

An Open-Source Electroacoustic Measurement System

Part 2: Sound Card Setup, System Characterization and a few more Examples

Richard Mann and John Vanderkooy

Overview

Our objective is to use “consumer grade” equipment, in particular USB sound cards, for electrical and acoustic measurements.

The goal of this work is to provide a practical test instrument based on Open Source software (GNU Octave) [1]. In addition to providing an immediately accessible platform, we hope that this spurs reader experimentation and investigation. By using the simple programs provided, the reader will learn the inner workings of acoustic measurement and computation. We were pleased to discover that a few lines of code were sufficient to perform many acoustic measurements, simulations and experiments.

GNU Octave is an open source alternative to Matlab [2]. Octave/Matlab is a powerful interpreted language, with scientific computation, simple graphics and many useful graphical user interface (GUI) features. In addition, many of the standard linear systems and signal processing features are built in, including Discrete Fourier Transform, Convolution, FIR and IIR filter design, resampling (with built in anti-alias filters) and much more. Our programs will work for both Octave and Matlab, and have been tested under both Linux and Windows.

Here, we present an interactive program, “SoundCardSetup.m”. The idea is to output a relatively short length (one second or less) stimulus sequence over and over, and each time, let the spectrum and the transfer function update on the display in real time.

In addition to setting up the sound card for measurements (Part 1), we have found this interactive program very useful for experimentation and debugging. For example, for speaker measurement, the microphone (or speaker) may be moved and effects are immediately visible as the screen updates. Similarly we may test electrical circuits, such as amplifiers, compressors, equalizers, etc.

Our program is motivated by the MLSSA software originally developed by Rife and Vanderkooy [3], which repeatedly computes the transfer function using an MLS sequence. We provide code for MLS, a pink version of MLS, whose power spectrum falls off as $1/f$, and for pure tones. Users may also create their own stimulus patterns.

Our program employs some basic features of the Matlab/Octave graphical user interface (GUI). The code is simple and highly configurable. In fact, several extra displays are included in the code (uncomment them in the code to activate them). The displays are customizable with a few changes, so custom displays can be added easily. We encourage the reader to run the program and experiment. You won't break the computer or the soundcard...

Qualifying the Audio Chain

Before we can perform acoustic measurements, we must first verify our measurement system. In particular, we must ensure proper input and output connections, that outboard amplifiers and drivers are working, that playback and record levels are set correctly, and that the soundcard is functioning properly.

As described in Part 1, the measurement system consists of a computer (PC/laptop), a sound card and some system to be measured, which we call the “device under test” (DUT) connected as in Figure 1. Following the convention in Part 1, the left channel output, *sig*, is “looped back” to the input, and this recorded “reference channel” is called *ref*. The DUT is fed from the reference signal, and its output is recorded by the second channel, called *dut*. The measurement works by comparing the signal *dut* either *ref* or *sig*.

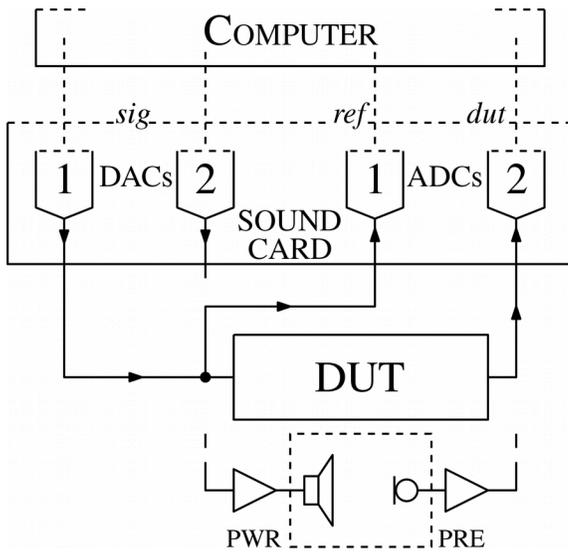


Figure 1. Architecture of measurement system (from Part 1). The sound card feeds the “device under test” (DUT), which may consist of an electro acoustic channel (power amplifier + speaker + microphone + preamp) as shown, or a purely electronic channel (amplifier, effects pedal, etc). The device is measured by comparing the response “direct path” (ADC 1, left channel) to the “indirect path” (ADC 2, right channel). Note: In Part 2 we always wire DAC 1 (left channel output) to ADC 1 (left channel input). The other path is DAC 2 (right channel output) input to DUT output to ADC 2 (right channel input).

Since we know the stimulus signal, we can measure the sound card chain (DAC → output amplifier → input amplifier → ADC) by comparing the loopback signal *sig* to the reference *ref*. To test the sound card a pre programmed stimulus signal (tone, pulse, etc) is used.

We emphasize that even though the sound card inputs may sound clear and undistorted, there can often be subtle problems in the signal chain, such as: noise/interference on input or outputs, ADC and DAC overloads, limiting and/or automatic gain control, timing errors, and other artifacts unnoticed in the signal. Sometimes these are due to extra hardware/software steps, in particular, resampling, that may introduce aliased signals, and/or reduced frequency response, and possibly distortion into the signal.

In addition there may be artifacts in the signals, from switching power supplies and digital signals. In fact, it was through this analysis that we detected low-level interference in the analog stages of our first input device (ART USB Dual Pre). A professional grade card should not have these problems, and we have shown that a second USB device (Focusrite Scarlett 2i2 mk2) has complete “end-to-end accuracy”. There will be many others.

Transfer Function

We treat the DUT as a linear shift invariant system. That is, there is no amplitude distortion, such as compression. Further there is only one signal path (ie., no echos or reverberation are present). In such a system a pure tone, ie., a single sine wave, at the input results in the same tone at the output. That is, for input $x(t) = A_i \sin(2\pi f t)$, the output is $y(t) = A_o \sin(2\pi f t + \varphi)$, where f is the frequency, A_i and A_o are the input and output magnitudes and φ is the phase shift of the signal.

The transfer function (TF) is the gain and phase shift as a function of input frequency. We use $H(f)$ to denote the transfer function. The magnitude response is $|H(f)| = A_o(f) / A_i(f)$. We typically plot the TF in *decibels* (dB), ie., $20 \cdot \log_{10} |H(f)|$. A gain of unity is 0dB. A gain of ten is 20dB. For a gain of two, corresponding to one digital bit, we get $20 \cdot \log_{10} 2 = 6.02$ dB, approximately 6 dB.

Stimulus Signal

A measurement system is usually based on a specific stimulus signal. In the simplest case, a fixed series of tones could be used. For each frequency the magnitude and phase shift of the output are measured. A plot of magnitude vs frequency and phase vs. frequency at a discrete set of frequencies is sometimes called a *Bode plot* and most measurement systems provide this type of measurement.

A generalization of this is to use a continuously varying tone (increasing linearly or exponentially in frequency). This is called a “swept tone” method, and is the basis for the measurements in Part 1.

