Engineering 253 – 2005
Professor Harvey Silverman
November 9, 2005
MATLAB Exercise #3
Design of a Sound-Level Meter

Last year, I had the class do the following design of a n A-Weighted sound-level meter. They used multirate filtering for the design and it was not a trivial problem. This year, I would like to have you design a similar sound-level meter, but use suitable DFT techniques to implement it. This should be simpler than the multirate design. For completeness, I have included the write-up for last year at the end of this document.

The block diagram of a typical seven octave Analog sound-level meter is shown in Figure 1. Each box labeled F(.,.) is a bandpass filter. There are usually three filters per octave and these are called third-octave filters. Remember that an octave is a doubling of the frequency, *i.e.*, an octave is the span from 1000Hz to 2000Hz. Third-octave filters for this octave would center at 1) 1000Hz, 2) 1259Hz and 3) 1585Hz. Thus the center frequency for each octave is easy to obtain from the octave number and "filter within octave" numbers as shown. After the filters, the power in each signal is accumulated for T seconds. "Fast" responses have T=125ms and the more usual "slow" responses have T=1s. The power measurements are then multiplies by a frequency weighting. These weightings, all shown in Figure 2, have been derived for different purposes, but the most widely accepted is the "A weighting that was selected to model the response of the human ear. The appropriate factor, e.g., for 1000Hz, 0dB = a 1.0 power factor, is then multiplied for each output (these can be derived from Figure 2). The outputs are then summed and put into a ratio against a reference. The standard reference is an equivalent pressure value of 0.000204 dynes/square centimeter. For this standard, a quiet location will be at about 55dB, and 90dB the level of sound that one needs to protect one's ears from if in this work environment for 40 hours a week.

DFT Method Implementation for 2005 Class

This year's class will try to make the device using DFT methods. This should be simpler than what had to be done last year. The first task is to select the proper DFT size and window, or maybe two sizes, one for a fast response and one for a slow one. Then you need to develop a way to accumulate the 21 third-octave filter values – or maybe not – color the results with the A-Weighted spectrum shown in Figure 2 and give the appropriate output representation. You will be asked to show your MATLAB system using a microphone with a standard PC sound card, and Ying Yu and I will compare the readings against those taken with a standard hand-held sound-level meter. i.e., results need to be in dBm defined above. Arpie will have a few microphones available if you want to test your system before showing it to us.

