DOI: 10.61782/fa.2023.1310

THE DEVELOPMENT OF A LABORATORY'S OWN ACOUSTIC SIGNAL PROCESSING SOFTWARE

John Laurence Davy[1,2*]

[1] School of Science, RMIT University, Melbourne, Australia
[2] Infrastructure Technologies, CSIRO, Melbourne, Australia

ABSTRACT

RMIT University discovered that the reverberation time measurement module of new commercial hardware and software was not as fully automated as it should have been. The author wrote Visual Basic for Applications software that fully automated the new commercial software. CSIRO discovered that the same commercial hardware and software sometimes produced reverberation times that were too long due to the software deciding that the decay had started before the sound was turned off. It was discovered that one of the reasons why this occurred was that the firmware random noise generator produced the same random noise each time it was started, which made decay curve averaging useless. The author was able to convince the commercial supplier to fix this problem. Another reason was that the software sometimes produced undefined levels and these undefined levels sometimes caused the software to think that the decay had started before the sound was turned off. The commercial supplier was unable to fix this problem. RMIT University has had problems when using linear averaging to measure reverberation time. These errors and inadequacies are some of the reasons why CSIRO has started the development of its own signal processing software which is described in this paper.

[*]***Corresponding author***: john.davy@gmail.com.

Keywords: *acoustic signal processing software, reverberation time measurement software, development of a laboratory's own software, access to computer source code.*

1. INTRODUCTION

These days most acoustic signal processing is conducted using software and/or firmware, partly because much of the old analogue signal processing hardware has failed with the passage of time. While some laboratories write some of their own acoustic signal processing software, many laboratories use commercial acoustic signal processing software for which they do not have access to the source code. This means that they are using "black box" software and cannot be certain that the software is doing what they think that is doing. Also, they cannot correct any errors that they discover in the software, modify the behavior of the software or add new features to the software. The need to use commercial software is forced on some laboratories because some suppliers of acoustic signal processing software do not release the details of the interfaces to their front-end hardware in order to force laboratories to use the supplier's acoustic signal processing software. CSIRO and RMIT University have discovered or become aware of several errors or inadequacies in commercial acoustic signal processing software. This is one of the reasons why the author has started to develop acoustic signal processing software for CSIRO for one commercial supplier's acoustical front-end hardware. He has also produced versions of this software which use soundcards as the front-end input. The main use of the soundcard version of the software is to interface to professional audio external USB soundcards.

2. FULL AUTOMATION OF COMMERCIAL REVERBERATION TIME MEASURING SOFTWARE

When RMIT University obtained new commercial acoustic signal processing hardware and software in 2004, they discovered that its reverberation time measurement module was not as fully automated as the previous firmware that they had been using previously. It was necessary to manually rename the result of each measurement and manually move each measurement result to a special location. Fortunately, the commercial software came with Microsoft Visual Basic for Applications (VBA). The author wrote a VBA macro which fully automated the reverberation time measurement process and enabled the reverberation time measurements results to be output as a Comma Separated Values (CSV) file or as a Microsoft Excel Spreadsheet file with a single button click.

3. GENERATION OF THE SAME RANDOM NOISE SIGNAL EACH TIME THE GENERATOR WAS STARTED

CSIRO discovered that its commercial acoustic signal processing software intermittently produced reverberation times that were too long. Further investigation showed that this was due to the commercial software deciding that the sound decay had started before the steady state sound had been turned off. Initially, it was not clear why this was happening. Eventually, it was discovered that there were two reasons for this behavior. The first reason was because the random noise generator was incorrectly producing the same random noise signal each time that it was started. This meant that the ensemble averaging of the decay curves that CSIRO conducted was not reducing the steady state ripple of the decay curve. This steady state sound ripple was sometimes large enough to convince the software that the sound decay had started before the sound was turned off. This occurred at low frequencies because the commercial software, like most software, used the same averaging time for all constant percentage bandpass filters. This averaging time had to be short enough to correctly measure the shorter reverberation times at high frequencies. The ensemble variance of the random noise sound pressure level is proportional to the inverse of the product of the statistical bandwidth of the bandpass filter (which is proportional to frequency) and the averaging time. Hence the steady state sound level ripple is greater at low frequencies.