Alternately, a broadband signal (a continuous range of frequencies) may be used. This allows all frequencies to be evaluated at the same time. Note that when presenting several tones at once, it may become difficult to separate out different causes of distortion. For example, harmonic distortion (multiples of the fundamental frequency), while obvious for a single tone, will be difficult to see if multiple fundamental frequencies are present.

Usual broadband stimuli are typically white noise, where all frequencies have the same magnitude, or pink noise, where the energy (squared magnitude) falls off as $1/f$. Noise may also be band limited, for example, to octave or fractional octave ranges.

Periodic Stimulus for Acoustic Measurements

The key idea of our system is that, instead of providing an arbitrary stimulus, such as noise, the stimulus signal is periodic. For example, for a sample rate of $f_s=44100$ and a sample length of $N=16384$ samples we get a stimulus period of $T=0.37$ seconds. Period times for various powers of two are given in Table 1.

Points N (N-1)	$f_s=44100$	$f_s=48000$
8192 (8191)	0.19 s	0.17 s
16384 (16383)	0.37 s	0.34 s
32768 (32767)	0.74 s	0.68 s
65536 (65535)	1.49 s	1.37 s

Table 1. Stimulus times (in seconds) for $f_s=44100$ and $f_s=48000$. Stimulus length is N samples. N is a power of two. N samples are used for pure tones, and N-1 samples are used for MLS.

There are two motivations for choosing a periodic stimulus. First, since the signal is periodic, there is no need to apply a (time) window to the data before taking the FFT. As a result, there will be no spectral spreading. A pure tone will show up as a single bin in the frequency domain.

Second, as long as N samples are chosen, the exact timing is not important. This avoids any synchronization problems. As long as a periodic signal is output for N samples or longer, it does not matter at what exact point we acquire the samples. The only requirement is that the *ref* and *dut* channels are synchronized to each other and that they are all active during the chosen portion of the acquisition time.

With ideal synchronization, one period alone should be sufficient. However, it is often desirable to output data for some duration before data is acquired. This avoids transients which occur when the stimulus is applied. We typically delay 0.5 to 1.0 periods before collecting data. In addition, to allow for inexact length of data captured, we add 0.5 to 1.0 periods at the end of the stimulus. These parameters, along with other choices, appear at the beginning of the program, and should be configured for your system.

Software Implementation

Our goal is to implement an interactive interface with minimum programming overhead. While the high level features of Octave/Matlab allow easy access to interface features (plots, buttons, sliders, file dialog, etc), realtime interactive programming (and debugging) remains challenging. The main difficulty arises from the fact user input (mouse, keyboard) is asynchronous.. Button presses and mouse movements may happen at any time during the program operation. Each possible action must be specified by a special part of code, called a “callback”. A typical Matlab/Octave program will have a main program (a script or function) with numerous “helper functions” to handle the callbacks. Further complexities arise in sharing variables between functions, buttons and sliders on the GUI and data in the main program. There are also subtle differences in between Matlab and Octave.

We sidestep these issues by operating synchronously. The program runs in a continuous stimulus-measurement loop, and only accepts input at the beginning (or end) of the stimulus cycle. Operation is controlled by a single interface window, which shows the running state of the program as a series of “radio buttons” at the left hand edge. (See screenshot, Figure 2). At any point the user can click a “radio button” to control the program. Once the current stimulus-measurement cycle is complete, the program will change state.

On startup (or change of Stimulus) a dialog is presented and the user may select a *stereo WAV file*. The file contains N points, which may or may not be a power of two. Typically the left and right output channels are equal, however, different signal may be used for each channel. This will allow our system to drive other types of systems, such as balanced bridged or quadrature inputs, in the future.

Note that the sample rate is taken from the file. This allows us to avoid digital resampling. Most products will work at multiple sample rates, however, a few systems are fixed at 44.1kHz (standard for CD music production) or 48kHz (standard for video production). We provide two types of test signal: sinusoid and MLS, along with programs to generate 44.1K and 48K WAV files. These programs could be modified for higher sample rates (88.2K, 96K and higher), provided the sound card drivers support them.

We provide two programs to generate stimulus signals. The first program, “make_sig_tones.m”, generates a single test tone (A4, ~440 Hz). Note that for a given stimulus length (N) and sampling rate (f_s), we have a discrete set of periodic tones, $f[k] = k * f_s / N$, where $k=0..N/2$ is the frequency index. For a desired frequency, say $f_0 = 440\text{Hz}$, we will chose the nearest integer $k=\text{round}(N*f_0/f_s)$. In the

case of $f_s=44100$ and $N=16384$ we get $k=163$, $f[163]=438.74\text{Hz} \sim 440\text{Hz}$. By selecting this particular frequency, the tone (and all its harmonics) will land on single bins in the FFT.

A second program, “make_sig_mls.m”, generates a special sequence, called a “maximum length sequence” or MLS. The MLS sequence is noteworthy in that it always has $N=2^n - 1$ points. That is, it has one point less than a “power of two” sequence. In principle, computing the FFT of such a sequence is much slower than $O(N \log N)$ ¹, but for fast computers (and reasonable N) this is not an issue.

The MLS sequence has several important properties. First it is a sequence with only two values, -1 and 1 . There are no intermediate values. Second, all frequencies are present, and all frequencies have exactly the same magnitude. Third, the phase of the signals is pseudo random, with no obvious coherent structure. Finally, the DC component is (almost) zero. In addition to a white MLS sequence, the above program also generates a pink noise version. The pink noise is obtained by generating the sequence $\text{mls}[n]$, $n=1..N$, taking the FFT, and scaling the coefficients as follows:

```
MLS[k] = fft(mls[n])
```

```
pink_mls[n] = ifft( MLS./ sqrt(abs(k)) )
```

where k is the frequency index ($-K \leq k < K$) and $K=\text{floor}(N/2)$.

Octave Software Setup

The programs need Octave 4.0 or greater. Release 4 provides the latest GUI features and an interactive editor that is common across Linux/Windows and Mac. The instructions below are specific for Ubuntu Linux 16.04 LTS (long term stable). Other distributions should be similar. We install Octave with the following commands:

```
# sudo apt-get install octave
```

```
# sudo apt-get install octave-signal
```

Start Octave and check the version

```
>> version
```

```
ans = 4.0.2
```

Our software needs the “signal” package. Like Matlab’s Toolboxes, packages provide extra function, in this case, all the signal processing, linear filtering and FFT code. Octave toolboxes can be loaded as part of the Linux distribution, or from a central repository called “Octave Forge”.

Octave has a simple “package manager” at the command line. To load the signal package, type the following,

```
>> pkg list
```

```
>> pkg load signal
```

If packages are not available for your distribution, or if you want the latest from the Forge, the Signal can be compiled from source with the following command.

```
>> pkg install -forge signal
```

Note that this command requires an internet connection and also requires the Octave development libraries (liboctave-dev package).

Running the program

The first step is creating the stimulus files. We recommend changing to a new directory, as many files are written. Once in a new directory, run the programs “make_sig_tones.m” and “make_sig_mls.m”. These programs generate pure tone (440Hz) and MLS and pink MLS files, respectively.

