section

Now that we are armed with notch filter theory, it is time for us to put it to good use.

3.1 So You Think You Can Design a Notch Filter

For all of the in-lab sections, we will be concerned with the following problem: we have a signal $x(n)$, which in this case is a pure ECG signal. However, the signal that we obtain, $x'(n)$, is *not* a pure ECG signal; it is an ECG signal corrupted with a noise signal, $\eta(n)$. In other words,

$$x'(n) = x(n) + \eta(n),$$

and we are interested in filtering out $\eta(n)$.

For the purposes of this problem, we consider $\eta(n)$ to be the function $A\cos(\Omega n)$ ($A \in \mathbb{R}$), for several reasons: this noise signal simulates the actual problem well, and is a real-valued signal. Now, we know that the cosine function contains two frequencies – $\pm\Omega$ – and in order to remove the error signal from $x'(n)$, we need to notch out these frequencies. Thus, a notch filter will satisfy the requirements of this problem. Let us flesh out this notch filter, which we will call N.

1. We know, from section 2.2.2, how to create a filter that can notch out one frequency. Then, in order to notch out *two* frequencies, we should consider each frequency separately and notch it out separately. With this in mind, generate the necessary frequency response, $N(\omega)$, for the filter N. It should have the form
$$N(\omega) = \frac{(e^{i\omega} - z_1)(e^{i\omega} - z_2)}{(e^{i\omega} - p_1)(e^{i\omega} - p_2)}$$
for some complex values of z_1, z_2, p_1, p_2.

2. Once we have the frequency response that we require, we need to determine the corresponding LCCDE, so that we can implement the filter in the time domain. We first consider the general case: determine the frequency response of a system F, which is specified by the general LCCDE
$$y(n) = \sum_{k=0}^{M} \alpha_k x(n-k) + \sum_{l=1}^{N} \beta_l y(n-l).$$

3. Based on your responses to step 1 and step 2, determine the LCCDE for the notch filter N. A few tips:
   (a) You may find it useful to bring $N(\omega)$ to the following form:
   $$N(\omega) = \frac{1 + D_1 e^{-i\omega} + D_2 e^{-2i\omega}}{1 + D_3 e^{-i\omega} + D_4 e^{-2i\omega}},$$
   for some complex constants D_1, D_2, D_3, D_4.
   (b) Try and simplify your LCCDE as much as possible, so that the coefficients of each term are real; use any identities that you may know.
   (c) The final LCCDE should have the form
   $$y(n) = [x(n) + C_1\cos(\Omega)\,x(n-1) + C_2 x(n-2)] + [C_3\cos(\Omega)\,y(n-1) + C_4 y(n-2)], \quad (4)$$
   where C_1, C_2, C_3, C_4 are constants that may, or may not, depend on α.

4. Draw the delay-adder-gain block diagram for your notch filter, based on the LCCDE 4 that you created in step 3.

Woah. What did you just do? You have not just designed a notch filter to notch out a particular signal, but you have also generated its LCCDE! Keep this LCCDE safe; the rest of the lab will be devoted to testing whether or not your notch filter does its job correctly.

3.2 Ω-Force

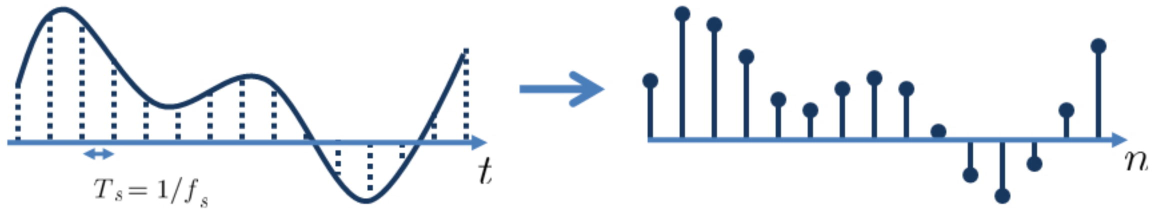
This section should take approximately **30 minutes** to complete. Before continuing with this section, read the document *The Many Faces of Frequency*, if you haven't already. It's available on the course website.

