EN 253: Matlab Exercise #3
Design of a Sound Level Meter

Douglas R. Lanman
29 November 2005

1 Introduction

The goal of this Matlab exercise is to design an A-weighted sound level meter. Although previous years have utilized multirate filtering to reduce the computational overhead, this write-up will focus on a more direct implementation using the Fast Fourier Transform (FFT).

Using the Data Acquisition Toolbox in Matlab, I have implemented a basic A-weighted sound level meter. This implementation allows the user to acquire samples from the sound card in real-time at any sampling rate supported by the hardware. The FFT algorithm is used to estimate the frequency spectrum of a windowed set of samples. The frequency spectrum is weighted using a closed-form expression for the A-weighting filter. The average signal energy is estimated, in the frequency domain, using Parseval's relation. A simple graphical user interface is provided to view real-time estimates of the signal level and the frequency spectrum. In addition, an off-line implementation is provided, as well as a simple signal generator for determining the calibration constant.

In Section 2 the basic design is described, including the specific FFT length and weighting filter equation. The details of the real-time implementation are presented in Section 3. In Section 4, two examples are presented: (1) off-line calibration with training data and (2) real-time calibration using commercial sound level meters. Finally, in Section 5 the computational cost (in MADDS) is presented.

2 Sound Level Meter Design

As previously discussed, my sound level meter uses a Discrete Fourier Transform (DFT) to determine the frequency spectrum of a windowed segment of the input audio stream. For efficient implementation, I use Matlab's FFT and a window length that is a power of two. The specific design choices as well as implementation details are discussed in the following sections.

2.1 Sampling Rate and DFT Length

As discussed in the exercise handout, commercial sound level meters usually have both a "fast" and a "slow" response rate. Typical fast responses are on the order of 125 ms, whereas slow responses are approximately 1.0 s. These rates can be used to select the appropriate DFT (or FFT) lengths. Recall that the DFT can be used to approximate the DTFT [See Mitra, pp. 240-241]; the N-point DFT equation was presented in Matlab Exercise #3 and [3] and is given by

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \ .$$

As discussed in class, the DFT can be used to approximate the continuous DTFT at a set of uniformly spaced samples. In order to determine the corresponding frequency of each DFT sample $X[k]$, we can apply the familiar equation

$$\Delta f = \frac{f_s}{N} = \frac{1}{NT} \tag{1}$$

where f_s is the sampling frequency, N is the length of the input sequence, and Δf is the frequency resolution. This relation will be used in Section 2.2 to determine the A-weighting filter coefficients.

Since our goal is to achieve a real-time implementation of the sound level meter, the FFT algorithm is used to evaluate the DFT. To date, we have only discussed the Decimation-In-Time (DIT) Radix-2 FFT. (As a result, I will limit my analysis to this form of the FFT – although Matlab supports arbitrary sequence lengths with `fft`.) The fundamental property of the Radix-2 FFT is that the input sequence must be a power of two, i.e. $N = 2^m$, for m a positive integer. Given a sampling rate f_s and a observation interval Δt, the *minimum* number of samples is given by

$$N_{min} = \lceil f_s \Delta t \rceil \ .$$

Note that, in general, this number is unlikely to be a power of two. As a result, we need to determine the next power of two greater than of equal to N_{min}. This can be done in Matlab using the `nextpow2` function.

As will be discussed in the next section, the A-weighting filter is designed to model the response of the human ear. As a result, it has a high-frequency cutoff of about 20 kHz. In order to support a Nyquist rate of 22.05 kHz, a default input sampling rate of 44.1 kHz is used. As will be discussed in Section 3, however, this implementation also allows any sampling rate supported by the sound card.