If you're running Linux, you can run Octave scripts from the command line directly, with the commands

```
octave make_sig_tones.m
```

```
octave make_sig_mls.m
```

Each command invokes an Octave shell, runs the script file, and exits. Octave scripts are useful for short programs. Windows users will likely just start an interactive Octave session and run the commands at the Octave prompt.

1 $O(N \log N)$ is “big O” notation, which describes the asymptotic scaling of the run time of the FFT (Fast Fourier Transform) algorithm, in terms of N , the number of data points.

Now you should have a bunch of stereo 16 bit WAV files. These are stereo 44.1k and 48k 16bit. Recall that each stimulus file has only one period, a second or less of sound. You can still load these into a WAV editor to view them interactively, and the reader is encouraged to do this.

The user can now run the main program SoundCardSetup.m.

A number of windows will pop up, including a dialog to choose the stimulus file.

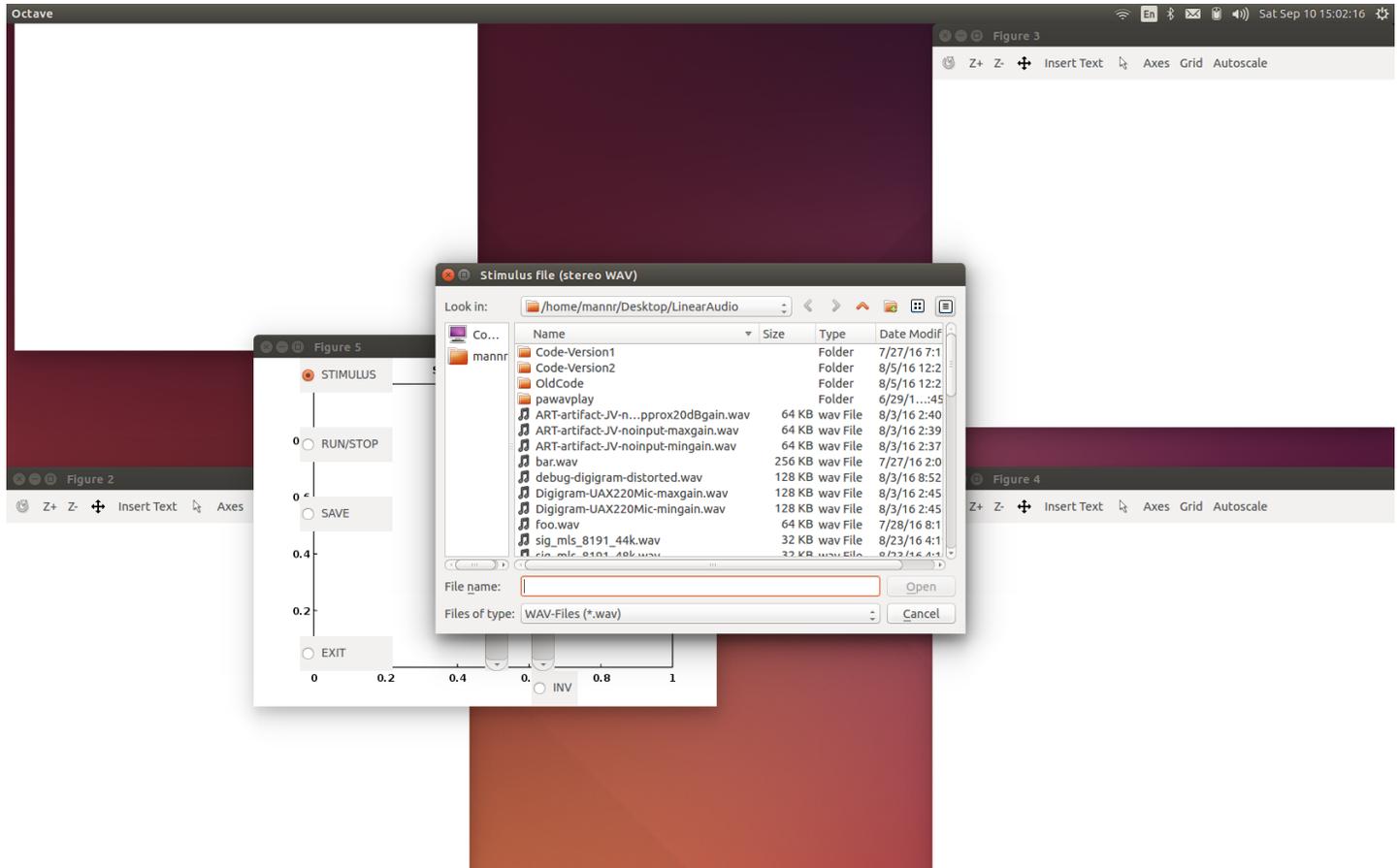


Figure 2. Screen shot of program startup. Dialog prompts for stimulus file.

The running system is shown in the screen shot below. We begin with the case with a pure tone stimulus (440Hz, 16384 points, 44.1k sampling rate). We have connected the *ref* (left) signal by “looping back” the left output to the left input using a standard 1/4” TS (tip sleeve) unbalanced “patch cord”. Balanced cables may be used as well. The *dut* (right) channel is left unconnected. The device is the ART USB Dual Pre.