The random noise was generated by the firmware in the hardware front-end. It was obvious that the reason for the same random noise signal, each time the random noise generator was started, was that the firmware was using the same starting seed for generating the random noise. Random noise generators usually use a different seed each time that they are started. This seed is usually obtained from the time outputted by a real time clock.

The author contacted the commercial equipment supplier, and they upgraded their firmware in their current range of measuring equipment. They were not going to upgrade the firmware in their previous range of measuring equipment because it was close to the end of its ten-year maintenance period after being discontinued. The author contacted them again and pointed out how serious the error was because it made ensemble averaging useless. The author also pointed out that international standards would have to be changed to include a warning about this fault. The commercial supplier then agreed to upgrade the firmware in their previous range of measuring equipment.

4. THE OCCURRENCE AND THE EFFECT OF "UNDEFINED" LEVELS

Unfortunately, even after the same random noise signal error was corrected by the firmware upgrade, reverberation times that were too long still intermittently occurred. It turned out that this was due to the occurrence of "undefined" levels in all frequency bands for a sequence of successive times. This was difficult to detect because the commercial software does not warn the user when undefined levels occur. The commercial software ensemble averages decay curves before calculating the reverberation time using the ensemble averaged decay curve. When "undefined" levels occur during the steady state sound period, the ensemble average decay curve produced by the commercial software surprisingly gradually increases above its expected steady state level before suddenly falling back to its expected steady state level when the "undefined" values stop occurring. It is this sudden fall that sometimes causes the commercial software to decide that the sound decay has started before the sound is turned off. CSIRO reported this fault to the commercial equipment supplier and supplied them with the configuration of their software that CSIRO was using. Unfortunately, the commercial equipment supplier was unable to reproduce this fault and it remains unfixed. CSIRO was surprised that the commercial equipment supplier was unable to tell them the source of the "undefined" levels, even if they could not tell CSIRO why the "undefined" levels were occurring.

5. PROBLEM WITH SHORT TERM LINEAR AVERAGING WHEN MEASURING VERY SHORT REVERBERATIN TIMES

RMIT University had problems when using commercial software with short linear averages to measure very short reverberation times in a very small reverberation room ("Alpha Cabin") with volume of 2.89 m^2. The decay curves had a very large ripple and the time intervals between successive linear integrals were surprisingly not uniform. This problem was only overcome by using exponential averaging, which is not desirable because if the exponential averaging time is not chosen correctly, exponential averaging can bias the measured decay rate. It seems that the commercial equipment supplier prefers exponential averaging except for long term L_{eq} measurements.

6. OTHER PROBLEMS

In a European round robin on sound absorption coefficient measurement conducted by ISO/TC 43/SC 2/WG 35, a highly respected standards laboratory obtained sound absorption coefficients that were clearly too low. After investigation, they concluded that this was due to the use of new "black box" hardware and software.
The author tests the performance of reverberation time measurement software by using it to measure the reverberation time of fast exponential averaging (time constant equals 1/8 s and averaging time equals ¼ s). The measured reverberation time should be approximately 1.73 s. In the early 1990's, the firmware in one piece of commercial equipment produced the wrong reverberation tine and non-linear decibel versus time decay curves. The firmware in another piece of commercial equipment produced the correct reverberation time, but decibel versus time decay curves that were linear except at the bottom end where they curved down to the bottom of the display range. This appeared to be an attempt to hide the background noise of the device.

