audioXPRESS
Tube, Solid State, Loudspeaker Technology

Article prepared for www.audioXpress.com

MATAA: A Free Computer-Based Audio Analysis System

By Matthias S. Brennwald

Discover this useful software tool, which acts as a versatile audio analyzer.

I needed an audio analyzer to test my loudspeakers and amplifiers. However, the systems available on the market are either not flexible enough for my needs, don't work with my preferred computer platform, or cost more than what I am willing to spend. I therefore started a new DIY project "building" my own audio analysis system using my computer's soundcard. After a while, the project grew bigger and I realized that I produced a very versatile and powerful audio analyzer, which I call "Mat's Audio Analyzer" (MATAA).

The operating mode of MATAA is the same as with many other computer-based audio analysis systems. Apart from that, however, MATAA is a little different from most of these systems in many ways. For instance:

1. MATAA does not rely on proprietary hardware (such as an expansion card or a "switch box") to handle the sound input/output and the electrical connections to the device under test (DUT). In contrast, you can use whatever soundcard, cables, switches, plugs, amplifiers, and so on that you think are appropriate for your needs.
2. MATAA does not rely on a specific computer platform, because MATAA runs from within MATLAB or GNU Octave. These "number-crunching" programs run on virtually all current platforms (keep reading if you are not familiar with MATLAB or Octave).
3. MATAA is free software released under the GNU General Public License[1] (GPL). You are free to study the source code, adapt it to your needs, and release your improved version under the GPL. You can download MATAA from **www.audioroot.net/mataa.html**.

MATAA is essentially a collection of MATLAB/Octave programs, which provide the building blocks for analysis procedures to test all kinds of audio devices. Furthermore, MATAA contains ready-made scripts using these building blocks to conduct typical analyses and tests (e.g., to measure the impulse response and the frequency response of a loudspeaker). Alternatively, you can design specific test routines to efficiently analyze virtually anything that can be analyzed with audio test signals.

HOW MATAA WORKS

Similar to most other computer-based audio analysis systems, MATAA feeds a test signal to the DUT and simultaneously records its response signal, which is then analyzed as desired (see **Fig. 1**, which I will explain in more detail). One notable strength of MATAA is that you can use MATLAB/Octave to generate *any* kind of test signal and then feed it to the DUT. Together with the wide range of MATLAB/Octave tools for data processing and analysis, this allows carrying out virtually *any* analysis you can think of.

MATAA does not have a whiz-bang graphical user interface. While this may seem a bit anachronistic for today's computer software, the lack of a graphical user interface is one of the reasons MATAA is so flexible. Have you ever experienced a program that couldn't do what you wanted because the corresponding button was missing, although the functionality would have been built into the software? Not with MATAA.

MATAA SOUNDCARD REQUIREMENTS

In principle, any soundcard that allows simultaneous sound input and output ("full duplex sound") is suitable for use with MATAA. However, the quality of the soundcard will, of course, crucially determine the quality of the measured data. For instance, the maximum sampling rate determines upper frequency limit; the signal/noise ratio and the sampling bit depth determine the dynamic range; and, depending on the intended type of analysis, the number of input and output channels available may also be important (stereo soundcards are suitable

for the vast majority of possible applications).

MATLAB AND OCTAVE

MATLAB and Octave are "number-crunching" programs that provide an all-in-one environment for analysis, processing, and graphing of data. Both MATLAB and Octave run on various computer platforms and operating systems. They provide a powerful programming language, which, compared to many others, is easy to learn and understand. Programs written for MATLAB are (largely) compatible with Octave (and vice versa). While MATLAB and Octave are very similar, they differ in one important point:

MATLAB is a commercial product by The MathWorks company[2], whereas Octave is free software released by the GNU project[3].

One shortcoming of both MATLAB and Octave with respect to MATAA is their limited capabilities for sound input and output. While both MATLAB and Octave provide commands to play and record sound, these commands do not work on all operating systems. Furthermore, simultaneous sound input and output, a prerequisite for MATAA, is not well implemented.