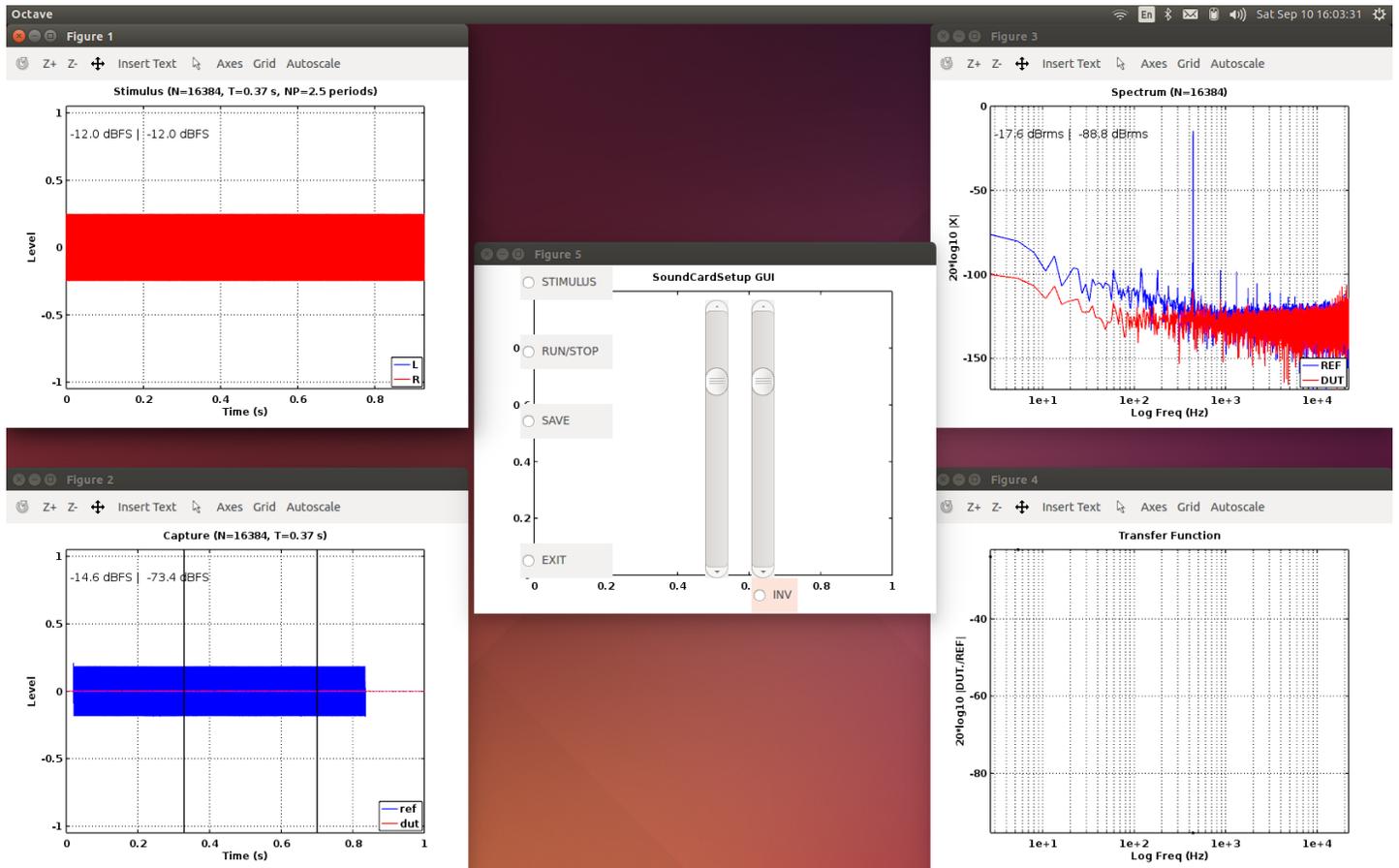


Figure 3. Running program. The GUI, with buttons and sliders appears at the center of the screen. The stimulus is a pure tone (440Hz) with $N=16384$ samples. REF signal (blue, left channel) is looped back. The DUT input (red, right channel) is not connected.

The windows in are as follows:

Stimulus (upper left)

This is the output stimulus, in this case a $\sim 440\text{Hz}$ sine wave. The stimulus output level defaults to 0.25, which corresponds to -12dBFS . dBFS refers to decibels, where FS denotes “full scale”, ie., 1.0. Note that both the left (blue) and right (red) outputs are active, however, only the right (red) is visible because the plots are overlaid.

Capture (lower left)

This is the signal captured by the sound card, with REF (left) shown in blue and DUT (right) shown in red. The levels are the peak values observed within the period (the two black vertical black lines). dBFS refers to “decibels full scale”, where full scale 0dB, is ± 1 on the inputs.

Note the levels on the loopback signal, at -14.6dB , is -2.6 dB below the stimulus (-12dB). This is the electrical gain of the circuit going from output back to input. The actual value observed will depend on the input and output gains and the DAC and ADC sensitivity.

Note that that there is a short delay at the start, and some missing signal at the end. Further, the exact timing will “jitter” a bit with each update. This is all OK as long as the data, exactly $N=16384$ points in this case, is stable between the two vertical black lines. If there are timing problems, you will have to modify the timing parameters at the beginning of the program.

Spectrum (upper right)

This is the log power spectrum of the DFT of the input data. Signal levels are reported as rms values, dBrms. Therefore a single sine wave of unit magnitude (full digital level) has a value of -3dBrms in the spectrum. In this case the level -17.6dBrms in the spectrum, is 3dB down from the capture input dBFS level, as expected.

GUI window (center of screen).

The program is controlled by a single GUI window. The “Radio buttons” on the left side show which State the program is in. Once the Stimulus file is loaded, the program goes to RUN mode. The screen is continuously updated.

Hit the RUN/STOP button to pause the program.

Note that when stopped, the user may then interact with the Octave figures, to zoom in on plots, do screen/figure capture, etc. This is useful to examine signal details.

The SAVE button pulls up a dialog to save the result. This is saved as a stereo WAV file, with N samples. The left channel is REF. The right channel is DUT.

The STIMULUS button pauses the system and inputs a new Stimulus.

Finally EXIT terminates to program.

We have also added sliders to allow setting of stimulus output levels (between 0 and 1). Finally, the INV button allows the DUT/right channel to be inverted (this reduces cross-talk on some cards-see Part 1). Note: each time a slider is moved, the new output values are printed to the standard output (the console where SoundCardSetup was run).

The next screen shot shows the system with an MLS stimulus (16383 points, 44.1kHz) and with both the left and right channels looped back. In this case both the REF and DUT signals are stimulated with an MLS signal. Ideally the spectrum should be flat. The low frequency rolloff is due both to the digital high pass filter (HPF) in the sound card, and its coupling capacitors. The response also falls off at high frequencies due to the anti-alias (AA) filter. Note that both REF and DUT channels are nearly identical, indicating that the sound card has matched L and R inputs and outputs.

The transfer function appears in the lower right window. The transfer function (TF2, as defined in Part 1) is found by dividing the (magnitudes) of the Discrete Fourier Transform (DFT) of the DUT by the REF. Each dot in the figure corresponds to a bin (frequency index) in the DFT. According to the figure the left and right channels are within 0.1dB between 20 Hz and 20 kHz.