What exactly is the frequency, Ω, that we have to notch out? We know that our pure ECG signal is corrupted by a 60 Hz signal, but the combination of both the ECG signal and the noise signal exists in *continuous-time*, and as such 60 Hz (or cycles per second) is the **continuous-time frequency** of the noise signal. Since a computer can only store discretized signals, we have to sample the corrupted signal and produce a corresponding discrete-time signal, as shown in Figure 9. The noise in this new, discrete-time signal has a different, **discrete-time frequency** (in radians per sample) and we need to determine this new frequency—this will be the Ω that we have to notch out.

CONTINUOUS-TIME FREQUENCY

DISCRETE-TIME FREQUENCY

Figure 9 Sampling a continuous-time signal to produce a discrete-time signal.



The document *The Many Faces of Frequency* concludes that

$$\Omega = \omega_0 T_s = \frac{\omega_0}{f_s} = \frac{2\pi f_0}{f_s},$$

where f_0 and ω_0 are the continuous-time frequencies of the original continuous-time noise signal, measured in *Hertz* and *radians per second* respectively, f_s is the sampling frequency measured in *samples per second*, and Ω is the discrete-time frequency of the derived discrete-time noise signal, measured in *radians per sample*.

For the lab, we will be trying to filter out a noise signal with continuous-time frequency 60 Hz. Using the relationship above, determine the discrete-time frequency, Ω, of the sampled noise signal, as a function of f_s. **This is the Ω that we will be notching out.**

3.3 Implementing the Notch Filter SubVI

As we have seen in Lab 6, subVIs in LabVIEW are very useful for taking chunks of a block diagram and encapsulating them into reusable blocks. In the case of the guitar string VI, we used a subVI to enclose the low- pass filter machinery. In this section, we will implement the notch filter, and then convert it into a subVI that we will use later on to feed the raw ECG signal into.

1. Download the `NotchFilterTemplate` VI from the course website. This template includes a cosine generator, a waveform viewer, and a skeleton frame of the notch filter itself. Let's quickly familiarize ourselves with what we have.

2. The cosine generator simply generates a cosine signal with a continuous-time frequency of f_{input} Hz given a sampling frequency of f_s (why is the case?). This will be the temporary input signal.

   The waveform viewer just displays the output signal of our system as a waveform.

3. Finally, the bare-looking for loop is the skeleton of the notch filter. This will be the *system* that we implement. The previous two components are merely a means of verifying that the notch filter is working.

Notice that there are three controls that parameterize the notch filter:

- f_0, which is the continuous-time frequency we want to notch out
- α, which is the sharpness of the notch filter
- f_s, which is the sampling frequency

At this point, let us focus our attention on fleshing out the notch filter.

4. To actually implement the notch filter, there isn't really anything more than realizing the LCCDE you constructed in section 3.1. You will need to employ the values from the three controls mentioned earlier. The comments[2] in the for loop may help point you in the right direction.

   To obtain the delayed versions of the signals you need, you may choose to either use the shift registers as provided in the template, or use feedback nodes (refer back to Lab 6 if you need a refresher on how they work).

5. Test your implementation by running the VI. Adjust the slider for α to any value near, but not exactly, 1, and then set the slider for f_0 to any value. Then, move the f_{input} slider around to see how the filter affects the cosine signal for different frequencies, especially near the value you had set for f_0. What do you think you should expect to see?

   If your system's behavior does not match your expectations, carefully check the block diagram, making sure it really reflects the LCCDE you want to implement. Forgetting to multiply a wire by some scalar is a common bug. If the block diagram looks right, but the behavior still seems off, ask your classmates or your lab GSI.