Therefore, I wrote a short program called TestTone that handles the sound input and output for MATAA externally from MATLAB/Octave (**Fig. 1**). MATAA talks to TestTone from within MATLAB/Octave, so the MATAA user does not need to care about TestTone, which is based on PortAudio[4], a free cross-platform library for sound input/output. It is therefore straightforward to enable sound input and output for MATAA on all computer platforms supported by PortAudio.

At the time of this writing, I have compiled TestTone for MacOS X. Also, Sang Shu (sangshu@hotmail.com) has compiled TestTone for Windows (thanks, Shu!). MATAA therefore supports sound input and output on these platforms, but other platforms (e.g., Linux) are not yet supported. If someone agrees to compile TestTone for Linux or any other computer platform and provide it to the public through the MATAA website (with your nametag on it), you are more than welcome to do so!

USING MATAA

In this section, I will show how to operate MATAA and how MATAA works in real-world applications. I ran these examples using Octave, but the operation and results would have been exactly the same with MATLAB. I used an Apple PowerBook G4 laptop computer with its generic built-in soundcard, which supports various sampling rates in the range of 32–96kHz with a sample depth of up to 24 bits. By feeding sine signals of varying frequencies from an analog signal generator to the soundcard input and analyzing the digitized result, I found that this soundcard has an efficient anti-aliasing filter, whose cutoff frequency is automatically adjusted to the sampling rate selected.

RC HIGH-PASS FILTER

In this first example, I use MATAA to test a simple RC high-pass filter with R = 4.79kΩ and C = 220nF. While not rocket science, this example serves as a good introduction to how MATAA works. **Figure 2** shows the RC filter and how it is connected to the soundcard.

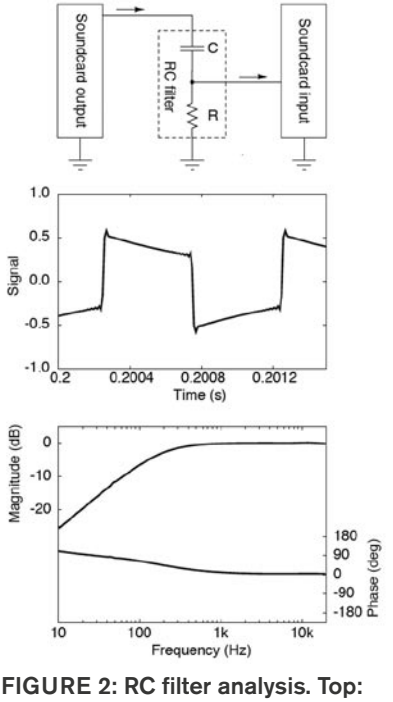

FIGURE 2: RC filter analysis. Top: setup of the RC filter and sound hardware. Center: squarewave response of the RC filter. Bottom: frequency response of the RC filter (magnitude and phase corrected for excess phase).

To start, I demonstrate how to feed a square-wave signal to a simple RC filter and measure its output signal. Using the MATAA command **mataa_signal_generator**, I produced a 0.1s long square-wave signal with a frequency of 1kHz and a sampling rate of 96kHz by typing the following command at the MATLAB/Octave prompt:

```
s = mataa_signal_generator('square',9600
0,0.1,1000);
```

The samples of the square-wave signal are now stored in the MATLAB/Octave variable **s**, but the signal has not yet been fed to the soundcard or the RC filter.

The mataa_measure_signal_response command feeds the square-wave signal to the input of the RC filter and simultaneously records the response signal at the output of the filter:

```
res = mataa_measure_signal_response
(s,96000);
```

The samples of the response signal are now stored in the variable **res**. **Figure 2** shows one cycle of the response signal (this figure was produced using the **mataa_plot_signal** command, but you could just as well use the standard MATLAB/Octave **plot** command).