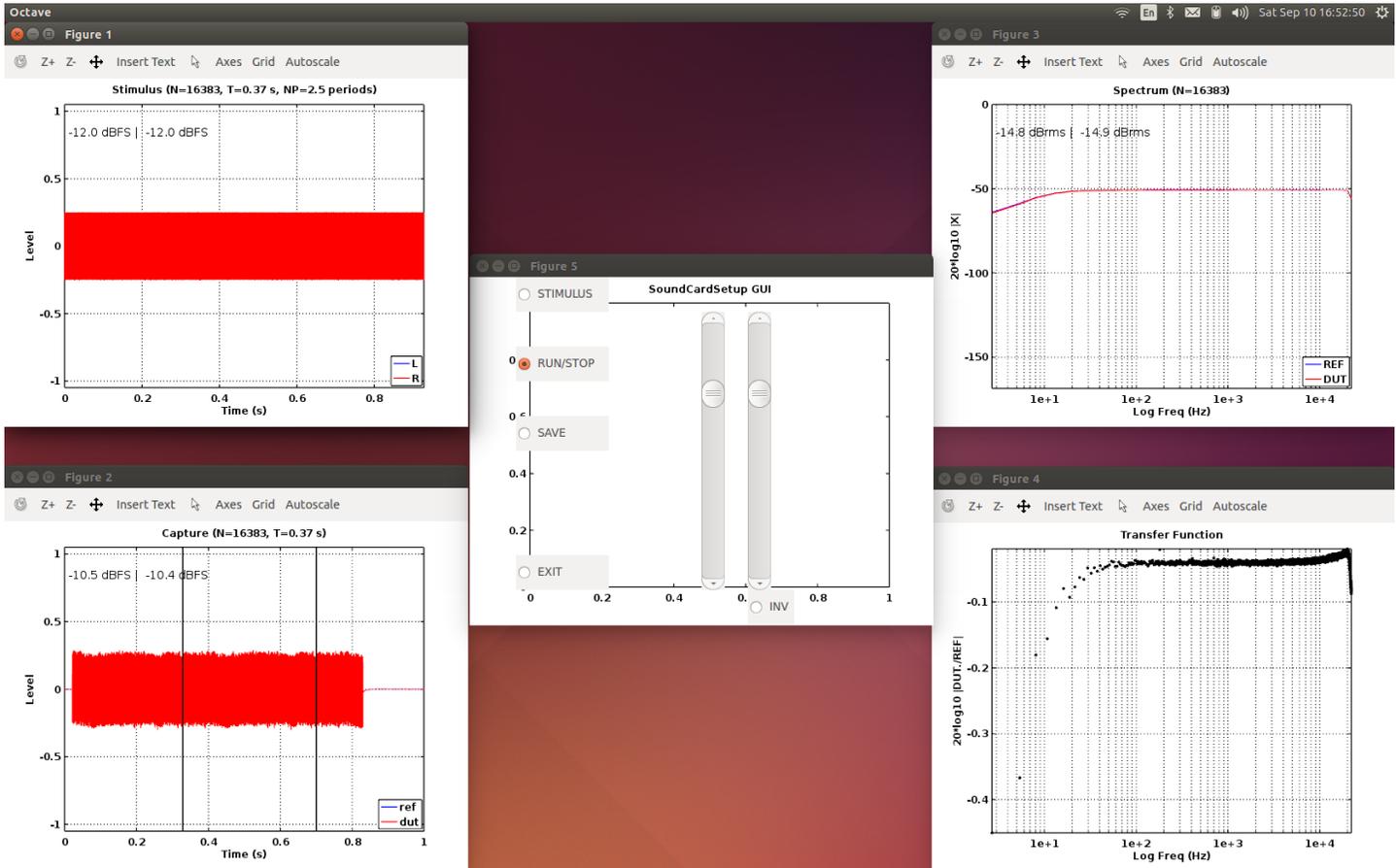


Figure 4. MLS stimulus. DUT is looped back. Transfer function (TF) is shown at lower right.

Programming Details

Ideally our program would output a (continuous) periodic stimulus, and while the output was playing, record inputs simultaneously to a circular buffer. Provided the FFT processing and plotting were fast enough, the display could be updated every $T = N/f_s$ seconds.

Unfortunately Octave (and Matlab) do not provide such fine-grained control on the audio input. In particular, audio input and output commands in Octave are restricted to single, finite duration intervals. Further, there is some (system dependent) delay between when the recorder starts and when the player starts. As a result, our strategy is to output more than one period of the sequence, and to select a single period within the capture where the data is stable.

The capture is embedded in a while loop, using the statements below. Similar code is used in the programs listed in Part 1, so it is worth reviewing the code used.

```

1. rec = audiorecorder(fs,16,2); % Set up audio recorder at sampling rate fs
2. ply = audioplayer(y,fs,16); % Queue audio player with signal y. (n rows by 2 columns)

3. play(ply); %this should play and allow immediate further execution
4. recordblocking(rec,Ts+Tdelay); % waits until all samples recorded

5. z = getaudiodata(rec,'double');% this retrieves the recorded data
6. ns=size(z,1); % total number of samples acquired

```

The first two lines set up the recorder and player objects (fs = sample rate, 16 bits per sample, 2 channels). The third line cues the data (y). Line three starts the player (this has some delay) and line four immediately starts the recorder and waits for the recorder to finish.

T_s is the length of the sequence. T_{delay} is a fudge factor. This is the time delay between when the recorder starts and when the player actually starts. This varies depending on the platform and the device drivers. The user will have to experiment with this number. We observed approximately 0.05 sec for Octave/Linux and approximately 0.3 sec for Octave/Windows 7. Finally line 5 extracts the data from the audio stream and converts each 16 bit sample to floating point (ie., normalized to be between -1.0 and 1.0).

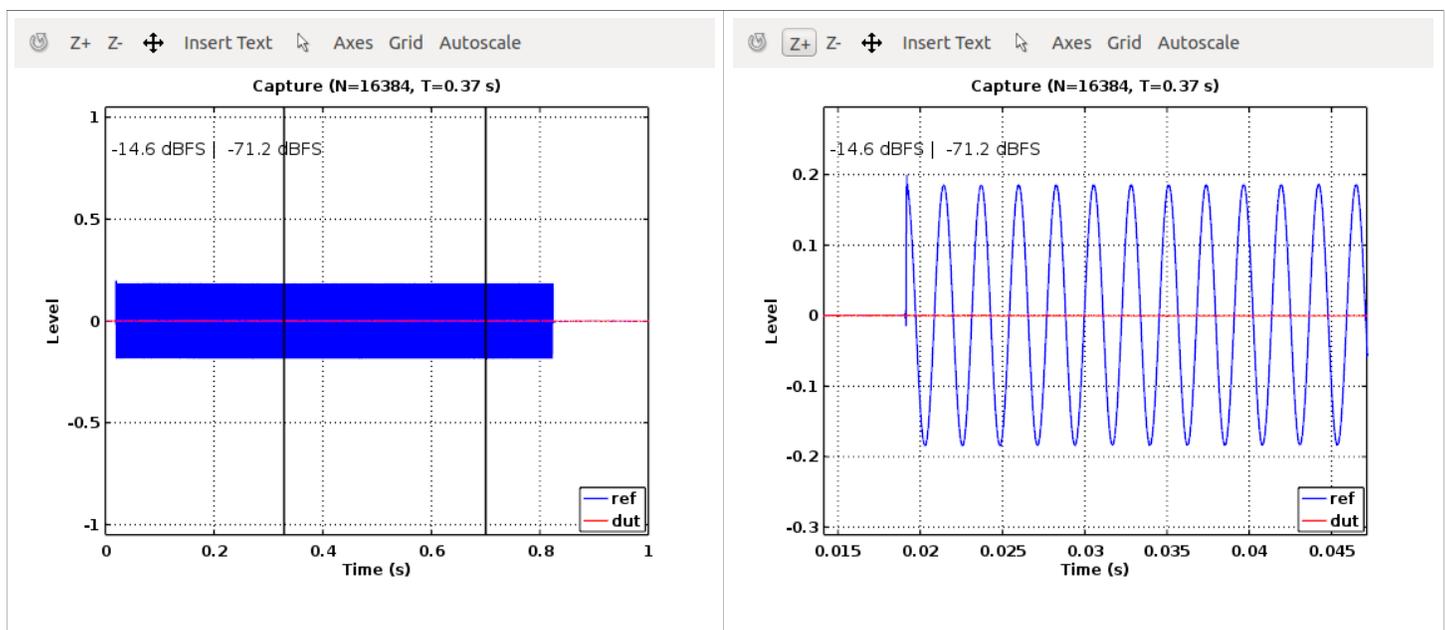
The following variables must be set (beginning of program). For Octave/Linux, the values are:

$NP = 2.5$; % number of periods to output

$Nofs = 0.5$; % offset (in periods) to start recording

$Tdelay = 0.05$; % delay (seconds)

Before describing the applications, we note some timing issues (Figure 6). The captured data is shown at the upper left. Zooming in at the beginning of the captured signal, there is a brief transient (upper right). As long as we capture away from the start and end of the sequence, the signal should be free of transients. Finally we note that while the exact delay/phase shift of the signal is variable, if exactly N samples are acquired, the stimulus phase is equal at the start and the end of the period. Compare the bottom row, showing the start and the end of the capture.