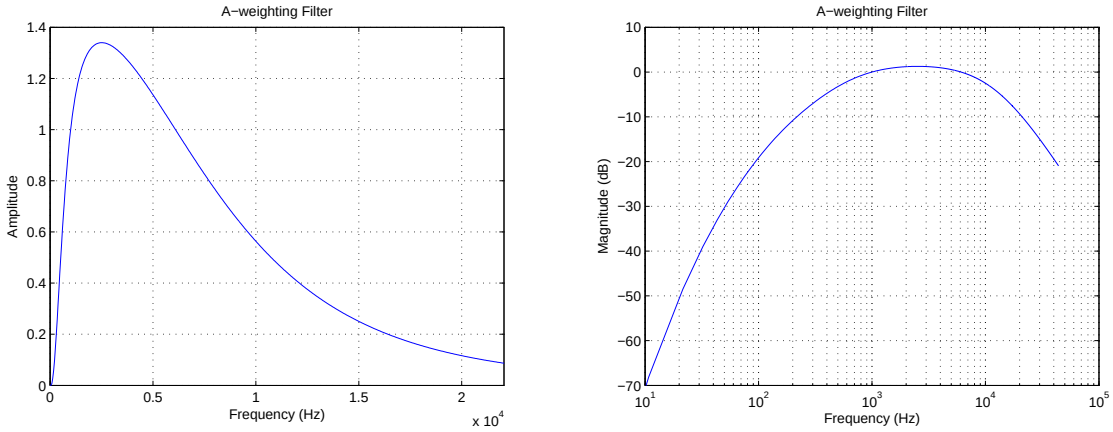
At this point, we have discussed the procedure used to select both the sampling rate (44.1 kHz) and the FFT window size. The specific window lengths N used for 44.1 kHz are shown in the table below.

Response Type	Desired Period	Minimum Length	Actual Length (N)	Actual Period
Fast	125 ms	5,513	8,192	186 ms
Slow	1.0 s	44,100	65,536	1.486 s

Note that, since the response times are not required to be *exactly* those stated in the exercise handout, we are free to choose them such that the window lengths are a power of two. Also note that the FFT lengths are short enough for real-time evaluation – as will be discussed in Section 5.

2.2 The A-weighting Filter

Although the exercise handout describes the design of 1/3-octave filters, since we are computing the FFT directly **it is not necessary to use the 1/3-octave filters**! As described, these filters are used to divide the frequency spectrum into several discrete frequency "bins" – a process automatically achieved using the FFT. As a result, we can apply the A-weighting directly to each FFT frequency bin. This simplifies the implementation and allows increased frequency resolution.



(a) Linear scale. (b) Decibel scale.

Figure 1: A-weighting Filter.

Figure 2 in the exercise handout shows several weighting schemes used in sound level meters, including the A, B, C, and D filters. As described in [1], the A-weighting filter was designed to mimic the response of the human hearing system. That is, the frequency attenuation of the A-weighting filter corresponds to an empirical average obtained across a broad sample of perceptual experiments. For this write-up, however, it is sufficient to state that the A-weighting is typically used in many commercial sound level meters and will be the only weighting scheme used in this implementation.

Since the problem statement does not provide a closed-form solution of the A-weighting curve, I searched the Internet for one. In general, the A-weighting curve can be obtained from international standards organizations, e.g. the American National Standards Institute (ANSI). For this implementation, however, I used a filter description available in [1].

$$\alpha_A(f) = \frac{(3.5041384x10^{16})f^8}{(20.598997^2 + f^2)^2 * (107.65265^2 + f^2) * (737.86223^2 + f^2) * (12194.217^2 + f^2)^2}$$

The filter response $\alpha_A(f)$ is shown in Figure 1. Note that the decibel-scale plot agrees with that shown in the exercise handout. Note that the A-weighting filter has a peak response near 3000 Hz (and rapidly decreases for lower frequencies). The attenuation gradually decreases from 3000 Hz to 22.05 kHz. As previously discussed, this is a crude approximation of the human hearing system – allowing us to quantify the level of sound sources with respect to a standard "human" observer.

Using Equation 1, we can determine the frequencies of each FFT sample $X[k]$. These frequencies $\{f_k\}$ can be used to evaluate the A-weighting filter coefficients using the previous equation. The A-weighted FFT samples are then given by

$$X_A[k] = \alpha_A(f_k)X[k], \text{ for } f_k = k\Delta f \ .$$

2.3 Calculating the Sound Level in dBA

At this point, we have produced an A-weighted frequency spectrum. In order to determine the signal level in dBA (A-weighted decibels) we must integrate the total signal energy over

either the fast or short response intervals. Conveniently, we have chosen the FFT-length to be equal to the response interval, so the "integration" is performed automatically by simply calculating the energy within the data window.

In general, the energy of a signal is defined as the sum of the squared magnitudes of the time-domain samples. Note that we cannot use this definition directly since our A-weighting has been applied in the frequency domain. In this case, we must apply Parseval's relation to estimate the signal energy in the frequency-domain. Otherwise, we would have to compute the inverse FFT and evaluate the filtered signal energy in the time-domain. By computing the signal energy in the frequency domain, we can significantly reduce the computational complexity of the sound level meter.

