

MULTI CHANNEL DATA ACQUISITION AND SIGNAL PROCESSING USING MATLAB

S. Bilawchuk, M.Sc., K.R. Fyfe, Ph.D., P.Eng.

aci Acoustical Consultants Inc.,

Suite 107, 9920-63 Ave, Edmonton, Alberta, Canada, T6E 0G9. www.aciacoustical.com

Published in Canadian Acoustics 2003 Acoustics Week In Canada Proceedings Issue

1. INTRODUCTION

With recent advances in digital data acquisition, multi-channel data acquisition systems for acoustics and vibrations are more commonplace. There has been an industry-wide shift from closed box systems with limited configuration options to PC based systems with numerous software driven data acquisition choices. These systems, however, are still relatively expensive and invariably contain a certain degree of constraints for both data acquisition and signal processing. Utilizing engineering computational program software such as MATLAB [1,2] enables individuals to customize their multi-channel data acquisition systems and use a variety of external hardware, all for less money than off-the-shelf systems. This paper is intended to discuss some of the highlights and pitfalls with using MATLAB for digital data acquisition and signal processing.

2. DATA ACQUISITION

The Data Acquisition Toolbox in MATLAB is designed to work with several hardware devices and offer complete user control over virtually all data-acquisition parameters. Variables such as sample rate, sample duration, and triggering can be easily controlled. Probably the simplest data acquisition device is the soundcard installed on most PC's. This can be used to acquire two channel data, with a sample rate up to 44.1 kHz with 16-bit resolution. The two most important things to be aware of with the sound card are the fact that it is A/C coupled (i.e. high pass filtered at about 2 – 3 Hz), and that it has preferred sample rates (i.e. 8 kHz, 10.025 kHz, 22.05 kHz, and 44.1 kHz). Acquiring data at sample rates other than these produced unpredictable/un-expected results. It is also important to consider the ability to sample dual channels simultaneously. Tests with a common 16-bit SoundBlaster sound card showed that the time lag between the acquisition on each channel translates to a much higher frequency than the 44.1 kHz maximum sample rate, and thus will not cause a problem for signal processing between the two channels.

Various other data acquisition cards are supported by MATLAB. All work for this paper was done using a Measurement Computing PC-CARD_DAS16/16-AO which has either 16 single-ended (common ground) or 8 differential analog inputs as well as several outputs and digital connections. The connections to the card (single or double) are VERY important and dependent on the type of transducers being used and the type of measurements made

(consult the user manual). Given the varied types of transducers used for typical acoustic/vibration sensory work, it was found that differential connections were appropriate. Although the card has a maximum sample rate of 200 kHz for a single channel, as the number of channels increases, the maximum sample rate goes down by (essentially) the ratio of channels. This is because the card uses a multiplexer to route each channel to a single A/D converter. As the number of channels increases, this process slows down. More expensive cards have multiple A/D converters, which allow for maximum sample rates in each channel regardless of the number of channels.

Using MATLAB, the data is acquired as a large matrix for each record block. The block length is dependent on the sample time and duration. Although these can be set to virtually any value, it is common to use radix 2 (2^n) numbers to take advantage of faster FFT algorithms [3]. Once the data is acquired, it must be stored or processed. Refer to the Appendix for sample data acquisition code. The variable t stores the time vector for the entire sample length, and the variable d has the data stored in a matrix with 1 column per channel.

As with any data acquisition system, an appropriate analog anti-aliasing low-pass filter is required. There are numerous filter choices available with specific data acquisition needs dictating which filter is appropriate. For the purposes of this paper, an external, switched capacitor filter was used. One key advantage to this filter is that the cutoff frequencies are controlled digitally by a variable clock frequency. This can be used in conjunction with the variable frequency output of some data acquisition cards to provide an infinitely variable set of frequencies. For example, when choosing to acquire the data at a particular sample rate, the computer program could automatically select the appropriate filter cutoff frequency and adjust it accordingly.

3. SIGNAL PROCESSING

Once the data has been acquired, it needs to be processed to obtain the desired form. MATLAB has various functions/operations built in for signal processing. Windowing, FFT analysis, cross and auto power spectrum calculations, coherence, averaging, L_{eq} measurements, integration and differentiation are just some of the key operations which can be done. Each of these operations are relatively simple, as illustrated in the sample code for calculating the cross spectrum and coherence between two channels, in the Appendix.

Similar operations such as FFT, windowing, and averaging require equivalent amounts of code. The key advantage is that the entire process is completely custom and additional features (such as additional channels) are minimal to incorporate.