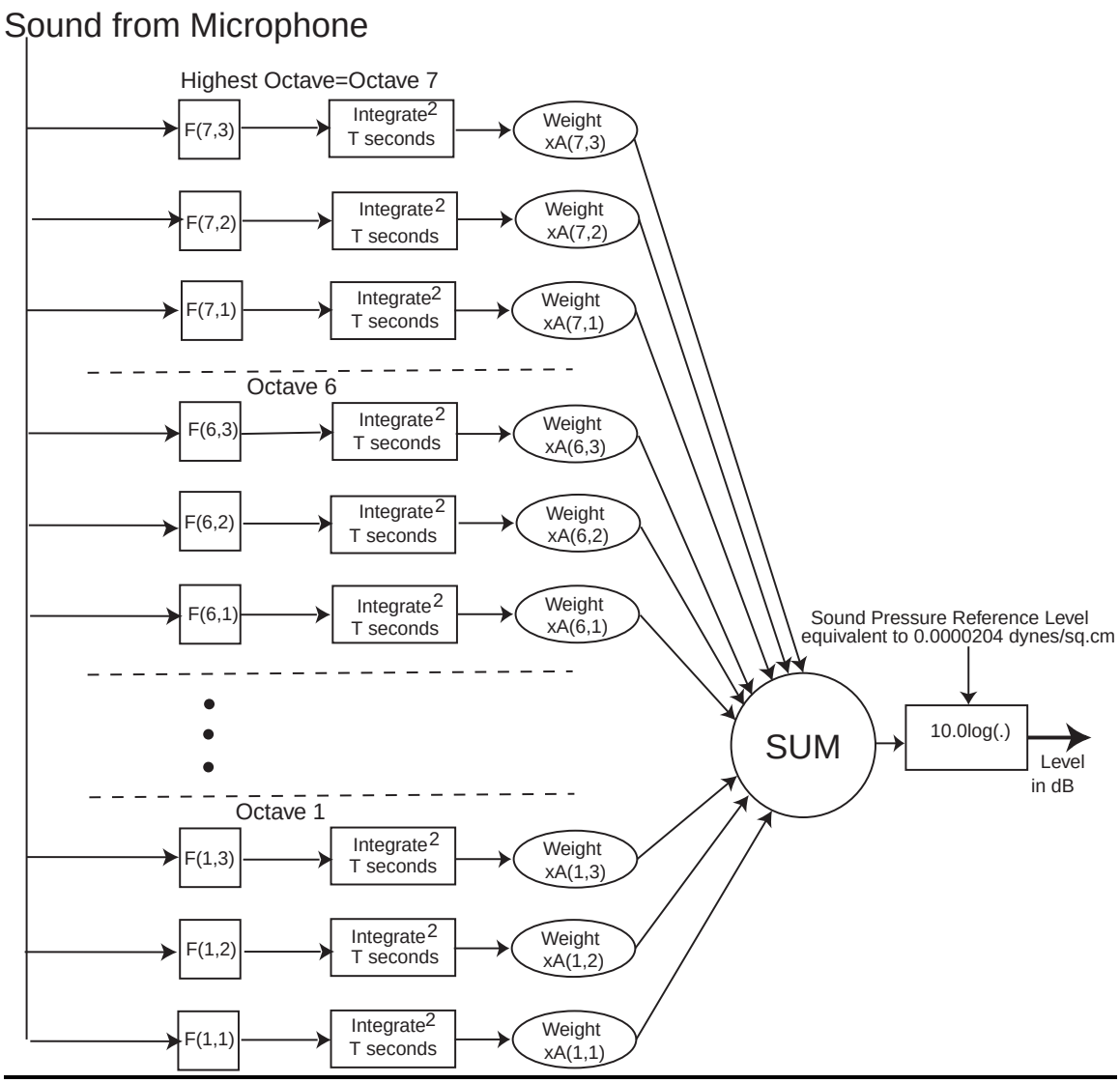

Figure 1
A Typical Analog Sound-Level Meter

You will also hand in a write-up that gives:

1) A description of your design telling how you included all the essentials using the DFT design.
2) An analysis of the cost in MADDS if the design were implemented using the FFT algorithm.
3) Does your design work in real time? Why or why not?

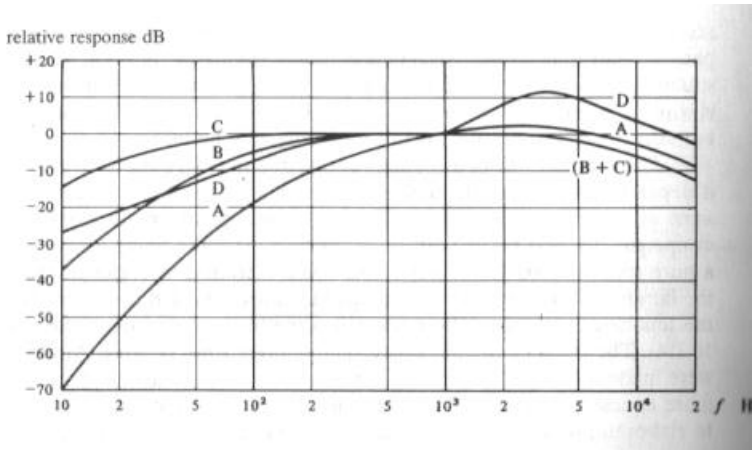

Figure 2

Definition of Weightings for Sound-Level Measurements

Multirate Method of 2004 Class

In class we have discussed optimal, linear-phase FIR filter design and the application of multirate filtering to reduce the cost of implementing these filters. In this MATLAB exercise, you will design a third-octave filterbank that might be used in a typical sound-level meter.