Apart from the slight high-frequency ringing, the shape of the signal looks exactly as expected for the RC high-pass filter. The high-frequency ringing is due to the steep anti-aliasing filter of my soundcard (I will discuss how to deal with signal distortions due to the anti-aliasing filter or other artifacts introduced by the sound hardware). In summary, typing only three commands was enough to produce a test signal, feed it to the RC filter, record the response signal, and plot the result.

In the next step, I demonstrate how to measure the impulse response of the filter and how to determine the transfer function of the filter in the frequency domain. By typing the following three commands, MATAA measures the impulse response using a white noise signal and calculates the frequency response (magnitude and phase) of the RC filter:

```
w = mataa_signal_generator('white',
96000,0.3);
h = mataa_measure_IR(w,96000);
[mag,phase,f] = mataa_IR_to_FR(h,96000);
```

On the first line, a white-noise signal **w** with a sampling rate of 96kHz and a

length of 0.3s is generated. On the second line, this white-noise signal is used to measure the impulse response of the RC filter using the MATAA command mataa_measure_IR. This command first feeds the signal in **w** to the RC filter and records the response of the RC filter (analogous to the prior square-wave test). The response signal is then deconvolved from the original signal (**w**), which results in the impulse response **h** of the RC filter.

On the third line, mataa_IR_to_FR is used to transform the time-domain impulse response to the frequency response using the Fourier transform (magnitude mag and phase phase as function of frequency f). The resulting frequency response (**Fig. 2**) corresponds to the expected first-order high-pass transfer function of the RC filter.

Note that the computer needs some time to process and feed the test signal data to the sound output. Also, while not the case in this example, the signal may be further delayed by the DUT itself or due to long signal travel times in the measurement setup (e.g., the time needed for a signal to travel from a loudspeaker to a microphone). If the signal recording stops immediately after the last sample of the test signal has been output from the audio output, the recoding of the DUT's output signal will therefore be truncated. By default, MATAA therefore extends the recording of the test signal by 0.1s before and after the test-signal output to avoid cutting off the recording (different delay values may be specified).

In the frequency domain, this delay in the DUT output signal corresponds to a phase shift that increases linearly with frequency. In the RC-filter example, the true phase shift is virtually zero at frequencies much higher than the cutoff frequency. You can therefore compute the excess phase due to the signal delay from the measured phase and the expected (zero) phase.

By specifying the frequency range where phase is expected to be zero (e.g., 5–20kHz), the following command removes the excess phase, leaving only the phase response of the RC filter (the so-called minimum phase, **Fig. 2**):

```
phase = mataa_phase_remove_trend
(phase,f,5000,20000);
```

As an alternative, MATAA also allows using the Hilbert transform to determine the minimum phase from the magnitude of the frequency response.

LOUDSPEAKER IMPEDANCE

Figure 3 shows the measurement setup to measure the electrical impedance of a loudspeaker[5] (or, in fact, anything else). The sound output is connected to a series combination of the loudspeaker and a resistor, which acts as an impedance reference. The value of the reference resistor should be of the same order as the impedance of the loudspeaker.

In contrast to the setup in **Fig. 2**, both stereo channels of the sound input are used. One channel records the signal voltage across the loudspeaker (DUT channel), while the other records the signal voltage at the sound output, which will be used as a reference (REF channel). By default, MATAA allocates the left channel to the DUT and the right channel to REF.

The purpose of using the second channel (REF) is to make the analysis immune to distortions of the original test signal by the sound output electronics. Instead of comparing the DUT signal to the original test signal generated by MATAA (as in the RC-filter example), the DUT signal is compared to the REF signal, i.e., to the true signal applied to the DUT.