6. If the system *is* behaving as expected, great! It's time to convert this VI into a subVI.

   First, save the current VI, and then save it again under the different name `Notch Filter`. Then, remove the surrounding while loop and the wait timer. Then, delete the cosine generator and the waveform viewer. The only things you should have left are the notch filter itself (the big for loop), and its three controls, f_0, α, and f_s.

7. Now, we need to specify the inputs and outputs of this subVI.

   The inputs we want are:

   - f_0, the continuous-time frequency we want to notch out
   - α, the sharpness of the notch filter
   - f_s, the sampling frequency
   - input array, the input signal

   and the output we want is:

   - output array, the output signal

   To do this, we first need to attach an array control block as the input signal to the system. In other words, we need to attach it to the auto-indexed tunnel that the cosine generator was originally attached to. To make one, right-click the auto-indexed tunnel, **Create**, and then **Control**. Name it `input array`.

   Similarly, we need an array indicator coming out of the system to be the output array. Namely, we want it attached to the auto-indexed tunnel that was originally attached to the waveform viewer. To make one, right-click the auto-indexed tunnel, **Create**, and then **Indicator**. Name it `output array`.

[2] Any programmer knows that comments are just plain awesome because they document the code and help explain what's going on. LabVIEW comments are no exception. You can make your own comments by double-clicking any blank area on the block diagram, and then typing the omment in the yellow-ish textbox that popups. You can then move around and resize the comment as needed. Remember, LabVIEW will completely ignore comments during running time, so it never hurts to comment more!

8. Now that we have all four controls as well as the indicator ready, we can now link them to the connector terminals. Recall that to do so, right-click the **VI Icon** and select **Show Connector**. Then, click on a connector terminal, followed by a click on the control or indicator you want to link to it. Feel free to make your own icon, but again, don't spend too much time on it

9. And we're done! We've made our notch filter, complete with the ability to specify several parameters, including the exact frequency we want to remove.

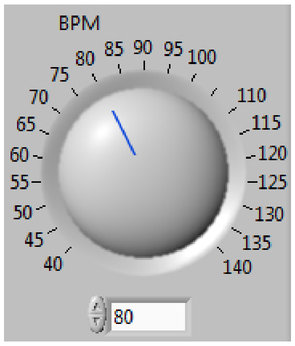
3.4 Generating the ECG

This section should take approximately **20 minutes** to complete.

In this section, we will explore the VI that will generate a simulation of an ECG signal for us to filter, and we will also modify it slightly for our purposes.

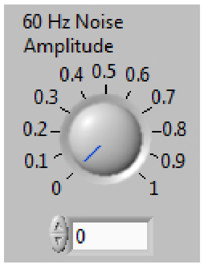
1. Open the LabVIEW Library File (LLB) called `SimulateECGSignalSource`; this Library keeps together the VIs needed to simulate an ECG signal. Select the VI also called `Simulate ECG Signal Source`. Run the VI and explore the various options available on the front panel to tweak the ECG signal. Some of the salient controls are:

   (a) `BPM`:

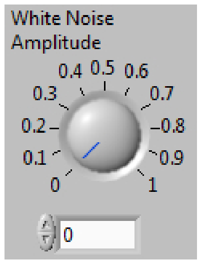
   This control helps to simulate a particular amount of heartbeats per minute.

   (b) `60 Hz Noise Amplitude`:

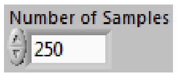
   This control determines the amplitude of the 60 Hz interference in the ECG generated.

   (c) `White Noise Amplitude`:



This control determines the amplitude of the white noise interference in the ECG generated.

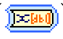
(d) `Number of Samples`:



This control determines how many samples of an ECG are plotted.

(e) `Simulate Acquisition Timing`:

This boolean control adds a delay to the generation of the ECG signal to ensure that it does not change rapidly, since in reality, acquiring an ECG signal is not instantaneous.