Recall that Parseval's relation is given by [Mitra, pp. 263, (5.115)] as

$$\varepsilon_x = \sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2 \ . \tag{2}$$

Also recall from class on 11/28/05 that for real-valued input signals, the DFT samples have complex-conjugate symmetry; that is

$$X[\frac{N}{2} + k] = X^*[\frac{N}{2} - k] \ .$$

Since the input time-domain samples must be real-valued, we find that the frequency spectrum must be symmetric about the $N/2$ sample. As a result, we can calculate the signal energy using only the first $N/2 + 1$ samples (of the A-weighted spectrum) as follows.

$$\varepsilon_x \approx \frac{2}{N} \sum_{k=0}^{N/2} |X_A[k]|^2, \text{ for } x[n] \in \mathbb{R} \tag{3}$$

Note that the strict equality doesn't hold, since we have counted the sample at $X[\frac{N}{2}]$ twice. As a result, Equation 3 can be used to estimate the total energy of the signal within the observation interval. Since the sound level meter supports several sampling rates and observations intervals, the average signal power should be used in the following analysis. Given an observation interval of Δt seconds, the average instantaneous signal energy is given by

$$\tilde{\varepsilon}_x \approx \frac{2}{N\Delta t} \sum_{k=0}^{N/2} |X_A[k]|^2 \ . \tag{4}$$

Note that, by using the symmetry property of the frequency spectrum, we have reduced the number of required operations by nearly a factor of two (only half of the frequency samples are required to evaluate the signal energy).

As discussed in the exercise handout, to get a final numerical output in dBA (A-weighted decibels), we need to have a reference signal level. The standard reference pressure is 0.000204 dynes/cm^2. In general, we won't know what input voltage this reference pressure corresponds to. That is, what $\tilde{\varepsilon}_x$ corresponds to 0.000204 dynes/cm^2? Let's assume that the appropriate reference signal level is $\tilde{\varepsilon}_{ref}$; as a result, we have

$$\text{Signal Level in dBA} = 10 \log_{10} \left(\frac{\tilde{\varepsilon}_x}{\tilde{\varepsilon}_{ref}} \right) = 10 \log_{10}(\tilde{\varepsilon}_x) - 10 \log_{10}(\tilde{\varepsilon}_{ref}) \ .$$

Note that $10\log_{10}(\tilde{\varepsilon}_{ref})$ is a fixed constant, so we can express the signal level in dBA as follows.

$$\boxed{\text{Signal Level in dBA} = 10\log_{10}(\tilde{\varepsilon}_x) + C} \tag{5}$$

The calibration constant C can be determined from laboratory experiments, as discussed in Section 4.

In this section I have reviewed the basic design of my sound level meter. In general, we found that the FFT algorithm can be used to estimate the frequency spectrum of a windowed set of samples. The frequency spectrum can then be weighted using a closed-form expression for the A-weighting filter. Afterwards, the average instantaneous signal energy can be estimated, in the frequency domain, using Parseval's relation. Given a value of the calibration constant C, we can evaluate the signal level in dBA using Equation 5.

3 Implementation Details

As discussed, the FFT has been used to implement the DFT. In addition, the data window lengths have been chosen to be a power of two – allowing the Decimation-In-Time Radix-2 FFT algorithm to be used. As discussed in class and in Section 5, this algorithm is computationally efficient and allows real-time operation in Matlab using the `fft` routine.

In this section I will review the specific implementation details for both the off-line and real-time sound level meters. A summary of source code files is presented for the end user. In addition, specific insights regarding the use of the Data Acquisition Toolbox are provided for users interested in real-time operation.

3.1 Matlab Functions

Both the off-line and real-time implementations utilize several core routines. These functions include: (1) the A-weighting filter coefficients, from Section 2.2, and (2) the decibel level analyzer, from Section 2.3. The "core" m-files are tabulated below.

Function Name	Description
`filterA.m`	Generates an A-weighting filter.
`estimateLevel.m`	Estimates signal level in dBA.

In addition to these routines, the real-time implementation requires several addition m-files. These programs handle the creation of the display window, the data acquisition from the sound card, and user input. A brief description of each function is provided below.

Function Name	Description
`analyzeSignal.m`	Off-line evaluation of dBA level.
`SLM.m`	Starts the real-time sound level meter.
`initSoundCard.m`	Initializes the sound card.
`stopSoundCard.m`	Closes sound card device.
`initDisplay.m`	Initializes SLM figure window.
`updateDisplay.m`	Updates the SLM display.
`runCalibration.m`	Estimates the calibration constant.