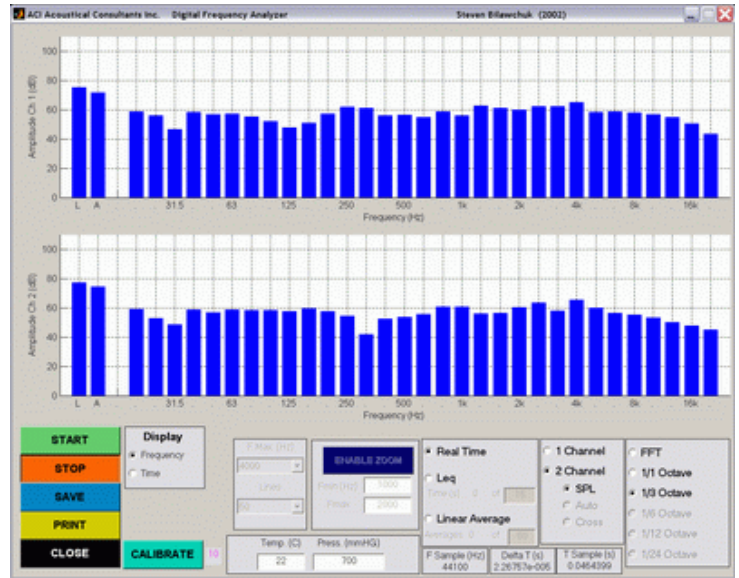
One important aspect to consider when using MATLAB is the process involved in acquiring the data, processing it, and displaying it. For a “real-time” application each of these processes needs to be done in succession before the next sample block can be obtained. Current versions of MATLAB do not allow for simultaneous operations such as acquiring the next signal while the previous one is being processed. Thus, the “real-time” capabilities of the entire process are limited by the computer processor speed and, more importantly, the efficiency of the code. Most applications have been found to be essentially “real-time” but the process does slow down as more channels are added and more complicated computations are performed. For most measurements, where long duration averaging is required, this has not been a concern. If a continuous data block is required, the entire process can be expedited by not analyzing and graphing the results for each data block sampled and waiting until the end.

4. GRAPHICAL USER INTERFACE

One of the most important features that makes MATLAB viable for data-acquisition and signal processing is the ability to write programs with a graphical user interface (GUI). Having data acquisition parameters, and plotting displays in the common “Windows” format enables much more power, flexibility, and “user-friendliness”. This paper will not digress toward the various methods and pitfalls of creating GUI programs, rather the purpose is to

state that with a few extra steps, a more powerful program for data acquisition can be obtained.

Figure 1 shows a simple program written for multi-channel data acquisition. This program is available for download at www.aciacoustical.com



5. REFERENCES

- 1) MATLAB Data Acquisition Toolbox Version 2.0 Users Guide, The Mathworks, www.mathworks.com
- 2) sample MATLAB code can be downloaded at www.aciacoustical.com
- 3) Frequency Analysis, Brüel and Kjær. 1987, K. Larsen & Son A/S

APPENDIX

The following is a sample of MATLAB code used to acquire a single data block:

```
ai = analoginput('mcc',1);           sets up connection to data acquisition card
ch = addchannel(ai,[0 1]);          adds channels 0 and 1 resulting in a total of 2 channels
set(ch, 'inputRange', [-5 5]);      sets input range to ± 5V
set(ai, 'samplerate', 1000);        sets sample rate to 1000 Hz
ai.samplespertrigger = 1000*10;     sets sample length to 10 seconds
start(ai); [d,t] = getdata(ai);     starts data acquisition and stores data into variables d (data) and t (time)
stop(ai); delete(ai);              stops data acquisition and deletes data in "ai" before acquiring next sample block
```

The following is a sample of MATLAB code used to perform the FFT calculation on 1 of the 2 channels

```
Ch_1 = fft(d(:,1))/blocksize*2;     FFT for Ch1 with scale correction
Ch_1(1) = Ch_1(1)/2;                Correction for DC Offset
Ch_1 = Ch_1(1:(length(Ch_1)/2 + 1)); Storing only the positive frequency components
```

The following is a sample of MATLAB code used to calculate auto and cross spectra as well as coherence:

```
G_11 = G_11 + conj(Ch_1) .* Ch_1;   auto power spectrum for Ch1 (added to previous value)
G_22 = G_22 + conj(Ch_2) .* Ch_2;   auto power spectrum for Ch2 (added to previous value)
G_12 = G_12 + conj(Ch_1) .* Ch_2;   cross power spectrum between Ch1 & Ch2 (added to previous value)
G_21 = G_21 + conj(Ch_2) .* Ch_1;   cross power spectrum between Ch2 & Ch1 (added to previous value)
H1 = G_12 ./ G_11;                  Frequency response function (H1)
H2 = G_22 ./ G_21;                  Frequency response function (H2)
coh = real(H1 ./ H2);                Coherence between Ch1 & Ch2
```