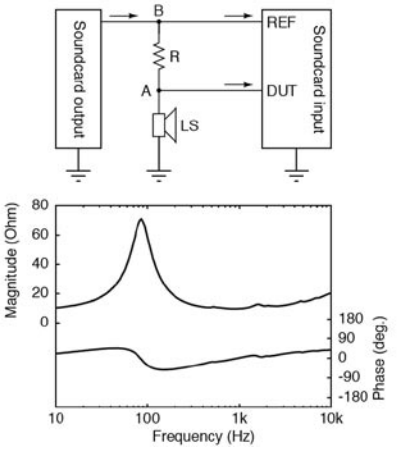

FIGURE 3: Loudspeaker impedance analysis. Top: measurement setup. Bottom: result of impedance measurement for a Fostex FE108Σ driver in free air.

Many soundcards are not designed to deliver undistorted output when directly driving loudspeakers or similar low-impedance loads. The REF signal may therefore be distorted with respect to the original test signal generated by MATAA. Furthermore, as previously illustrated, the anti-aliasing filter of the sound input may also distort the test signal. You can sidestep this distortion problem by comparing the DUT signal to the REF signal rather than the original MATAA signal, because the soundcard distortion affects both the DUT and REF signals in the same way.

The reference resistor R and the loudspeaker with its impedance Z constitute a voltage divider between the soundcard output and ground (**Fig. 3**). U_A is the voltage between point A and ground (i.e., the voltage across the loudspeaker). U_B is the voltage between point B and ground (i.e., the voltage across the loudspeaker and the reference resistor). The ratio of these two voltages is $U_B/U_A = (R+Z)/Z$. Solving this equation for the loudspeaker impedance gives:

$$Z = R\,\frac{U_A}{U_{B-U_A}}$$

For a sinusoidal test signal, U_A and U_B reflect the amplitudes of the sine signals at points A and B. Note that the sine signals may show a phase shift due to the complex nature of the loudspeaker impedance. Mathematically, this is expressed with complex values U_A and U_B. For non-sinusoidal signals, you can compute U_A and U_B as a function of frequency using the Fourier transform of the signals measured at points A and B. Inserting these frequency-dependent values in the previous equation then yields the loudspeaker impedance Z as a function of frequency[5].

This procedure to determine the frequency-dependent impedance of a loudspeaker (or any other DUT) is implemented in the MATAA tool mataa_measure_impedance. With a reference resistor R = 8.6Ω (that's what came out of my parts box), the following command measures the loudspeaker impedance (magnitude mag and phase phase) in the frequency range from 10–10000Hz:

```
[mag,phase,f] = mataa_measure_imped-
ance(10,10000,8.6);
```