A Better Way to Make an FIR Digital Sound-Level Meter

Your job is to develop a sound-level meter in MATLAB that is a bit more clever than just sort-of copying the analog design above. Note that if you did copy the design above, you would have to design 21 bandpass filters – 21 applications of the MATLAB program *fdatool*. The lengths of the filters would double for each octave in order to have controlled overlapping performance of the filters. That is, for optimal FIR filters, relative to a fixed sampling frequency, filter transition widths will halve for each octave which results in doubling the filter lengths. Over seven bands, this would imply that if octave 7 needed a 101 tap filter, octave 1 would require a filter 64 times longer, about 6400 taps. A filter in octave 1 is 64 times narrower than a filter in octave 7. First, it is hard for the optimizing program to fit a polynomial of order about 1600, so this could "overload" *fdatool* and, even if not, take a long time for each design. Worse even, the lowest frequency filters of over 6400 taps, would take a long time to compute. For 44.1kHz sampled data, 44,100x6400x3(filters in the lowest octave) MADDS implies about 1.0GFlops. In MATLAB that will take a long time!! In fact, the number of MADDS for this direct implementation is about 3(filters per octave) x (1 + 2 + 4 + 8 +16+32+64) x L (average length of the octave 7 filters) x S (sampling rate), or 381LS MADDS/second.

Another structure uses multirate filtering and looks like the design shown in Figure 3. Note some of the advantages of this:

1. Only four filters need to be designed, three bandpass and one lowpass
2. Decimation by 2 implies a **half-band** lowpass filter may be used. A half-band filter is symmetric about ¼ the sampling rate in its specification and of an odd number of taps. Half-band filters turn out to have zeroes at every even numbered tap and thus require half the implementation cost.
3. A recursive program can be used for each octave.
4. The cost of computing these filters is the same **per output point** for every octave. However, as the sampling rate is getting lower by a factor of two for each octave number, the cost is reduced by a factor of two. Thus the cost for the 21 filters is
5. L(average length of filters) x 3(filters/octave) x(1+1/2+1/4+1/8+1/16+1/32+1/64) x S(sampling rate). This is just a tiny bit below 6LS MADDS/second, a saving of about 64. in computational cost.
6. An additional cost for the decimation is small. Six applications of the filter F(4) which is a half-band filter for 80dB down in passband and stopband is about L MADDS per application. Thus the cost is L x (1/2+1/4+1/8+1/16+1/32+1/64) x S, or slightly fewer than LS MADDS/second.

The saving in design, programming and in computation is considerable, so this is a much better way to do the sound-level meter if one is to use FIR filters.

Step-by-Step – Here's What to Do

1. Select a sampling rate. The best would be 25,600samples/sec, as that would make octave 1 start at 100Hz, i.e., the octaves start at 100, 200, 400, 800, 1600, 3200, 6400Hz. However, you can choose any rate that you can get sound data into a PC. 44.1kHz is probably a bit high and the numbers are ugly, but it will sort of work. For 44.1kHz sampling, octave 7 would be 11,025-22,050, but octave 1 only goes down to 344Hz or so. Maybe 22,050Hz would be better as a starting point. State you sampling rate and why you selected it.
2. Design three bandpass filters having N odd that cover octave 7. Stopband attenuation is -40dB and passband ripple must be at least -40dB. What is meant here is clarified in Figure 4. There has to be some overlap of the filters and the 21 ultimate filters need to cover all the frequencies. Note in Figure 4 that the adjacent octave filters will have the same shape as those shown, but either be twice as narrow (on the left) or twice as wide (on the right). Show the composite for the highest octave of the magnitude of the frequency response (as in Figure 4). Also give the lengths of each of the filters and all the design parameters.