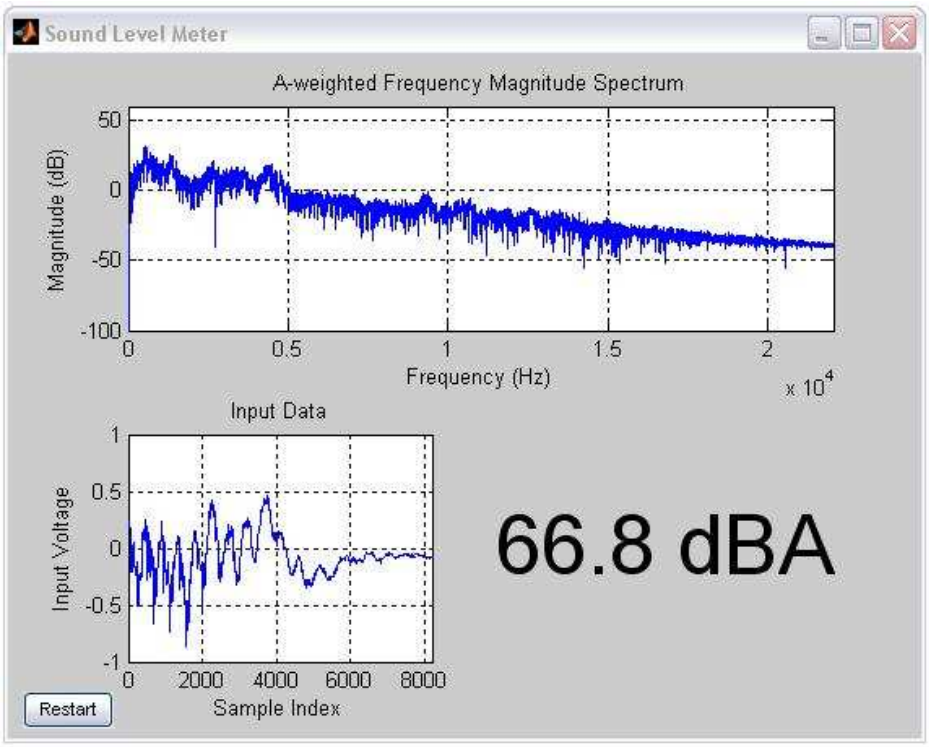


Figure 2: Real-time sound level meter display window.

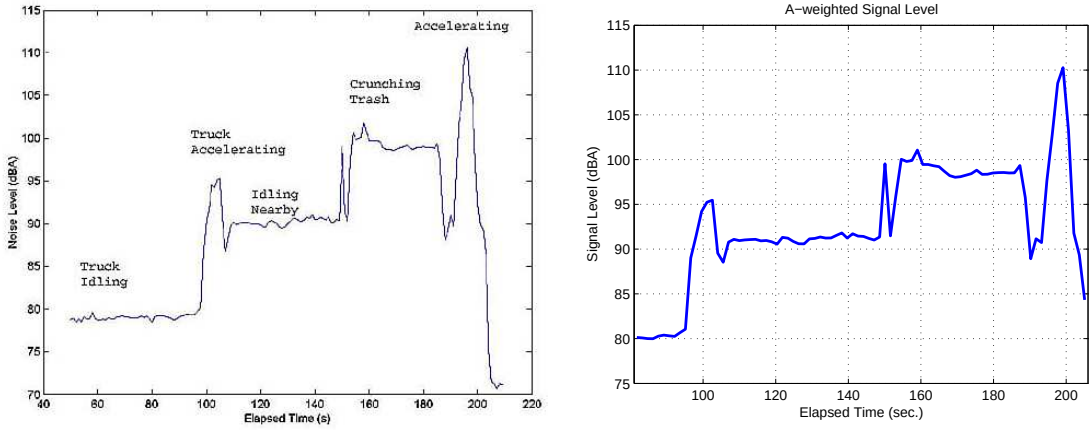
Note that all Matlab source code is included in the Appendix. Also note that the Data Acquisition Toolbox must be installed to use the real-time sound level meter.