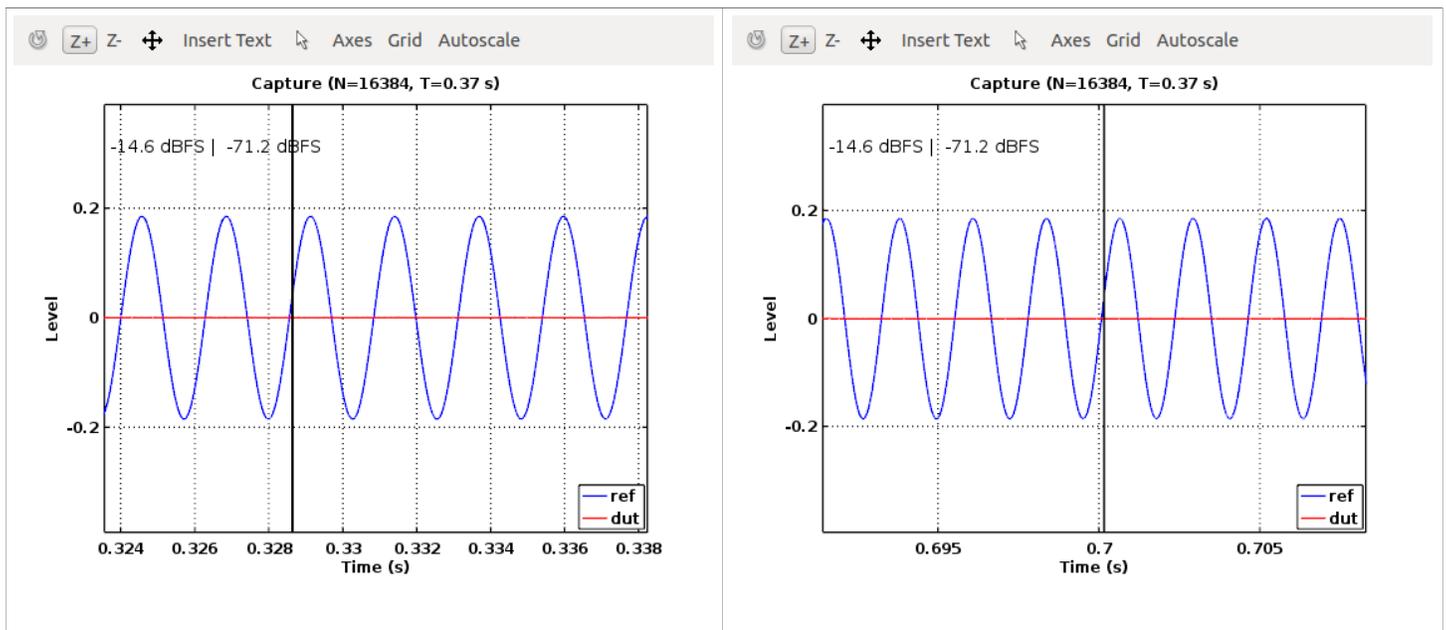


Figure 5: Timing details. Upper left: Captured signal (blue = left channel = REF, red = right channel = DUT). Upper right: Detail, transient at start of captured signal. Lower left: detail, start of signal, Lower right: end of signal.

Testing Your System

The pure tone should be the first test of your system. A single tone and only that tone should show up in the spectrum. Distortion and noise will both be easily visible, and if necessary, we can set levels very precisely using a single tone.

Figure 6 shows the spectrum for the tone (440Hz). The zoom view on the right shows detail of the expected tone (440Hz) along with the harmonic distortion (880,1320, 1760, ...). Note that because the input is periodic, those fall in single bins in the DFT and show up as single, sharp lines in the spectrum. Given such a periodic stimulus, harmonic distortion can be measured directly by the amplitude of these components in the spectrum, relative to that of the fundamental.

We also note a few extra peaks at multiples of 1kHz, which we believe to be artifacts from the USB interace. The device shown below is the ART USB Dual Pre interface, but we have observed these in some other interfaces too.

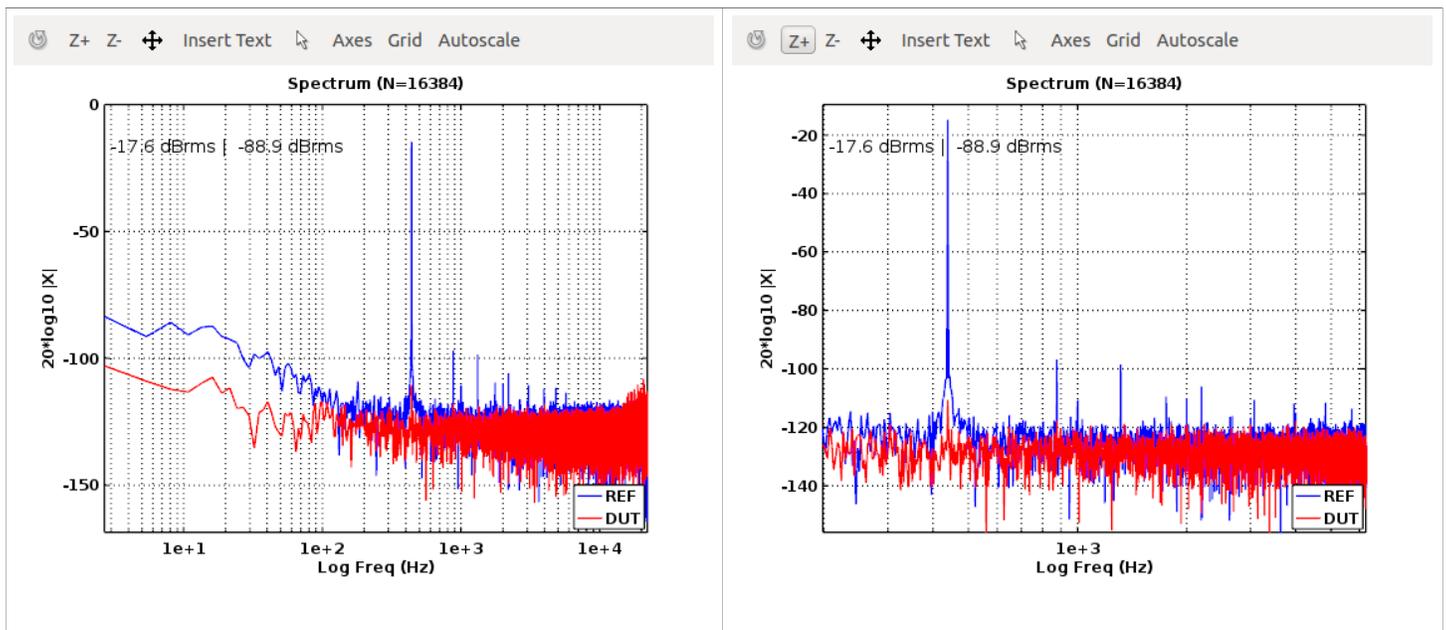


Figure 6. Pure tone (440Hz) stimulus on REF input. DUT (red) is not connected. Right panel is a zoom of the frequency range. Note the extra peaks at 1kHz and multiples of 1kHz.