2. Move to the block diagram and notice that the ECG on the front panel is merely an indicator on the data produced by a subVI called `subSimulate ECG Signal.vi`. However, the data is in `Dynamic Data` form. To work with the data, we need to convert it to a form that we can use – in this case, we need to convert it to a **waveform**. Use the `Convert from Dynamic Data` block () and select the `Single Waveform` option, which is at the bottom of the list, as shown in Figure 10. The output of this block will be the waveform that we need.

   Dynamic data is the type of data generated by **express VIs**, which are VIs "whose settings can be configured interactively through a dialog box" (as described by LabVIEW Help). We have seen express VIs before, such as the `Express XY Graph`.

3. We will need to use both the `Y` and `dt` components of the resultant waveform; to this end, use the `Get Waveform Components` block to extract these components. The components we need are the sampling period (`dt`) and the waveform data (`Y`).

4. Use the `dt` value, and one other LabVIEW block (which one?), to determine the *sampling frequency* of the waveform generated, in *Hertz*. **Herein, we will refer to this quantity as f_s**, the actual sampling frequency used by the VI.

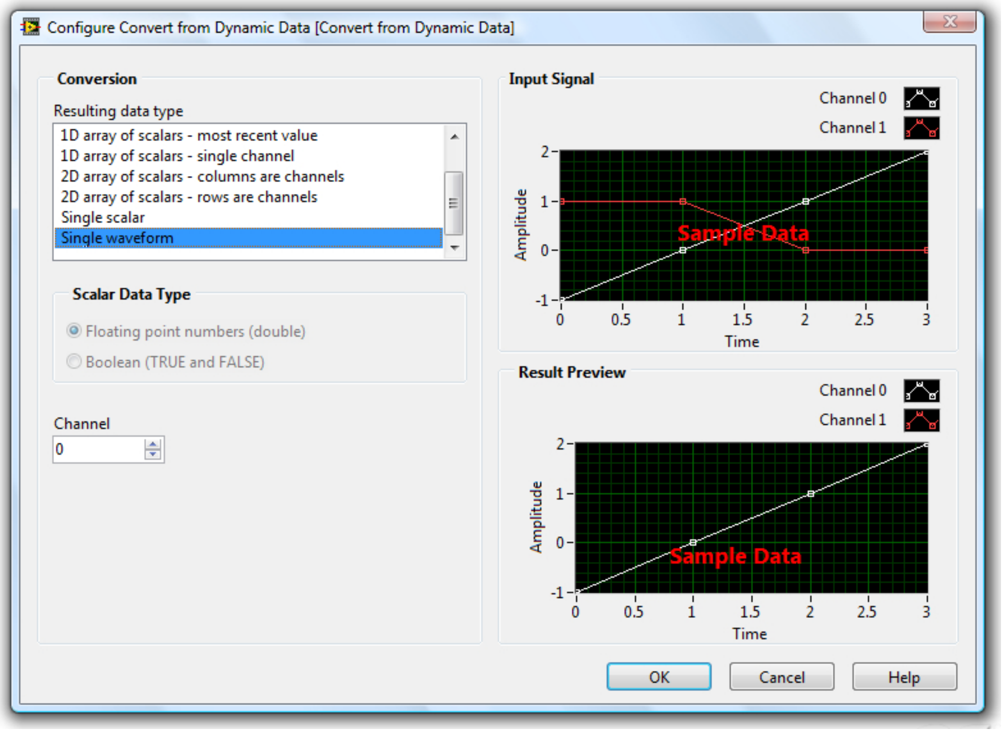
3.5 Filtering the ECG: Bringing It All Together

This section should take approximately **30 minutes** to complete.

You have designed a discrete-time notch filter, you have even *created* that notch filter, and you have explored an ECG signal simulator: now it is time to attack the actual problem, that of filtering power line interference.

1. Right-click on the block diagram of `Simulate ECG Signal Source`, choose `Select SubVI...` and select the `Notch Filter` subVI that you built in section 3.3. If done properly, your subVI should appear as a box having the default LabVIEW icon for VIs. Hover over the inputs and output to verify that they have been named properly, and that all of them are present.

