EECE 2150 – Circuits and Signals: Biomedical Applications

Lab 16 Digital Signal Manipulation of the ECG Signal

Introduction:

Now that you have some ECG signals recorded using the analog portion of the ECG system, combined with the A/D converter, you are ready to work in the digital world to improve and extract information from your signal. Although you filtered out many of the high frequencies before A/D conversion, you may be able to do a more complete job in the digital realm. Because $60\,Hz$ interference is within the bandwidth of your ECG signal, it is not easy to filter it out of the signal before A/D conversion. We may be able to do better by manipulating the digital signal. Finally, extracting information such as the heart rate or the variability of the heart rate may be useful in clinical settings. **You should complete as many of these tasks as you have time for (not necessarily in this order).**

1. Build a digital low-pass filter to reduce the noise in the ECG signal.
2. Remove the 60 Hz interference, either by modifying the Fourier coefficients of the signal or by using a digital notch filter centered at 60 Hz.
3. Automatically detect the heart rate in the ECG signal.

Filtering DT signals using the Matalb *filter* function

Background: The expressions in discrete time that correspond to differential equations in continuous time are called *difference equations*, and have the form (after moving all output terms except one to the right side of the equation):

$$y[n]=-\left(a_1 y[n-1]+a_2 y[n-2]+\ldots a_N y[n-N]\right)+¿$$

Note in particular that the output coefficients a_k on the right side have minus signs so that they will have + signs on the left hand side.

This equation corresponds to an algorithm (actually, there are many many ways to actually implement this, but here we are talking conceptually, at a high level). To see this, suppose that

1. we start computing outputs for $n=0$, and
2. we have advance knowledge that both $y[n]=0$ and $x[n]=0\,for\,n<0$.

The input signal $x(n)$ can be any discrete signal, as long as this last condition is satisfied.

Then the first output, $y[n=0]=b_0 x¿]$ since all other terms are zero since they correspond to terms for which $n<0$.

The next output value, $y[n=1]=-a_1 y[0]+b_0 x[1]+b_1 x[0]=-a_1 b_0 x[0]+b_0 x[1]+b_1 x[0]$ (since again all other terms =0).

The third value, $y[n=2]=-a_1 y[1]-a_2 y[0]+b_0 x[2]+b_1 x[1]+b_2 x[0]$, for which we could substitute in the expressions we already have for $y[0]$ and $[1]$.

Thus we see that we can recursively compute all values of the output $y[n]$ for as many outputs as we like, based on the filter coefficients and the input values.

Matlab implements this type of algorithm (in a much more sophisticated way, of course), in a function called, appropriately enough, *filter*. The Matlab function *filter*, in the basic way in which we will use it, takes as its inputs the vectors of coefficients that describe the filter, and which are what is returned by the filter design functions. This works because, since similarly to continuous time filters, the coefficients for the system function and for the corresponding difference equation are the same).

See *help filter* in Matlab and ask if you are not sure how to use it, but it is really straight-forward. You just supply the two vectors of coefficients (being sure to keep the order straight of which vector is the numerator and which is the denominator) and the input to *filter*, and it returns the filtered output.

Designing and filtering digital filters directly

There are many ways to design digital filters directly from frequency specifications in the discrete frequency domain. You can learn more about this in the Linear Systems and Digital Signal Processing classes later on. Here we will experiment with one design, known as "Parks-McClellan" (after the people who first derived this method) or "equiripple" (you will soon see why this name is appropriate) filters. The Matlab routines we will use for the design are *firpmord*, to find the required order and other key parameters from the specifications, and *firpm* to actually design the filter from those parameters.

If you run *help firpmord* you will see that it requires four sets of input variables. The first two parameters are 1) a vector of key "break" frequencies and 2) a vector of desired amplitudes at those frequencies. As an example, for a low pass filter with a band pass edge at a frequency corresponding to $250\ Hz$ with a given sampling frequency and a stop band edge corresponding to $350\ Hz$, the first vector would be $[250\ 350]$ and the second one ¿]. **Why do you think the second vector is [1 0**]? Explain in your notebook, or ask if you don't understand. The third vector gives desired tolerances in each of those bands. Also note that if you do not supply a sampling rate as the fourth argument, the help says that the default value is 2. See lecture notes for more information.

The parameters returned by the order design routine, *firpmord*, are used directly as the input to the filter design routine *firpm*. Here there is only output variable (vector of coefficients) as this type of filter has only a numerator set of coefficients, in other words the denominator coefficient is equal to 1. In terms of the difference equation this implies that there are no "recursions" on past output values, just a direct weighting of the input signal. It turns out that this type of filter, known as Finite Impulse Response (FIR), has some significant advantages in certain situations. Again, ask if you are curious to learn more.

Exercise 1: Design a Parks-McClellan filter using these routines low-pass filter your signal. Use *freqz* to check your design; obtain the transfer function as a function of frequency and plot the magnitude (and

phase, if you want) of the transfer function. Note here you need to supply the number “1” for the input parameter “a”. Why do you think this is called an “equiripple” design? What do you notice about phase response? Can you explain the jumps in the phase response here?

Filter your ECG signal(s) with your Parks-McClellan design using *filter*. Here, as with *freqz* you need to specify the “1” in the input for the 2nd input parameter.

Try different design specifications for the Parks-McClellan design and record your results in your notebook. Note that if you make the filter too “good” in some sense (cutoff frequency too low, stopband ripple too small, transition too fast), it will begin to distort the signal. Ask if you have questions.

Exercise 2: Option 1: Using the same methods, design a 60 Hz notch filter to remove the 60 Hz interference. This is likely to result in signal artifacts, but see what happens and discuss. See lecture notes for more information.

Option 2: Refer to lab 12, section 2.5, and use this method to remove the 60 Hz interference (and any other frequencies that are particularly prominent, such as the harmonics of 60 Hz (120, 180, 240, …).

Exercise 3: Option 1: Use threshold detection (or another method of your choice) to count the heartbeats in a certain interval and determine a heart rate. Option 2: Use the fft of the heart signal to determine the heart rate, again using MATLAB to find the peak of the lowest frequency peak. Again, you can refer to lab 12 for information on how to get started with this option.