By default, this command uses a sine-

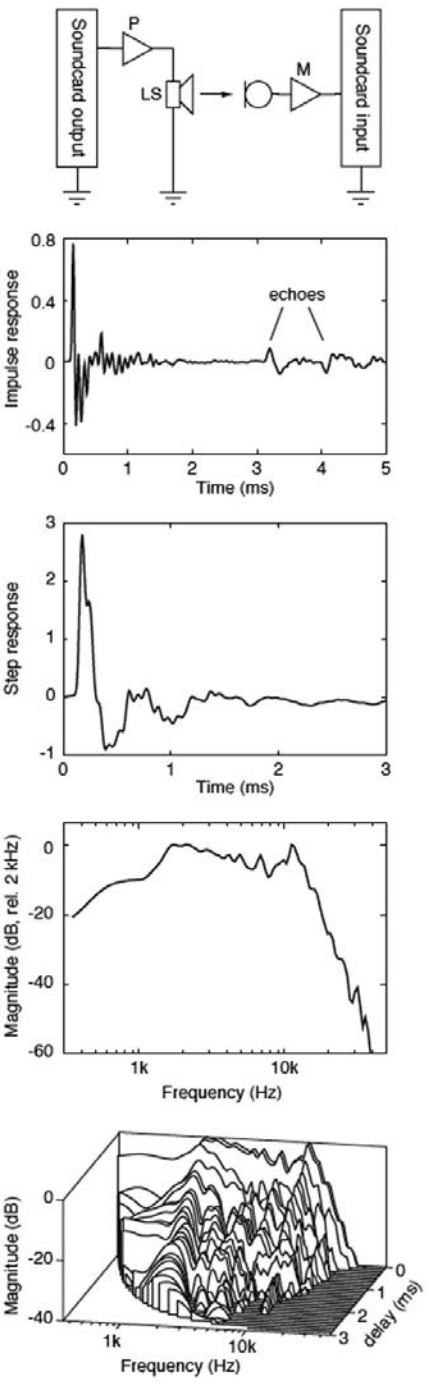

FIGURE 4: Acoustic loudspeaker analysis. From top to bottom: measurement setup (P: power amplifier, M: microphone amplifier, LS: Fostex FE108Σ loudspeaker in free air), impulse response (with room echoes), step response (with echoes truncated, note the different time scale), anechoic frequency response, and cumulative spectral decay diagram ("waterfall plot").

sweep test signal of appropriate length, selects a suitable sampling rate, and smoothes the result in 1/48-octave bands (other values may be specified). The smoothing attenuates the noise that will be picked up by the loudspeaker, because it also acts as a microphone. **Figure 3**, generated with the mataa_plot_impedance command, shows the result for a Fostex FE108Σ full-range driver operated under free-air conditions.

ACOUSTIC LOUDSPEAKER ANALYSES

In this section, I demonstrate how to measure the impulse response of a loudspeaker, and how to calculate the step response, the anechoic frequency response, and a cumulative spectral decay diagram ("waterfall plot") of the loudspeaker. I recommend the book by Joe D'Appolito[5] for a thorough and well-written introduction to the methods and concepts involved in these analyses (and everything else you ever wanted to know about loudspeaker testing).

Figure 4 shows the setup to measure the impulse response of a loudspeaker (the same Fostex FE108Σ as before). Both the microphone (a Behringer ECM8000) and the loudspeaker were mounted 90cm above the floor, with a distance of 100cm in between each other. Note that the loudspeaker is driven through a power amplifier to buffer the soundcard output signal. In contrast to the impedance analysis, calibration of the analysis using the REF channel is therefore not mandatory. Hence, for the sake of simplicity, I decided not to use the REF channel, although MATAA does allow calibrating the impulse-response measurement using the REF channel.

Acoustic analyses can be prone to environmental noise. I therefore used a maximum length sequence (MLS) test signal to achieve a good signal/noise ratio for the impulse-response measurement. Other signals such as pink or white noise are suitable, too.

The following commands first produce an MLS test signal (**s**, with a length of 2^{14} - 1 = 16383 samples), which is then used to measure the impulse response **h** of the loudspeaker using a sampling rate of 96kHz (analogous to the RC-filter example):

```
s = mataa_signal_generator('MLS',96000
,0,14);
h = mataa_measure_IR(s,96000);
```

Note that the impulse response in **h** will show a delay due to the travel time of the test signal from the loudspeaker to the microphone (about 2.92ms for a distance of 100cm between the driver and the microphone).

The following two commands remove this delay and shorten the impulse response to a length of 5ms:

```
t0 = mataa_guess_IR_start(h,96000);
h = mataa_signal_crop(h,96000,t0,t0+
0.005);
```

On the first line, the start time (**t0**) of the impulse response starts is determined automatically. On the second line, the impulse response is cropped to the range between **t0** and **t0**+5 ms.

Figure 4 shows the resulting impulse response, which is followed by echoes from the floor and the walls of my room. The first echo occurs at about 3.1ms, which corresponds to the time delay of the echo from the floor with respect to the direct sound from the loudspeaker. You must remove the echoes to determine the anechoic frequency response.

The simplest method to do this is to shorten the impulse response to the time range that is free of room echoes (i.e., 0–3ms):

```
h = mataa_signal_crop(h,96000,0,0.003);
```

Alternatively, MATAA also provides a tool to multiply the impulse response by various types of window functions in order to attenuate the echoes while retaining the anechoic part of the signal. For the sake of simplicity, however, I won't discuss signal windowing here.