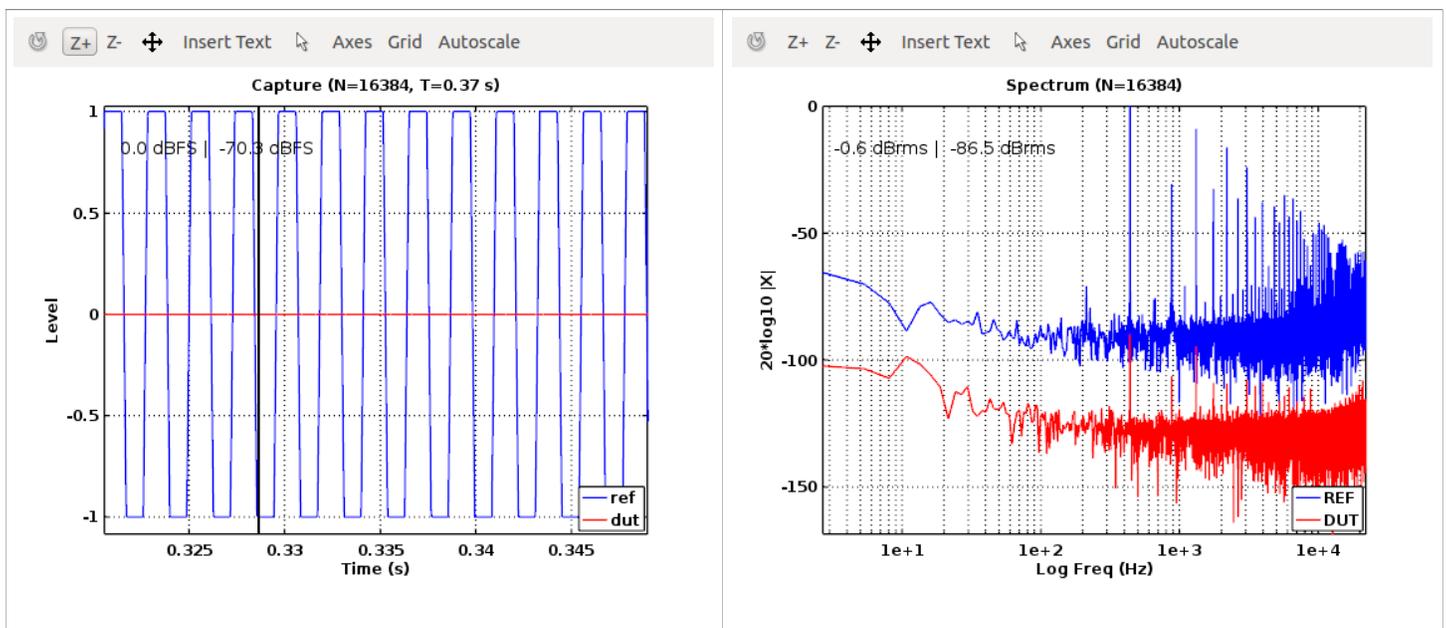


Figure 7. Clipping for pure tone. REF (left channel, in blue) is looped back. DUT (right channel, in red) is not connected. There is significant cross talk in the right channel.

Figure 7 shows harmonic distortion due to clipping. On the left is the input sine wave, with so much gain that it becomes almost a square wave. The harmonic peaks fall on single bins of the DFT at the right. A natural extension is to add a computation of harmonic distortion.

We can measure the frequency response of the sound card using an MLS signal. Figure 8 shows spectrum for a 48kHz stimulus, with both the REF and DUT channels connected (looped back). The left panel is for the ART USB DualPre. The right panel is for the Focusrite Scarlett 2i2. Note that the response falls off at low frequencies due to digital and analog high pass filtering. The response falls off at high frequencies due to the anti alias (AA) filter. Note that while both cards pass signal, there are very small ripples in the ART due to the different type of digital filter.