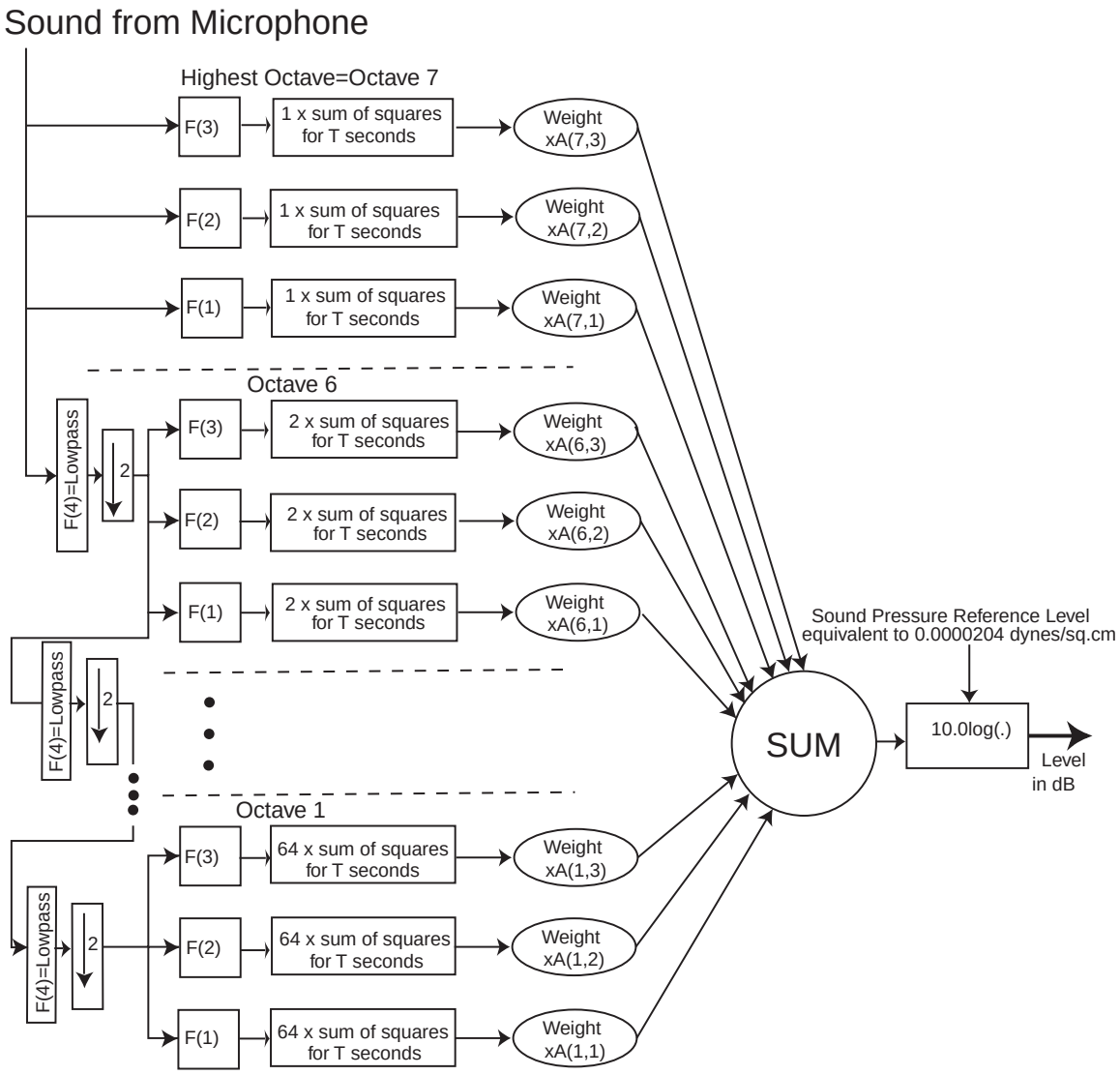

Figure 3

Multirate Filterbank for Sound-Level Meter

3. Design the half-band lowpass filter for -80dB stopband attenuation and -80 dB of passband ripple with N odd. Check that all even numbered points are(essentially) zero. Hand in the time-domain plot and the magnitude of the frequency response as well as your parameters for the design.
4. Implement the sound-level meter of Figure 3 in MATLAB. Assume the data you are measuring is of length at least one second long and that we shall accumulate for the "longer" period of T=1second. The input data is a vector of length for a few seconds at your sampling rate and will be called SOUND. You must include your MATLAB code for the sound analyzer in the write-up.