7. SIGNAL PROCESSING SOFTWARE DEVELOPMENT WITH PYTHON

In 2020, a major commercial supplier of acoustical measuring equipment and software released an Open Applications Programming Interface (Open API) for their current range of acoustical measurement front-end hardware. Because of all the issues mentioned above CSIRO purchased a license for this Open API. Later, the commercial supplier made the license for this interface available for free when they released another firmware update for their front-ends. CSIRO asked for their money back, but were refused. However, the fact that the license was free meant that CSIRO did not have to purchase another Open API license when they obtained another front-end module.
The commercial supplier had made available the source code for a number of example programs using the Open API in a number of different software languages. However, the only example programs that performed any signal processing were written in Python. Hence the author's initial software development was written in Python. Python is available for free and its NumPy and SciPy modules have extensive signal processing capability, including Fast Fourier Transforms (FFTs) and digital filtering design and implementation.
The two Python examples calculated and displayed a single FFT spectrum. Thus, the author's first Python program continuously calculated, displayed and stored FFT spectra in real time. Only the level in decibels is displayed, but both the level in decibels and the phase in degrees are stored and can be saved as a CSV file which can be viewed as a text file and read into Microsoft Excel for further processing. Both the level in decibels and the phase in degrees are needed to calculate the individual cross spectra in Excel. These individual cross spectra can then be averaged over the repeated FFTs. If the front-end has only two input channels, it would be appropriate to calculate the single cross spectrum in the program, but this not appropriate if the front end has more than two input channels because there are so many possible cross spectra that can be calculated. The program can be set to calculate a given number of FFTs and save the results to disk, or to run continuously until stopped by the user but only display the level in decibels.
The program was then modified to calculate fractional octave band spectra using linear or exponential averaging. This version of the program was then extended to calculate reverberation time.

8. THE CHANGE TO C#

A problem with these two Python programs is that they could not run continuously in real time at the highest sampling rates and with the maximum number of channels even when parallel processing was used. Eventually it was discovered that Python could not read the data in over the ethernet interface fast enough even when performing no data processing. The highest speed case was 24 bits per sample at 131072 samples per second per channel and six channels. At this stage the reluctant decision was made to change to C# which experiment showed could read the data in fast enough over the ethernet interface. The reason for this difference is that Python is an interpreted language although there is some pre-compilation, while C# is a compiled language. However, C# was run in debug mode in Microsoft's Visual Studio which is like running an interpreted program. The Community version of Microsoft's Visual Studio is free for non-commercial use. C# would run even faster as a stand-alone non-debug compilation.

An upgrade to the firmware of the six-channel input front-end, that the author was using, enabled each channel to use a separate ethernet port. This enabled each channel to be read and processed in a separate parallel thread.

A problem with using C# is that, unlike Python, it does not come with modules that contain FFT routines or filtering design and implementation routines. It is necessary to use third party modules. Thus, the software uses FFTW which was developed at the Massachusetts Institute of Technology. FFTW is reportedly one of the fastest implementations of the Fast Fourier Transform and is reportedly used in the commercial MATLAB software package. However, it is written in C and thus it is necessary to use one of several C# wrappers. The author's C# programs use the FFTW.NET C# wrapper. These programs use the NWAVES software package to design and implement the digital filters and to calculate the FFT window functions.

9. SOUNDCARDS

Having written these two programs, the author then decided to convert these programs for use with soundcards. The main reason for doing this was to be able to use professional external USB soundcards as front-ends to replace aging and failing analogue acoustic measuring and analyzing equipment. The author used the Audio Streaming Input Output (ASIO) soundcard software interface that is supported by most professional audio soundcards or one of several possible Microsoft Windows soundcard software interfaces. If a soundcard does not have an ASIO interface, it may be possible to use the ASIO4ALL software interface which tries to interface to one of the Microsoft Windows soundcard software interfaces. Because the ASIO software interface, the ASIO4ALL software interface, and the Microsoft Windows soundcard interfaces are written in C, the author used the NAUDIO package to interface to them. If the sound card has more than two input channels or more than two output channels, it is necessary to use the ASIO interface to use more than two input channels or two output channels.

The author's acoustic group at CSIRO has two professional external USB soundcards. One of these has eight analogue inputs and ten analogue outputs. The other has two analogue inputs and two analogue outputs. Both these devices can supply 48 V phantom power. It is possible to purchase 48 V phantom power preamplifiers for pre-polarized measurement microphones, but it is also possible to purchase 48 V phantom power to Integrated Electronics Piezo-Electric (IEPE) adaptors. Because two of CSIRO's commercial acoustical measurement front-ends support IEPE, CSIRO decided to purchase 48V phantom power to IEPE adaptors, IEPE half inch measurement microphone preamplifiers and half inch pre-polarized measurement microphones. CSIRO also has a commercial acoustical measurement external USB front-end. Upon opening, this device was found to consist of an IEPE input board and an external USB sound card board. This device came with six integrated quarter inch microphone preamplifiers and quarter inch pre-polarized measurement microphones. This device and the combination of the 48V phantom power to IEPE adaptors and the external USB professional audio soundcards can also be used with piezo electric accelerometers which have built in IEPE preamplifiers. Hence the IEPE approach gives maximum flexibility. For measurement microphones which require an external polarizing voltage, it is necessary to use an appropriate microphone preamplifier power supply or use one of the acoustic measurement front ends with 7 pin LEMO or Brüel and Kjær microphone preamplifier sockets.