3.2 Running the Off-line Sound Level Analyzer

Although my goal was to design a real-time sound level meter, I also implemented a basic off-line sound level analyzer with `analyzeSignal.m`. In general, given a vector of samples x sampled at frequency F_s, the command `analyzeSignal(x,Fs)` will produce a plot of signal level (in dBA) versus time. Note that the response type (i.e., "fast" or "slow") can be set on line 15 and the calibration constant C is set on line 19. The off-line sound analyzer is demonstrated in Section 4.1.

3.3 Running the Real-time Sound Level Meter

The real-time sound level meter is started by running the command `SLM` at the Matlab command prompt. (Note that the Data Acquisition Toolbox is used for real-time operation. Also note that the acquisition code is based on the `demoai_fft.m` demonstration program.) A typical output window is shown in Figure 2. Note that the output window displays: (1) the windowed input signal, at the lower left, (2) the A-weighted frequency spectrum, at the top, and (3) the current dBA level estimate, at the lower right. Data acquisition can be stopped/restarted by clicking on the button in the lower left. Note that the calibration constant, sampling frequency, and response type can be set on lines 12-27 in `SLM.m`. The calibration procedure is presented in Section 4.2.



(a) Signal level given by [2].

(b) Signal level using `analyzeSignal.m`.

Figure 3: Comparison of estimated dBA signal levels for "trash truck" example.

4 Experimental Results

In order to verify my sound level meter implementation, I used a test signal with known dBA levels. In addition, in order to calibrate the real-time analyzer, I used a pair of commercial sound level meters. In this section I will briefly summarize my experimental results using both the off-line and real-time analyzers.

4.1 Off-line Example

As was discussed in Section 2.3, we must empirically determine the value of the calibration constant C in Equation 5. This can be done off-line using a sample signal with known dBA levels. Searching the Internet, I found a sample of a trash truck with corresponding dBA measurements [2]. (The wave file is available online.) The signal level measurement provided by E. Carr Everbach is shown in Figure 3a. Using `analyzeSignal.m` with the sample signal, I obtained the signal level graph shown in Figure 3b. Note that these plots are similar, however small differences exist due to a lack of knowledge of the observation interval used by the author and other analysis parameters. Also note that the plot in Figure 3 was obtained by adjusting the calibration constant until the output matched the author's.

From this example, we see that my implementation of an A-weight sound level meter can recreate a known signal level measurement – verifying the implementation. In addition, this example highlights the importance of the calibration constant. Without a proper value of C, the two dBA levels would be biased by a fixed amount.

4.2 Real-time Calibration Procedure

The previous example illustrated how to analyze a signal off-line using training data with known signal levels. How can one obtain training data for the real-time signal level meter? One method is to use calibrated commercial sound level meters to calibrate the Matlab real-time meter.

In order to calibrate the Matlab sound level meter, I placed two commercial sound level

meters (obtained from Prof. Silverman) next to the input microphone. In addition, I positioned a speaker several feet away from the sound level meters and microphone. A simple signal generator, implemented in Matlab, was used to produce both white noise and pure-tone test signals (see `runCalibration.m`). By observing the output of the commercial sound level meters for a similar weighting and response type (i.e. A-weighting for the "slow" response), I was able to determine the value of the calibration constant C.

It is important to note that this calibration procedure is prone to errors, especially when the sound level meters and input microphones are not close to one another. In addition, since a directional microphone was used, the value of the calibration constant could be offset by some small amount. In general, however, this produce should be sufficient to achieve calibration accuracies within a few dBA.

5 Analysis of Computational Cost

The computational cost of our implementation can be measured in MADDS (i.e. multiplications and additions). Since the design is based on the FFT algorithm, I will begin by reviewing the computational costs of the Decimation-In-Time (DIT) Radix-2 FFT.

Recall from class on 11/23/05 that the DIT Radix-2 FFT requires $\frac{N}{2}\log_2 N$ complex multiplications and $N\log_2 N$ complex additions. Since one complex multiplication can be implemented with four real multiplications and two real additions, the total number of real multiplications is given by $2N\log_2 N$. Similarly, one complex addition requires two real additions, so the total number of real additions is given by $3N\log_2 N$.

In Section 2.1 we showed that a 8,192-point FFT was required for the fast response rate and that a 65,536-point FFT was required for the slow response rate. Using the previous set of equations, we can compute the total number of real multiplications and real additions required to evaluate the DIT Radix-2 FFT for a single data window. Computational costs are tabulated below for the 44.1 kHz sampling rate.