Figure 10 Dialog box for the `Convert from Dynamic Data` block.



Configure Convert from Dynamic Data [Convert from Dynamic Data]

Conversion

Resulting data type

- 1D array of scalars - most recent value
- 1D array of scalars - single channel
- 2D array of scalars - columns are channels
- 2D array of scalars - rows are channels
- Single scalar
- [✓ Single waveform]

Scalar Data Type

[✓ Floating point numbers (double)]
Boolean (TRUE and FALSE)

Channel: 0

2. Connect the ECG data (`Y`, extracted in step 3 of section 3.4) as the input signal to the notch filter, with the other inputs being controls that change f_0 and α. In addition, use the f_s that you retrieved in step 4 of section 3.4 to control the sampling frequency.

3. Let us finally see our beautiful theoretical handiwork in practice. Plot the output signal using a `Waveform Graph`, which requires the `Build Waveform` block. The final front panel should resemble the one shown in Figure 11. Notice that the x-axis is labeled `Time`, so generate your x-axis values accordingly.

   A potential roadblock is ensuring that the values along the x-axis are actual points in time, as opposed to simply sample numbers, so that the graph of the output signal can be interpreted better. One way to generate the array of values along the x-axis correctly would involve the `dt` component that you extracted in step 3 of section 3.4. Because the `Build Waveform` block accepts a `dt` input to be used in your `Waveform Graph`, you must make sure that you supply the correct `dt`.

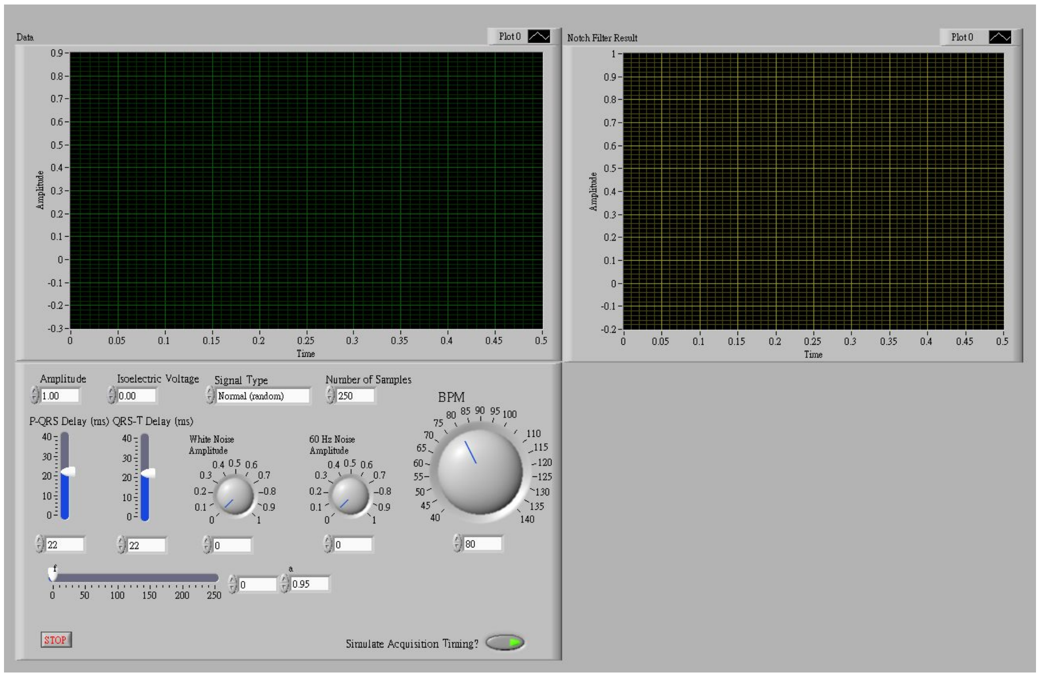
4. Back on the front panel, set your notch filter to notch out the 60 Hz frequency. Increase the `60 Hz Noise Amplitude`. If everything was done properly, the output ECG signal should be a clean filtered version of the input ECG signal. We recommend the following sanity checks:

   (a) Tweak the value of α and observe any changes in the output signal. Observe that the lower the sharpness, the wider the range of frequencies attenuated.

