ME 333 Introduction to Mechatronics
Winter 2011
Follow-Up to Assignment 6: Digital Finite Impulse Response (FIR) Filters

There are many different ways you could have written the program for the assignment. For me, I probably would have defined four `float` arrays: `filter, signal, paddedsignal, result`, of lengths `MAXORDER+1, MAXLENGTH, MAXLENGTH+2*(MAXORDER), MAXLENGTH+MAXORDER`, respectively. The array `paddedsignal` would just add `MAXORDER` zeros to the beginning and end of `signal`. Then I would define a single function to do the filtering:

```
int doFilter(float *filptr, float *sigptr, float *result, int
fillength, int siglength, int timedelay)
```

that simply takes a pointer to `filter`, a pointer to `signal`, a pointer to the array where you want the result to be stored, and two integers, one giving the number of elements of the filter and one giving the number of elements of the signal. (These numbers should be less than the `MAX` values.) The final integer, `timedelay`, determines whether the output signal is causal (delayed) or not. The function first creates the elements of `paddedsignal`, with `fillength-1` zeros added to the beginning and end of `signal`, then runs two nested for loops, the outer one `siglength+fillength-1` times, the inner one `fillength` times, and fills in the results in the `result` vector.

The figures below were all generated in Matlab. Figures 1–3 refer to the programming assignment, while the remaining figures describe some other filters applied to the same signal.

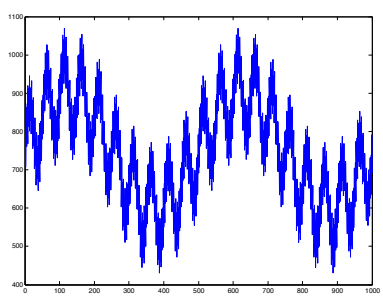

Figure 1: The original 1000-sample signal, with components at DC, 2 Hz, 20 Hz, and 400 Hz. This signal is stored in the variable `SIG` in Matlab. Sampling frequency is 1 kHz, so the Nyquist frequency is 500 Hz.

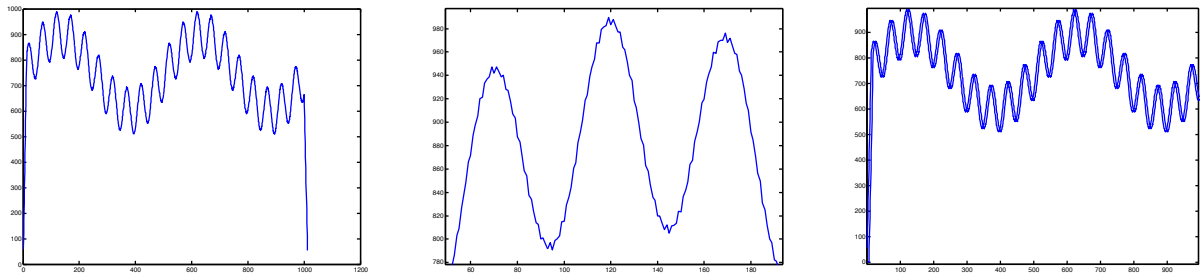

Figure 2: `maf=ones(13,1)/13; plot(conv(maf,SIG))`. **Left**: The result of a 12th-order (13-sample) MAF applied to (convolved with) the original signal. Since the original signal has 1000 samples, and the MAF has 13 samples, the filtered signal has 1012 samples. (In general, if two signals of length j and k are convolved with each other, the result will have length $j + k - 1$.) This is equivalent to first "padding" the 1000 samples with 12 samples equal to zero on either end (sample numbers -11 to 0, and 1001 to 1012), then applying the 13-sample filter 1012 times, over samples -11 to 1, then -10 to 2, etc., up to samples 1000-1012. This zero-padding explains why the signal drops to close to zero at either end. Another alternative would be to only apply the 13-sample filter 988 times, with signal sample numbers 1-13, 2-14, etc., up to 988-1000. This would shorten the signal by 12 samples. **Middle**: Zoomed in on the smoothed signal. **Right**: The result of the causal filter is the same as the result of the acausal filter, except delayed by six samples.