Professional audio external USB soundcards are not suitable for measuring absolute voltage levels because their peak input voltages are not calibrated, they have relatively low input impedances, and they have continuously variable uncalibrated input attenuators. But absolute voltage measurement is not required for most acoustical measurements because the whole measurement system is usually calibrated by placing a sound level calibrator on each of the measurement microphones when absolute sound pressure measurements are required. A vibration calibrator can be used with accelerometers and geophones. For many acoustic and vibration measurements only measurements of the level differences are required. This is the case for airborne sound insulation measurements if the measurements are made in both directions or if the measurement chains are swapped between rooms. It is also the case for reverberation time measurements and hence for diffuse sound absorption coefficient measurements. For two microphone impedance tube measurements, the two measurements are swapped during the calibration procedure, and again microphone calibration is not needed. Frequency response measurements also do not require absolute voltage measurements, because the measurements at two different frequencies only have to be relative, and usually both the input and output signal levels are measured.

Professional audio external USB soundcards have to have good linearity in order to have low distortion and have to have low noise to avoid annoyance. They have fairly flat frequency responses from about 20 Hz up to their ani-aliasing filters. Most current professional audio external USB soundcards are 24 bit per sample and support the following sampling frequencies: 44.1, 48, 88.2, 96, 176.4 and 192 kHz. These frequencies appear to be very accurate. The phase and level matching between different input channels and between different output channels appears to be fairly good.

One of the differences between the acoustic measurement front-ends that CSIRO has and the soundcards is that the acoustic measurement front-ends have large buffers and transfer the measured sample values in blocks that are always a positive integer power of 2. The soundcards have relatively small buffers because they are trying to obtain low latency and the number of samples that they transfer at a time is not necessary a positive integer power of 2. Thus, the author's programs for the soundcards had to implement sufficient buffering and supply the data in blocks with a positive integer power of 2 number of samples for the signal processing.

One of CSIRO's acoustic measurement front-ends has four input channels and two output channels. The firmware in this device can produce a number of different types of output signals. The author's software uses this capability to switch on and off random noise for interrupted random noise reverberation time measurements. However, CSIRO's other acoustic measurement front-end only has six input channels. Thus, it is necessary to use an external random noise generator for interrupted random noise reverberation time measurements. Hence the author implemented a USB connection to an Arduino board to turn on and off a relay controlling the random noise. He also implemented a USB connection to a General Purpose Interface Bus (GPIB) controller to switch on and off the random noise generator in a Norsonics NE830 real time analyzer. The acoustic front-end with the two output channels can also output arbitrary waveforms like a soundcard, but the author has not yet used this capability. For the sound cards, the author has implemented the output of pink random Gaussian noise, white random Gaussian noise, sine waves, square waves, sawtooth waves, swept sine waves and zero signal output (for turning off the output).

Once the FFT and fractional octave filtering programs were converted to the two different soundcard interfaces, the author wrote a stepped sine program for measuring frequency responses for the two different soundcard interfaces. He will eventually adapt this program to work with the acoustic front-ends.

10. CONSOLE AND GRAPHICAL USER INTERFACES (GUI)

The programs described in this paper were initially written as Microsoft Windows Console applications. The configuration is controlled by setting variables which are at the start of the program source code. Since the programs are fast enough that they can be run in debug mode in Microsoft Visual Studio which is a Graphical User Interface (GUI) Program, the program settings are actually changed in a GUI program, but the rest of the interaction with the program occurs in a Console window.

The author is in the process of converting these Console programs to GUI programs. There are some issues involved with doing this. Microsoft Windows was initially developed when personal computers were only single threaded. This means that the Microsoft COM protocol which is used to communicate with the Microsoft Windows graphical components is not thread safe. But threads have to be used for parallel processing. This means that it is necessary to check whether the program is calling a Microsoft Windows graphical component from the same thread that it is running in. If this is not the case, it is necessary to call the graphical component via a function which is forced to run in the same thread as the graphical component. This forcing is necessary because functions normally run in the thread that they are called from.