The impulse response is now free of room echoes, but still needs to be corrected for the frequency response of the microphone. This is done as follows:

```
h = mataa_microphone_correct_
IR('Behringer_ECM8000',h,96000);
```

This command reads the frequency response of the microphone from the specified ASCII file and uses this data to correct the impulse response.

After this correction, the echo-free impulse-response can now serve as the basis for various analyses, such as those shown in **Fig. 4**:

- Step response: `hs = mataa_IR_to_SR(h,96000);`
- Frequency response (anechoic) smoothed to 1/24 octave bands (mag-

nitude and phase): `[mag,phase,f] = mataa_IR_to_FR(h,96000,1/24);`

- Cumulative spectral decay diagram (waterfall plot) with 30 lines covering the spectral decay of the impulse response during the full anechoic range (3 ms): `T = linspace(0,0.003,30); [mag,f,T] = mataa_IR_to_CSD(h,96000,T,1/24);`

AMPLIFIER DISTORTION

Figure 5 shows the measurement setup to analyze the harmonic distortion of an amplifier. The amplifier output is connected to a load resistor R_L = 7.6Ω and a potentiometer R_G = 4.7kΩ to attenuate the output signal of the amplifier (otherwise, the sound input would be overloaded due to the gain of the amplifier). Again, I chose these values because that's what came out of my box.

MATAA provides the `mataa_measure_HD(f1,T,fs,N)` program to measure the harmonic distortion products and total harmonic distortion (THD) at a given base frequency. This command feeds a sine signal to the DUT (**f1** is the frequency of the sine, **T** is its length, **fs** is the sampling rate, **N** is the number of harmonics to be included in the analysis). The program then determines the Fourier spectra of the resulting DUT and REF signals at the soundcard input. These spectra are then normalized such that the amplitude of the base frequency (**f1**) is k_1 = 100%.

To correct for possible distortion artifacts introduced by the soundcard, the REF spectrum is subtracted from the DUT spectrum, yielding in the "true" distortion spectrum of the DUT. Finally, the program returns the amplitudes $k_2, k_3, \ldots, k_N$ of the harmonics and the THD, which is defined here as the geometric sum of the harmonics:

THD =

$$\sqrt{k_2^2 + k_3^2 + \ldots + k_N^2}$$

(note that other definitions exist).

In the current example, I demonstrate how to measure THD and the first four harmonics ($k_2, k_3, \ldots, k_5$) as a function of frequency. To this end, the distortion analysis is repeated at 32 different frequencies in the range of 30–9600Hz. I accomplished this by putting the following commands into a MATLAB/Octave script file:

```
Nf = 32; Nk = 5; fs=96000;
f = logspace(log10(30),log10(9600),Nf); thd
= repmat(NaN,1,Nf);
k = repmat(NaN,Nk,Nf); for i=1:Nf
[thd(i),k(:,i)] = mataa_measure_
HD(f(i),0.1,96000,Nk); end
```

On the first line, the number of frequencies at which distortion is to be analyzed (**Nf**) is set to 32, the number of harmonics to be analyzed (**Nk**) is set to 5, and the sampling rate (**fs**) is set to 96kHz. On the second line, a list (**f**) of the frequency values within the range of 30–9600Hz is produced (the frequeny values are distributed logarithmically). On the third line, the variables to hold the THD data (**thd**) and the distortion harmonics (**k**) are initialized with empty values **(NaN).** The last three lines constitute a loop in which the THD and distortion harmonics are analyzed repeatedly at the increasing frequencies in **f**.

I used this MATLAB/Octave script to analyze the distortion of my Quad II tube amplifier. I set the output power to 13.9W (at higher power, clipping becomes visible on the scope). The result of the distortion analysis is shown in **Fig. 5**. Note that the resulting THD is nicely in line with the specified[6] THD value of ~0.2% at 700Hz and 12W output.