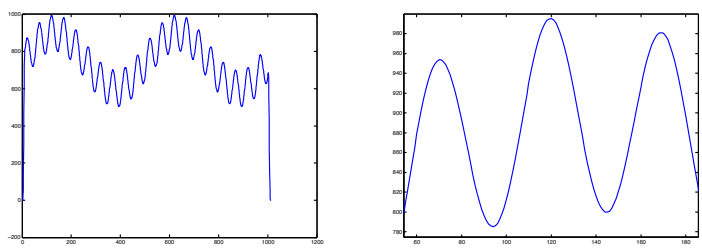

Figure 3: `lpf=fir1(12,0.2); plot(conv(lpf,SIG))`. **Left**: The signal smoothed by a 12th-order FIR LPF with a cutoff at 0.2 times the Nyquist frequency (using the fir1 convention of -6 dB at cutoff). **Right**: A zoomed in view, showing superior performance to the 12th-order MAF.

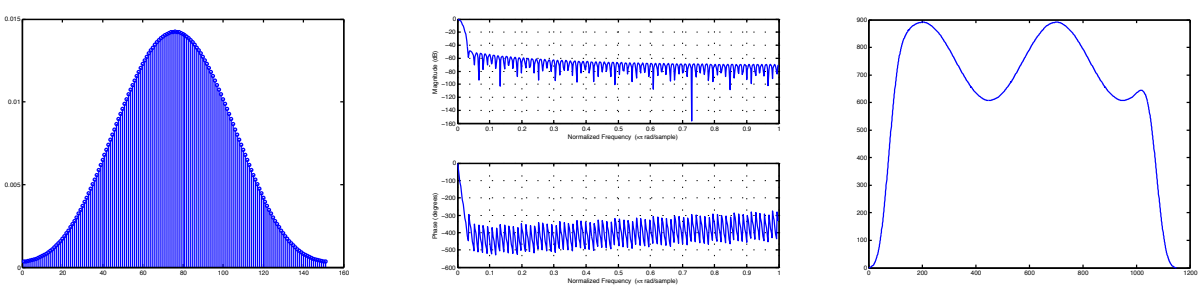

Figure 4: `b=fir1(150,0.01)`. **Left**: `stem(b)`. The coefficients of a 150th-order FIR LPF with a cutoff at 5 Hz (normalized 0.01). **Middle**: `freqz(b)`. The frequency response. **Right**: `plot(conv(b,SIG))`. The smoothed signal, where only the 2 Hz (normalized 0.004) and DC components get through.

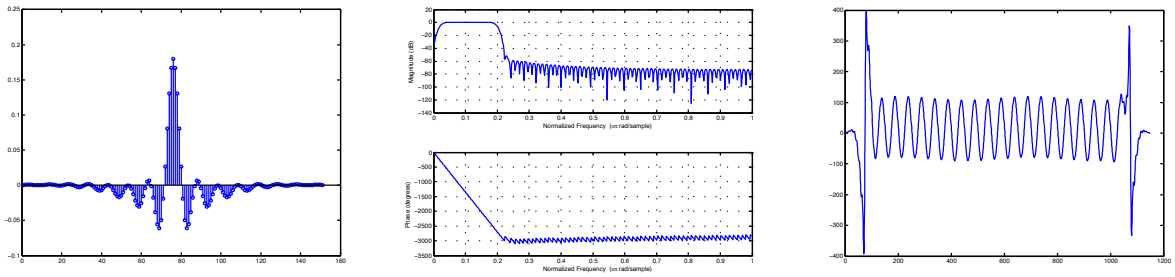

Figure 5: `b=fir1(150,[0.02,0.2])`. **Left**: `stem(b)`. The coefficients of a 150th-order band-pass filter with cutoffs at 10 Hz and 100 Hz (normalized 0.02 and 0.2, respectively). **Middle**: `freqz(b)`. The frequency response. **Right**: `plot(conv(b,SIG))`. The signal consisting mostly of the 20 Hz component, with small DC and 2 Hz components.