To use the Microsoft Windows graphing functions, it is necessary for both the Console and the GUI programs to use the .NET Framework rather than the more general .NET environment.

11. FRACTIONAL OCTAVE BANDPASS FILTERS

It is possible to design analogue third order third octave Butterworth bandpass filters which satisfy the class 1 requirements in the current international octave-band and fractional-octave-band filter standards [1-3]. This is not the case for digital third order third octave Butterworth bandpass filters. The highest frequency filter in each octave band only satisfies the class 2 requirements in part of the frequency range below its centre frequency. There are two reasons for this. The North Americans pointed out many years ago that the frequency response limits in the current international standards are not well centred about Butterworth bandpass filter frequency responses. The second reason is that analogue bandpass filters tend to be symmetrical in the logarithmic frequency domain and digital bandpass filters tend to be symmetrical in the linear frequency domain.

Digital third order elliptic bandpass filters can be designed to satisfy the current octave-band and fractional-octave-band filter standards. However, the author chose to use digital fourth order Butterworth bandpass filters because the ratio of the 3 dB down or half power bandwidth to the standard noise or effective bandwidth of an arbitrary analogue Butterworth bandpass filter of any order can be calculated [4]. This ratio can be used to calculate the lower and upper 3 dB down or half power frequencies of the analogue Butterworth bandpass filter which has the noise or effective bandwidth required by the current standards. These two frequencies divided by the Nyquist frequency are needed as inputs to the software used to design the digital Butterworth bandpass filter. The author understands that this software usually works by designing an analogue Butterworth lowpass filter, converting this filter to an analogue Butterworth bandpass filter, and then converting this filter to a digital Butterworth bandpass filter.

Some caution is needed here, because the equation for the ratio of the 3 dB down or half power bandwidth to the standard noise or effective bandwidth of an arbitrary analogue Butterworth bandpass filter of any order is strictly speaking only correct for the analogue Butterworth bandpass filter which is converted to the digital Butterworth bandpass filter. Secondly, the standard definition of the noise or effective bandwidth used to derive this ratio assumes that the exciting signal is white Gaussian random noise. Somewhat surprisingly, the current international standards use a nonstandard definition of the normalized noise or effective bandwidth which assumes that the exciting signal is pink Gaussian random noise. This appears to be done so that measurements of the normalized noise or effective bandwidth can be made using a constant amplitude sine wave whose frequency varies exponentially with time and because a pink Gaussian random noise signal has a flat fractional octave spectrum. Numerical integrations to calculate the noise or effective bandwidth following the requirements of the current international standards of a fourth order Butterworth fractional octave bandpass filter designed following the method described above showed that the noise or effective bandwidth was less than the required by 0.039 dB for a 31.5 kHz octave filter and by 0.010 dB for a 40 kHz third octave filter. These two decibel differences are less than one quarter of the magnitudes of the allowed decibel differences between the noise or effective bandwidth and the required value of ±0.4 dB for class 1 filters and ±0.6 dB for class 2 filters. In the 1970s, CSIRO purchased a General Radio third octave band real time analyzer. This consisted of a bank of parallel analogue third octave filters and a scanning digital voltmeter which sampled at an average sampling rate of about 50,000 samples per second. Because it scanned the outputs of about 50 filters, each filter was scanned at an average sampling rate of only about 1000 samples per second. Thus, many of the outputs of the higher frequency analogue filters were scanned below the Nyquist frequencies required for sampling their output signals. To try and avoid coherence between the output signals of the analogue filters and the sampling rate, the sampling rate was varied by a factor of about two over the selected linear integrating time.

Although this approach worked well, in 1978 when the author visited the Danish Technical University, he discovered that they were unhappy with this solution. They used the General Radio bank of parallel analogue third octave filters, but had developed their own scanning digital voltmeter which scanned the third octave filters in each lower octave band at half the sampling rate used in the octave band immediately above. With this approach, they were able to sample all the filter outputs with a Nyquist frequency above the frequency range of each filter, but still have the same total sampling rate.