5. To get a final numerical output in dB, you need to have a reference point. We have no idea where 0.000204 dynes/square centimeter would be in our space, so let's make a practical choice that would allow us to be off by only some fixed factor that would need to be added in. We assume the 16-bit scale for the A/D converter on the sound card produces integer results in the range [-32, 768, 32767]. Our best choice of referent, then, is unity. That is, the output value O is $O[n] = 10.0 \log_{10}(\frac{Normalized, weighted\ \_\ power}{1})$.
6. Test your system on three or four different recorded sounds of different levels. Does it make sense? Describe your experiments.

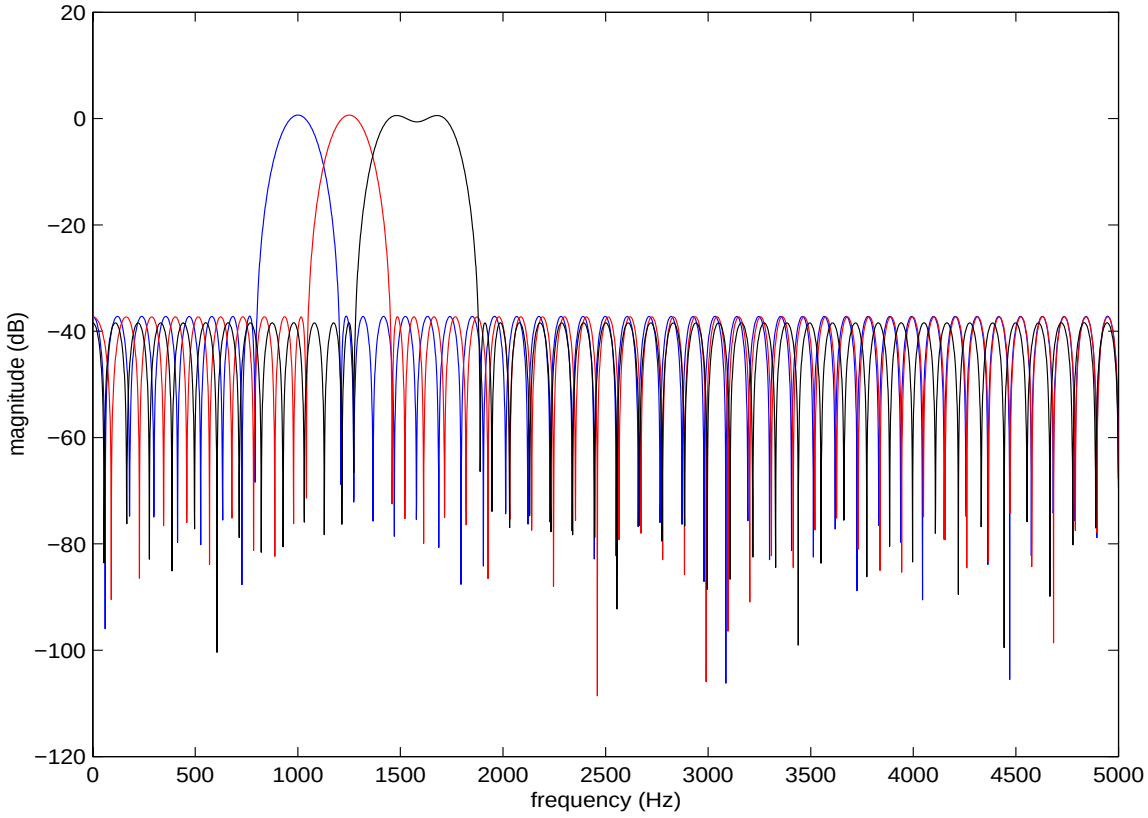

Figure 4

Coverage of the Octave from 1000Hz-2000Hz (designed with 10kHz sampling) for Stopband Attenuation of -40dB and Passband Ripple of -40dB.