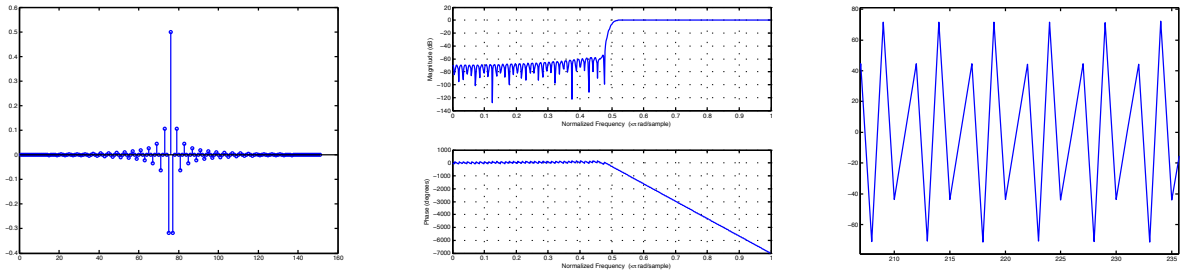

Figure 6: `b=fir1(150,0.5,'high')`. **Left**: `stem(b)`. The coefficients of a 150th-order high-pass filter with cutoff at 250 Hz (normalized 0.5). **Middle**: `freqz(b)`. The frequency response. **Right**: `plot(conv(b,SIG))`. Zoomed in on the high-passed signal.

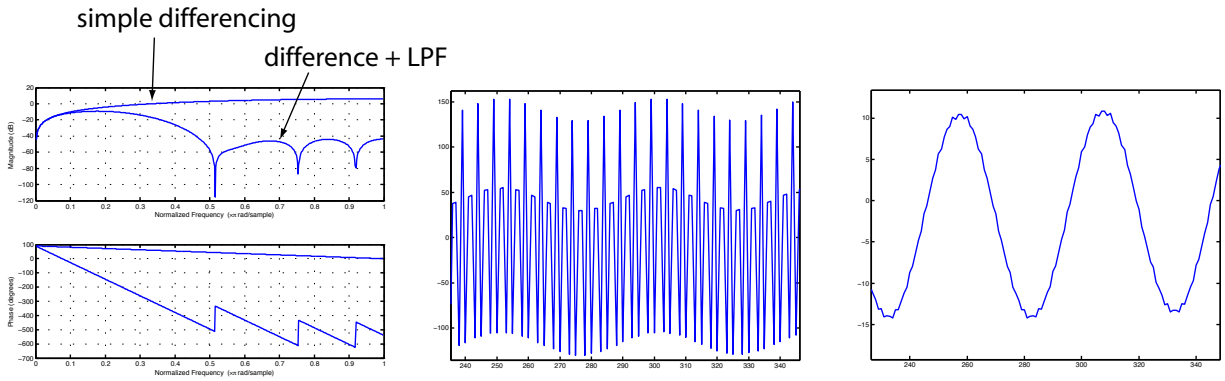

Figure 7: A simple differencing (or "velocity") filter has coefficients $b[0] = 1, b[1] = -1$, or written in Matlab, $b = [1 - 1]$. (Note the order: the coefficient that goes with the most recent input is on the left.) A differencing filter responds more strongly to signals with larger slopes (i.e., higher frequency signals) and has zero response to constant (DC) signals. Usually the signal "velocities" we are interested in, though, are those at low frequency; higher-frequency signals tend to come from sensing noise. Thus a better filter is probably a differencing filter convolved with a low-pass filter. **Left**: `b1 = [1 -1]; b2 = conv(b1,fir1(12,0.2)); freqz(b1); hold on; freqz(b2)`. This plot shows the frequency response of the differencing filter, as well as a differencing filter convolved with a 12th-order FIR LPF with cutoff at 100 Hz (0.2 normalized). At low frequencies, where the signals we are interested in live, the two filters have the same response. At high frequencies, the simple differencing filter has a large (unwanted) response, while the other filter attenuates this noise. **Middle**: `plot(conv(b1,SIG))`. Zoomed in on the signal filtered by the simple difference filter. **Right**: `plot(conv(b2,SIG))`. Zoomed in on the signal filtered by the difference-plus-LPF.