Response Type	Period (Δt)	Length (N)	Real Multiplications	Real Additions
Fast	186 ms	8,192	212,992	319,488
Slow	1.486 s	65,536	2,097,152	3,145,728

Note that the number of real additions is greater than the number of real multiplications by a factor of N. Let's assume that our processor can perform a single Multiply-Add (MADD) operation within a clock cycle. As a result, we can use the number of real additions as an upper bound on the number of MADDs required per output point (where each output point corresponds to the signal level measurement for a length-N data window). If we update the signal level measurement every Δt seconds (equal to the response rate), then we can obtain the number of MADDS per second as

$$\text{MADDS per second} = \frac{3N}{\Delta t}\log_2 N = 3f_s \log_2 N$$

where f_s is the sampling rate. For a sampling rate of 44.1 kHz, we find that 1,719,900 MADDS/sec and 2,116,800 MADDS/sec are required for the fast and slow response rates, respectively.

Using a 3.2 GHz processor, the following computational loads are predicted for FFT evaluation.

Response Type	MADDS/sec	3.2 GHz Processor Load
Fast	1,719,900	0.0537%
Slow	2,116,800	0.0662%

Note that these values are only rough estimates, however it is clear that the DIT Radix-2 FFT allows real-time implementation of our sound level meter! In addition, since the computation load is negligible, we can update the sound level estimate more frequently than the response rate Δt using a sliding window. As result, the real-time display presented in Section 3 can be updated at a rate greater than the response time.

At this point we have evaluated the computation cost of the standard DIT Radix-2 FFT presented in class. It is important to note that we could also use the fact that the input signal is real-valued to further reduce the computational costs of evaluating the FFT. In this case we could use a single complex-valued FFT to evaluate two real-valued FFTs (see class notes on 11/28/05). Regardless of this fact, we find the FFT lengths required by this implementation are achievable in real-time using modern processors.

In addition to the FFT, my design requires several additional real-valued multiplications and additions to implement: (1) the A-weighting filter, (2) Parseval's relation, and (3) the dBA signal level measurement. Note that, using Equation 5, a single addition is required to find the signal level (once the mean energy has been estimated). In addition, the signal level estimation requires evaluating a logarithm (typically using a table lookup). As a result, we will ignore the computational cost of Equation 5 since it is negligible in comparison to the FFT cost.

The A-weighting filter only has to be applied to the first $N/2 + 1$ frequency samples, since those are the only values used in evaluating Equation 4. This requires $N/2 + 1$ real multiplications. Similarly, the evaluation of Parseval's relation, using Equation 4, requires $N/2 + 1$ real-valued multiplications (to compute the squares) and $N/2$ additions to evaluate the summation. In either case, these operations are order N and are also negligible in comparison to the FFT requirements.

In conclusion, we find that the overall order of the sound level meter implementation is given by $O(N \log_2 N)$, corresponding to the FFT evaluation. Although additional operations are required to apply the A-weighting filter and determine the signal energy, these operations are $O(N)$. As a result, we find the the simplicity of the DIT Radix-2 FFT allows for real-time signal level estimates in Matlab.

6 Conclusion

In this write-up we have presented the design of a basic A-weighted sound level meter. The Fast Fourier Transform (FFT) is used to efficiently evaluate the DFT of a windowed segment of input samples. The frequency spectrum is modified by the A-weighting filter, mimicking the human hearing response. Afterwards, the signal level is measured in the frequency domain by applying Parseval's relation and symmetry properties of real-valued signals.

Using Matlab's Data Acquisition Toolbox, this implementation allows real-time analysis. As presented in Section 5, the Decimation-in-Time Radix-2 FFT is computationally inexpensive (on modern processors). As a result, real-time operation of a FFT-based sound level meter is achievable.

As presented in Section 4, the calibration constant C (from Equation 5) must be determined empirically. Commercial sound level meters were used in this write-up to calibrate the Matlab implementation. In general, one must recalibrate the system whenever a new microphone is used or the gain levels are adjusted in the system settings or in hardware.

References

[1] U. Beis. Weighting filter set. `http://www.beis.de/Elektronik/AudioMeasure/WeightingFilters.html`.

[2] E. C. Everbach. Noise quantification and monitoring. `http://www.swarthmore.edu/NatSci/sciproject/noise/noisequant.html`.

[3] S. K. Mitra. *Digital Signal Processing: A Computer-Based Approach*. McGraw-Hill, New York, NY, 2006.