   (b) Ensure that, even with changes in the values of `BPM` or in the `60 Hz Noise Amplitude`, the output ECG will always be a clean filtered version of the input noisy ECG.

   (c) The output signal should be almost a replica of the input signal when the `60 Hz Noise Amplitude` is zero.

Figure 11 `Simulate ECG Signal Source.vi` Front Panel.



Amplitude 1.00
Isoelectric Voltage 0.00
Signal Type Normal (random)
Number of Samples 250
P-QRS Delay (ms) 22
QRS-T Delay (ms) 22
White Noise Amplitude 0
60 Hz Noise Amplitude 0
BPM 80
f 0
a 0.95
Simulate Acquisition Timing?

5. Congratulations once again! You have solved a problem in the medical field simply by applying your knowledge of discrete-time notch filters, and along the way, you have successfully attempted basic filter design and created a subVI. Go on and take a bow.

4 Post-Lab Section

4.1 Cleaning It Up Further

In the `Simulate ECG Signal Source.vi` used in the in-lab sections, there is another source of noise called `White Noise`. This simulates another kind of interference that can occur when you collect the ECG data. This kind of noise can originate from several places: the ECG electrodes, the cables and wires connecting the electrodes to the monitor, the electrical coupling inside the ECG monitor, among other places. The characteristics of white noise are beyond the scope of this course and will be introduced in EE126. For our purposes, we can treat it simply as a high frequency noise with a wide range of high frequency components. Cleaning out the noise is only the first step in ECG signal processing; there exist LabVIEW VIs that simulate the entire process[1].

4.2 A Better Low-Pass Filter

Since the ECG data consists of frequency components from 0 Hz to 25 Hz, we imagine that a good low-pass filter can filter out the noise, and still leave a good amount of data for further processing. This is the main task for this section.

Unfortunately, the low-pass filters we have explored in lecture or in lab 05 are not powerful enough to achieve a good performance. Designing a good low-pass filter is the main focus topic of EE123, so we do not go into the details of such filter design here. However, if we are given the frequency response of a low-pass filter, we should be able to evaluate and test its performance. Indeed, the low-pass filter that we will be exploring is very similar to, although much more powerful than, the one-step moving average low-pass filter that we explored in lab 05.

Assume that we have the following frequency response $H(\omega)$ of our better low-pass filter:

$$H(\omega) = \frac{0.0004 + 0.0017e^{-i\omega} + 0.0025e^{-2i\omega} + 0.0017e^{-3i\omega} + 0.0004e^{-4i\omega}}{1 - 3.1806e^{-i\omega} + 3.8612e^{-2i\omega} - 2.1122e^{-3i\omega} + 0.4383e^{-4i\omega}}.$$

1. Plot the magnitude and phase response using LabVIEW. Confirm first that the filter with frequency response H is indeed a low-pass filter, and that this filter outperforms the two-point moving average filter introduced in lab 05.
2. Derive the LCCDE for this filter.
3. Implement this filter in the time domain as a subVI called `Better Low-Pass Filter`. Since this low-pass filter has no parameters that can be adjusted, the specifications for this subVI would simply be:
   (a) **Input Signal** as an **input**: An array containing the values of the input signal at different points of time.
   (b) **Output Signal** as an **output**: An array containing the values of the output signal at different points of time, spaced apart at the same sampling period as the input signal.
4. Cascade your filter with the notch filter in `Simulate ECG Signal Source.vi`: in other words, the input to the better low-pass filter should be the output of the notch filter. Also, redirect the plot of the output signal to plot the result of the better low-pass filter instead.
5. Run your new VI with both the white noise and the power line interference, and determine if the output signal is a cleaned version of the input signal.