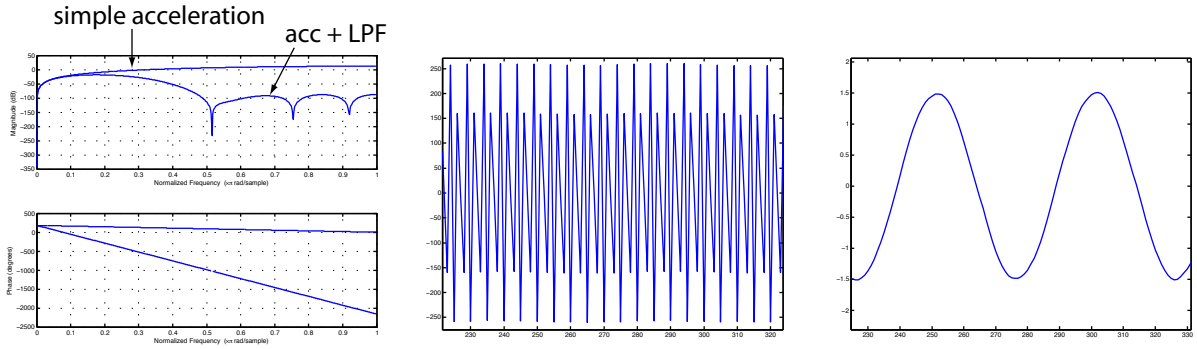

Figure 8: We can also make a double-differencing (or "acceleration") filter by taking the difference of two consecutive difference samples, i.e., convolving two differencing filters. This gives a simple filter with coefficients [1 -2 1]. This filter amplifies high frequency noise even more than a differencing filter. A better choice would be to use a filter that is the convolution of two difference-plus-low-pass filters from the previous example. **Left**: `bvel=[1 -1]; bacc=conv(bvel,bvel); bvellpf=conv(bvel,fir1(12,0.2)); bacclpf=conv(bvellpf,bvellpf); freqz(bacc); hold on; freqz(bacclpf).` The low-frequency response of the two filters is identical, while the low-pass version attenuates high frequency noise. **Middle**: `plot(conv(bacc,SIG)).` Zoomed in on the second derivative of the signal, according to the simple acceleration filter. **Right**: `plot(conv(bacclpf,SIG)).` Zoomed in on the second derivative of the signal, according to the low-passed version of the acceleration filter.

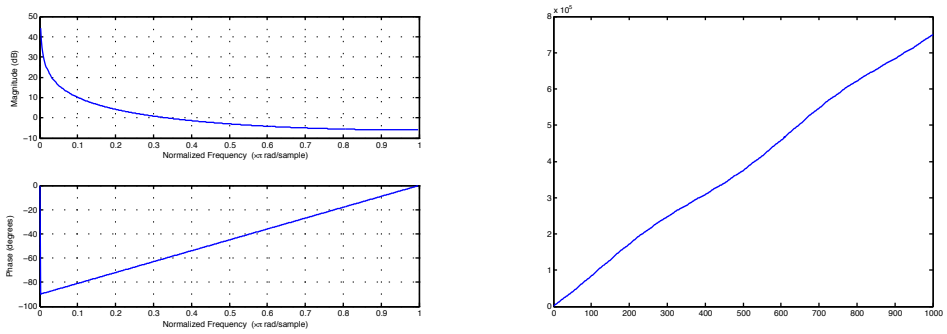

Figure 9: As a final example, let's generalize to IIR filters, a class that includes FIR filters. If we want to compute the integral of `SIG`, then our filter can be written a[0]*out[n] + a[1]*out[n-1] = b[0]*in[n], where a[0]=1, a[1]=−1, and b[0]=1. **Left**: `a=[1 -1]; b=[1]; freqz(b,a).` Note that the frequency response of the integrator is infinite to DC signals (the integral of a nonzero constant signal goes to infinity) and low for high frequency signals. This is opposite of the differencing filter. **Right**: `plot(filter(b,a,SIG)).` The `filter` command applies the filter with coefficients `b` and `a` to `SIG`. This generalizes `conv` to IIR filters. (We can't simply use `conv` for IIR filters, since the impulse response is not finite.) The upward slope of the integral is due to the nonzero DC term. We can also see the wiggle due to the 2 Hz term. It is basically impossible to see the 20 Hz and 400 Hz terms in the signal. .....

Now that you've learned a bit about filters, you can explore Matlab's Signal Processing Toolbox in more detail. Type `doc` to bring up the help window, click the arrow next to the Signal Processing Toolbox, go to Functions/Digital Filters, and check out FIR and IIR filter design, as well as GUIs/fdatool. Butterworth and Chebyshev IIR filters are among the most popular filter choices.