The first Brüel and Kjær third octave band real time analyzer used a bank of parallel analogue third octave filters and a bank of parallel analogue root mean square voltmeters and exponential averaging devices whose output voltages were digitally sampled at a low sampling rate and displayed on a video screen. There was one analogue root mean square voltmeter and one exponential averaging device for each analogue third octave filter.

When Brüel and Kjær developed their first digital third octave band real time analyser, they adopted a version of the idea used by the Danish Technical University. In half of the signal processing time, all the outputs of the digital third octave bandpass filters in the top octave were calculated and the signal was low pass filtered and every second sample was passed for processing to the digital filters in the octave band immediately below the current octave band and further low pass filtering. This halved the Nyquist frequency and meant that the filtering for the filters in the next highest octave took only half the time of the top octave band, namely one quarter of the total signal processing time. The next highest octave band then took only one eight of the total signal processing time, and so on. This meant that all the filters in all the octaves bands could be processed in real time in the total amount of the signal processing time that was available. The software developed by the author also uses this process to obtain the fastest signal processing speed that is possible. Another reason for using this process is that numerical instability can occur if the bandwidth of a digital bandpass filter is very small compared to the Nyquist frequency. This can occur if digital bandpass filters in a large number of octave bands are directly calculated using the unreduced original sampling rate because the bandwidth of a constant percentage band-pass filter is proportional to its centre frequency. The low pass filter is a twelfth order Butterworth low-pass filter which is 0.1 dB down at the top edge frequency of the next highest octave band. This top edge frequency is of course equal to the bottom edge frequency of the current octave band for which signal processing is being conducted.

The signal processing software developed by the author can conduct linear or exponential averaging. For reverberation time measurements it is desirable to be able to vary the averaging time as a function of the centre frequencies of the fractional octave filters. For exponential averaging it is possible to set a different averaging time for each fractional octave filter. However, because the number of fractional octave filters becomes large when the frequency range in octaves becomes large and the number of filters per octave is also large, the software currently uses a constant exponential averaging time for all filters in a particular octave band, but allows this constant exponential averaging time to be different for different octave bands. The initial linear averaging time is the same for all frequencies, but longer averaging times can be obtained by calculating the running average of the current and the last so many linear averages. Again, the number of averages combined to form a running average is currently the same for all fractional octave filters in a particular octave band but can be different for different octave bands. For statistical purposes, it is sometimes desirable to use statistically independent levels when calculating reverberation times. Thus, the software enables the use of only every *N*th running average when *N* linear averages have been averaged to form the running average.

12. FAST FOURIER TRANSFORMS (FFT)

Most Fast Fourier Transform (FFT) software packages only return the complex amplitudes of the non-negative frequency spectrum lines because the complex amplitude of a negative frequency spectrum line of an FFT of real number values is equal to the complex conjugate of the complex amplitude of the corresponding positive frequency spectrum line. This means that it is necessary to multiply the complex amplitudes by 2. But it is then necessary to divide the complex amplitudes by $\sqrt{2}$ to convert them to root mean square values. This means that the nett effect is multiplication by $\sqrt{2}$. These corrections are not needed for the zero-frequency spectrum line. It is then necessary to correct for the effects of the windowing function $w(n)$ that has been used.by multiplying the complex amplitudes by the amplitude correction factor A_w,

$$A_w = N\Big/\sum_{n=0}^{N-1} w(n) \;.....................\; (1)$$

where N is the number of samples whose FFT is being calculated. Effectively, the complex amplitudes are divided by the average value of the windowing function. However, conducting a reverse FFT of the spectrum of an FFT should produce the original samples. This is not the case unless both the FFT and the reverse FFT are divided by divisors whose product is equal to *N*, the number of original samples. Unfortunately, most FFT software packages divide the FFT results by 1 and the reverse FFT results by *N*. Exactly the opposite is needed for signal processing. This means that the complex amplitudes have to be multiplied by

$$\sqrt{2}A_w/N = \sqrt{2}\Big/\sum_{n=0}^{N-1} w(n) \;................\; (2)$$

The use of a windowing function increases the noise or effective bandwidth B_e of a spectral line beyond the line spacing $\Delta f = 1/(N\Delta t)$ of the centre frequencies of the spectral lines, where Δt is the time interval between samples. If random noise is being measured, the results can be made independent of the spectral line spacing and the window being used by dividing the moduli squared of the complex rms amplitudes of the spectral lines by the noise or effective bandwidth B_e to obtain the spectral densities.