   The amplitude of the white noise should be below 0.1. In the real world, if your white noise has a high amplitude, then there is something wrong with your device!

4.3 I Have an LLB in LabVIEW

For lab sessions to come, we will focus a little more on block diagram organization, layout, and style. To this end, we will be implementing and using more subVIs in various exercises, and in order to keep a VI together with its embedded subVIs, we will make further use of LLBs, or LabVIEW Library Files, one of which we have seen and used in section 3.4.

A good analogy to LLBs would be ZIP files or tarballs, which keep files that depend on each other together as one cohesive component. If we try to move one VI that has subVIs elsewhere, we will also need to move the subVIs themselves to the same place, since LabVIEW expects the subVIs to be in the same directory as the main VI. LLBs take care of this automatically for us.

We will mainly be converting between directories and LLBs, via the `LLB Manager`. We have the main ECG signal generator – the VI `Simulate ECG Signal Source` – in an LLB, and we would like to add our subVIs – both the `Notch Filter` subVI and the `Better Low-Pass Filter` subVI – into this LLB.

1. In the `LLB Manager`, navigate to the directory containing the `Simulate ECG Signal Source` LLB file. Right-click on the LLB file and select `Convert`.
2. Follow the prompts and the instructions to convert the LLB file into a directory with the same name.
3. Move the VI files for your subVIs inside this directory. After this step, your directory should contain the VIs

   (a) `Simulate ECG Signal Source`
   (b) `subSimulate ECG Signal`
   (c) `Notch Filter`
   (d) `Better Low-Pass Filter`

   You may need to re-insert your subVIs into the main `Simulate ECG Signal Source` VI at this point.

4. In the `LLB Manager` again, navigate to the directory containing the `Simulate ECG Signal Source` directory. Right-click on the directory and select `Convert`.

5. Follow the prompts and the instructions to convert the directory into an LLB file with the same name.

Great! You know have a cohesive ECG noise-filtering package[3]. Test that it works by moving the LLB file into different directories and running the `Simulate ECG Signal Source` VI.

4.4 Submission Instructions

1. Save your responses to step 1 in section 4.2 in a VI called `BLPF Frequency Domain`.

2. Type your response to step 2 in section 4.2 **as a comment** in the `Better Low-Pass Filter` subVI.

3. Log on to bSpace and click on the `Assignments` tab.

4. Locate the assignment for `Lab 8 Post-Lab` corresponding to your section.

5. Attach the following files to the assignment:

   (a) A text document, called `PARTNERS.txt`, containing your name and your partner's name, if any.
   (b) The VI `BLPF Frequency Domain`.
   (c) The subVI `Better Low-Pass Filter`.
   (d) The `Simulate ECG Signal Source` LLB, now including the better low-pass filter along with the notch filter.

4.5 Submission Rules

1. Submit your files *no later than* 10 minutes after the hour in the beginning of your next lab session, during the week of **April 25, 2011**.

2. Late submissions will *not* be accepted, except under unusual circumstances.

3. These exercises are recommended to be done *in groups of two*. Only one person need submit the required files, however.

5 Acknowledgments

Special thanks go out to the teaching assistants (TAs) of the Spring 2009 semester (Vinay Raj Hampapur, Miklos Christine, Sarah Wodin-Schwartz), of the Fall 2009 semester (David Carlton, Judy Hoffman, Mark Landry, Feng Pan, Changho Suh), and of the Spring 2010 semester (Xuan Fan, Brian Lambson, Kelvin So) for providing suggestions, ideas, and fixes to this lab guide.

[3]If any of your family and friends need such a package, you can surprise them.

References

[1] Mihaela Lascu and Dan Lascu. LabVIEW Electrocardiogram Event and Beat Detection. *WSEAS Trans. on Computer Research*, 3(1), January 2008.