CLOSING REMARKS

The versatility of MATAA and the lack of a graphical user interface may (wrongly) suggest that operation is cumbersome and less streamlined than with other computer-based audio analyzers. In contrast, these peculiarities are one of the reasons I prefer MATAA over other audio analysis systems. On the one hand, you can have MATAA conduct virtually any analysis you can think of, while on the other hand, you can save the necessary steps for a specific analysis to a MATLAB/Octave script file so that you can repeat the analysis by simply running this script. In addition, MATAA already includes several scripts that automatically conduct some typical analyses.

MATAA therefore offers "plug-and-play" for these applications and allows new users to become familiar with MATAA. Finally, from my own experience, I have the impression that MATAA forces the user to think first about how to conduct the analysis best. This ultimately leads to a better understanding of the analysis and therefore improves the quality of the results and their interpretation.

I would like to stress that you shouldn't abuse your soundcard. Shortening the soundcard output, for example, usually does not destroy anything, but too much voltage at the soundcard input will. While a few volts won't do any harm, I have learned the hard way that a capacitor with a maximum voltage of 100V is not suitable to protect the soundcard from a 400V B+ voltage in a tube amplifier (I tried to analyze the ripple voltage of the B+). The best way to protect your soundcard is to think twice before applying a signal to it.

Also, inserting a voltage limiter just before the soundcard input may be a good idea. For instance, grounding the soundcard input through two series-connected 5V zener diodes with reversed polarity will limit the signal to ±5V (but the zeners will also blow up

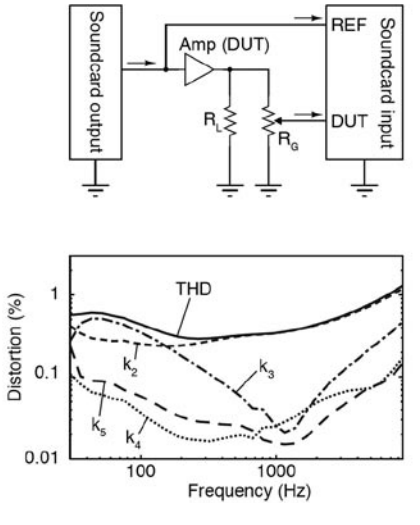

FIGURE 5: Distortion analysis. Top: measurement setup (RL: load resistor, RG: potentiometer to attenuate and adjust the signal level for the sound input). Bottom: result of distortion measurement for my Quad II tube amplifier.

if too much voltage is applied for too long). Finally, if you are paranoid, you can minimize the risk of destroying the entire motherboard of your computer by using an external soundcard connected to your computer via the USB or FireWire port, for instance.

MATAA works fine for me, but I am sure other users will find a few rough spots that I am not yet aware of. Also, there are applications that are not (yet) covered by MATAA. Instead of trying to think of such applications myself and writing suitable programs, I decided to let other users request, suggest, or even write new code to improve and expand on the functionality of MATAA. I hope this results in a broad and ever-growing range of applications that are supported by MATAA "out of the box." So get in touch with me and other MATAA users through the MATAA homepage www.audioroot.net/mataa.html if you need (or want) MATAA to do something you don't know how to accomplish, or if you've written code that expands the functionality of MATAA. *aX*

REFERENCES

1. Free Software Foundation (FSF). GNU General Public License. www.gnu.org/licenses/gpl.html.
2. The Mathworks, Inc. www.mathworks.com.
3. Free Software Foundation (FSF). www.gnu.org.
4. PortAudio. PortAudio—an open-source cross-platform audio API. www.portaudio.com.
5. Joseph D'Appolito, *Testing Loudspeakers*, Old Colony Sound Lab, 888-924-9465, custserv@audioXpress.com.
6. *Instruction Book—QUAD 22 Control Unit and QUAD II Power Amplifier.* Quad, 1964.