$$B_e = N\Delta f \sum_{n=0}^{N-1} w^2(n) \Big/ \left[\sum_{n=0}^{N-1} w(n)\right]^2 \;..........\; (3)$$

The software presents the mean square values or spectral densities as decibels in real time in the form of graphs. The values displayed in each set of graphs can be averages of a number of measurements if necessary, in order to reduce the computing load of redrawing graphs very frequently. The software can save the results to disk in the form of comma separated value (CSV) files which can be read into

Microsoft Excel for further processing. The amplitudes are saved as the moduli squared of the complex rms amplitudes expressed in dB relative to a reference rms amplitude which can be specified. For FFTs, the phase angles in degrees are also saved.

13. REVERBERATION TIME MEASUREMENTS

The reverberation time measuring part of the software calculates both reverberation times and decay rates for each individual decay, for the ensemble averaged decay curve of each measurement configuration, and for the overall ensemble averaged decay curve. It also calculates an extensive array of statistics.

Modern computer operating systems allocate slices of time to the many processes that are running nominally at the same time and possibly actually at the same time on multiple threads and on multiple processing units. The output and input streams to the digital to analogue and from the analogue to digital converters are buffered to try and avoid lost data, there are delays due to the conversion transmission processes, and there is a time of flight delay between the loudspeaker and the microphone. All this means, that it is impossible to know at exactly which input samples the sound is turned on and turned off.

The author’s software uses the sudden increase in the level in the highest frequency band in which the reverberation time is being measured to determine the sample number when the sound first effects the microphone signal. When the output sound sequence is generated by the software, it knows how many sample intervals the sound is turned on for, and thus it knows when to start looking for the beginning of the sound decay. The decay curves are aligned in time so that the starting times of their decays are the same before they are ensemble averaged together. When averaging the ensemble average decay curves from different microphone and loudspeaker combinations, the ensemble averaged decay curves are also aligned in time so that the starting times of their decays are the same and normalized in level so that their steady state levels are the same.

When determining the evaluation range, what does “5 dB below the initial sound pressure level” mean for a decay record which consists of individual points rather than a continuous curve?

There are at least three possibilities for determining the starting point.

1. The first point in time which is more than “5 dB below the initial sound pressure level”.
2. The point in time immediately prior to the first point in time which is more than “5 dB below the initial sound pressure level”.
3. The point, of the two points chosen using possibilities 1 and 2, which is closest in level to “5 dB below the initial sound pressure level”.

The same possibilities exist when determining the finishing point, except that “the finishing level” replaces “5 dB below the initial sound pressure level”. But how is the finishing level determined? There are at least two possibilities.

1. The finishing level is the level which is (the evaluation range plus 5 dB) “under the initial sound pressure level”.
2. The finishing level is the level which is the evaluation range under the starting point.

The author’s software supports the use of all these possibilities and a variable evaluation range.

14. CONCLUSION

This paper presents examples demonstrating the need to minimize the use of “black box” software and hardware by developing a laboratory’s own software. It then describes the development of such software and some of the issues that had to be addressed. The author plans to eventually write sound intensity measurement software and two microphone impedance tube measurement software. He may also write sound level meter software.

15.REFERENCES

[1] IEC 61260-1:2014 Electroacoustics – Octave-band and fractional-octave-band filters – Part 1: Specifications Edition 1.0 2014-02, in, International Electrotechnical Commission, Geneva, Switzerland, 2014, pp. 93.

[2] IEC 61260-2:2016 Electroacoustics – Octave-band and fractional-octave-band filters – Part 2: Pattern-evaluation tests Edition 1.0 2016-03, in, International Electrotechnical Commission, Geneva, Switzerland, 2016, pp. 55.

[3] IEC 61260-3:2016 Electroacoustics – Octave-band and fractional-octave-band filters – Part 3: Periodic tests, in, International Electrotechnical Commission, Geneva, Switzerland, 2016, pp. 51.

[4] J.L. Davy, I.P. Dunn, The statistical bandwidth of Butterworth filters, Journal of Sound and Vibration, 115 (1987) 539-549.