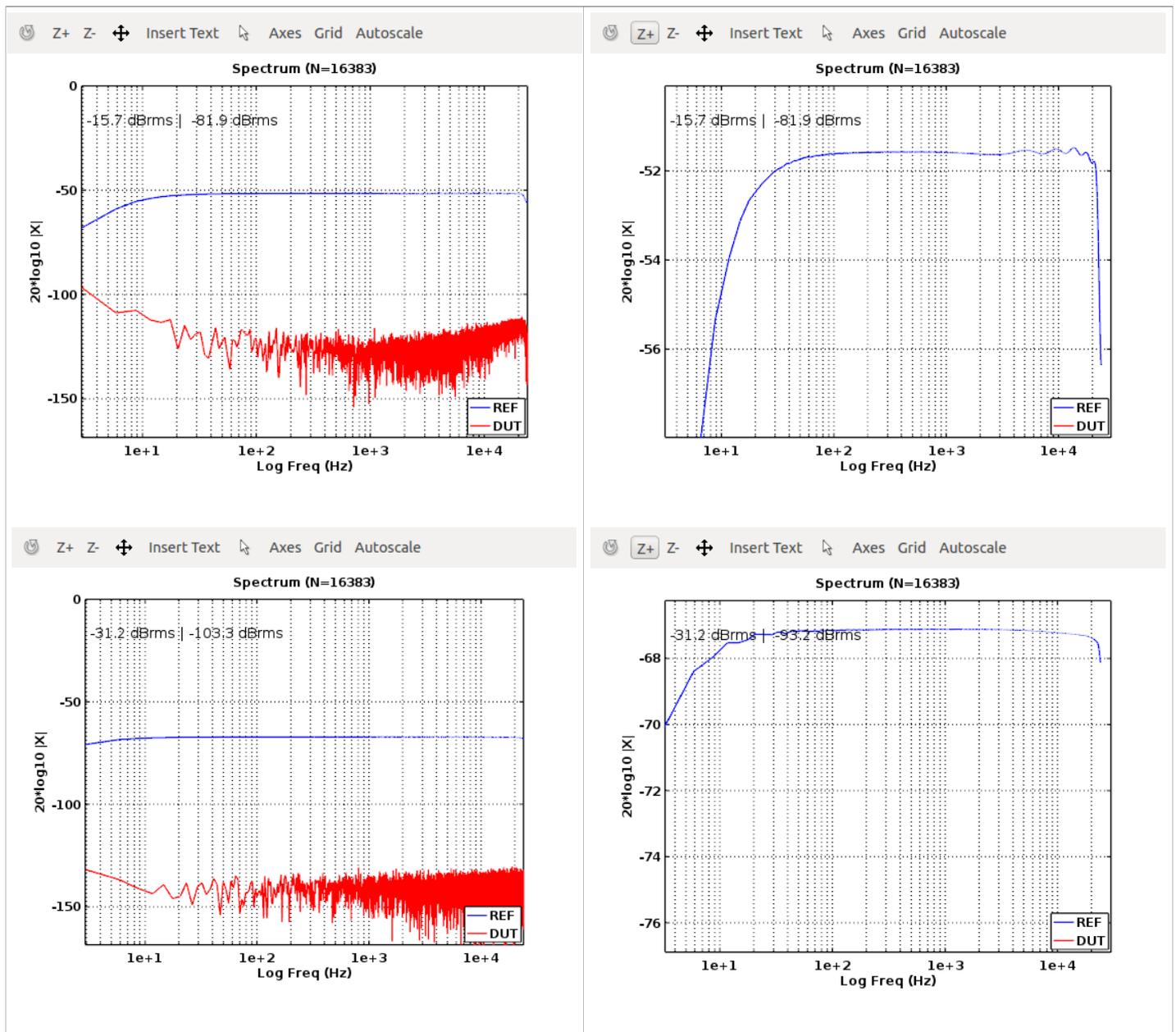


Figure 8. Frequency response (for REF, loopback). Top. ART USB DualPre (44k) Bottom Focusrite Scarlett 2i2 (48k). Right shows vertical axis zoomed in.

Calibration

Following the notation in Part 1, the signal chain is denoted as follows

$$\begin{array}{ccccccc}
 \text{Digital output} & \rightarrow & \text{DAC} & \rightarrow & \text{DUT} & \rightarrow & \text{ADC} \rightarrow \text{Digital input.} \\
 D_o (\#) & & V_o (\text{Volts}) & & V_i (\text{V}) & & D_i (\#)
 \end{array}$$

In acoustics measurements the DUT consists of a speaker to convert the signal from an electrical signal (voltage) to sound pressure (in Pascals), followed by a microphone to convert the sound pressure back to an electrical signal (voltage). For purely electronic measurements the DUT consists of an electrical device, such as an amplifier, filter, equalizer, etc.

The DAC converts the output signal D_o (digital units, from -1 to 1) to output voltage V_o using the relation $V_o = D_o * S_{DAC}$. S_{DAC} is the output sensitivity of the DAC, measured in Volts per digital unit. The ADC converts the input voltage V_i to a digital value, using the relation $D_i = V_i / S_{ADC}$, where S_{ADC} is the input sensitivity, measured in Volts per digital unit.

S_{DAC} is measured by outputting a full scale (0dB) sine wave and measuring the (peak) voltage with an AC voltmeter. Table 2 below shows the values obtained using a National Instruments myDAQ Virtual Instruments Oscilloscope to this voltage. In each case the output gain (if available) was set to maximum.

	UCA 202	ART USB DualPre	Focusrite Scarlett 2i2
S_{DAC}	1.9 V	1.0 V	1.7 V

Table 2. DAC output voltages.

S_{ADC} is determined by outputting a known digital output level (D_o), looping the output back to the input, and measuring the digital input (D_i) from the Capture display window. Looping output to input we have: $V_o = V_i$. That is, $V_o = D_o * S_{DAC} = V_i = D_i * S_{ADC}$. Which yields: $S_{ADC} = S_{DAC} D_i / D_o$.

The tables below show the computations. Note that the digital levels and the gain are read in decibels. If G is the gain in decibels, the linear ratio D_i / D_o , is computed as $10^{-G/20}$.

Behringer UCA 202/222	Level
Digital output in dB: $20 \log_{10}(D_o)$	0 dB
Digital input in dB: $20 \log_{10}(D_i)$	-0.3 dB
Digital gain in dB: $20 \log_{10}(D_i / D_o)$	-0.3 dB
$S_{ADC} = S_{DAC} 10^{-Gain/20}$	1.97 V

Table 3. Gain for Behringer UCA 202/222. The gain is fixed and very close to unity.

ART USB Dual Pre	Minimum gain	50% gain	Maximum gain
$20 \log_{10}(D_o)$	-8.6 dB	-8.6 dB	-39.3 dB
$20 \log_{10}(D_i)$	-12.9 dB	-1.7 dB	-2.1 dB
$20 \log_{10}(D_i / D_o)$	-4.3 dB	6.9 dB	37.2 dB
$S_{ADC} = S_{DAC} 10^{-Gain/20}$	1.64 V	0.45 V	0.014 V

Table 4. Gain for the ART USB Dual Pre. Note that the XLR and TRS connectors have the same gain.

The gain range on the ART Dual Pre is approximately 40dB. We also note that control is highly nonlinear, with most of the gain occurring near the maximum level on the trim pot. Both XLR (mic) and TRS (line) have the same gain structure.

Focusrite Scarlett 2i2 (line/inst/mic)	Minimum gain	50% gain	Maximum gain
$20 \log_{10}(D_o)$	-39.3 dB	-39.3 dB	-39.3 dB
$20 \log_{10}(D_i)$	-58.3 dB (-49.5dB , -39.8dB)	-35.9 dB	-2.1 dB
$20 \log_{10}(D_i / D_o)$	-19 dB (-10.2dB, -0.5 dB)	+3.4 dB	37.2 dB
$S_{ADC} = S_{DAC} 10^{-Gain/20}$	15.1 V (5.5V, 1.6V)	1.15 V	0.023 V

Table 5. Gain for the Focusrite Scarlett 2i2. Line input. For Minimum gain, Line/Instrument and Mic levels are also shown. The values in brackets are for the instrument and mic inputs, respectively. These values are only shown for the lowest gain setting.

The gain range of the Focusrite Scarlett is approximately 50dB. The gain control is also more linear than the ART.

The values in brackets show the Focusrite values the Instrument inputs (1/4" TRS) and mic inputs (XLR) for minimum gain setting. The Line input has approx -20dB of gain, the instrument input approximately -10dB and mic input has approximately 0dB gain at the minimum setting.

Noise floor

The minimum possible “noise floor” is obtained by shorting pins 2 and 3 on the input XLR connector. This prevents the inputs from picking up any interference, or any noise associated with the sensor (microphone) connected to it. In addition, the output levels are set to zero to avoid crosstalk between the output and input stages.

The following figures show the noise floor on both inputs. In this and future figures, left (blue) is set to minimum gain, and right (red) is set to maximum gain. Adding (approximately) 40dB of gain has increased the noise floor by only 6dB.

Below on the right is the noise floor for the Focusrite Scarlett 2i2. The ADC has excellent performance, however, at maximum gain (~50 dB), the microphone preamplifier adds approximately 20dB to the noise floor.

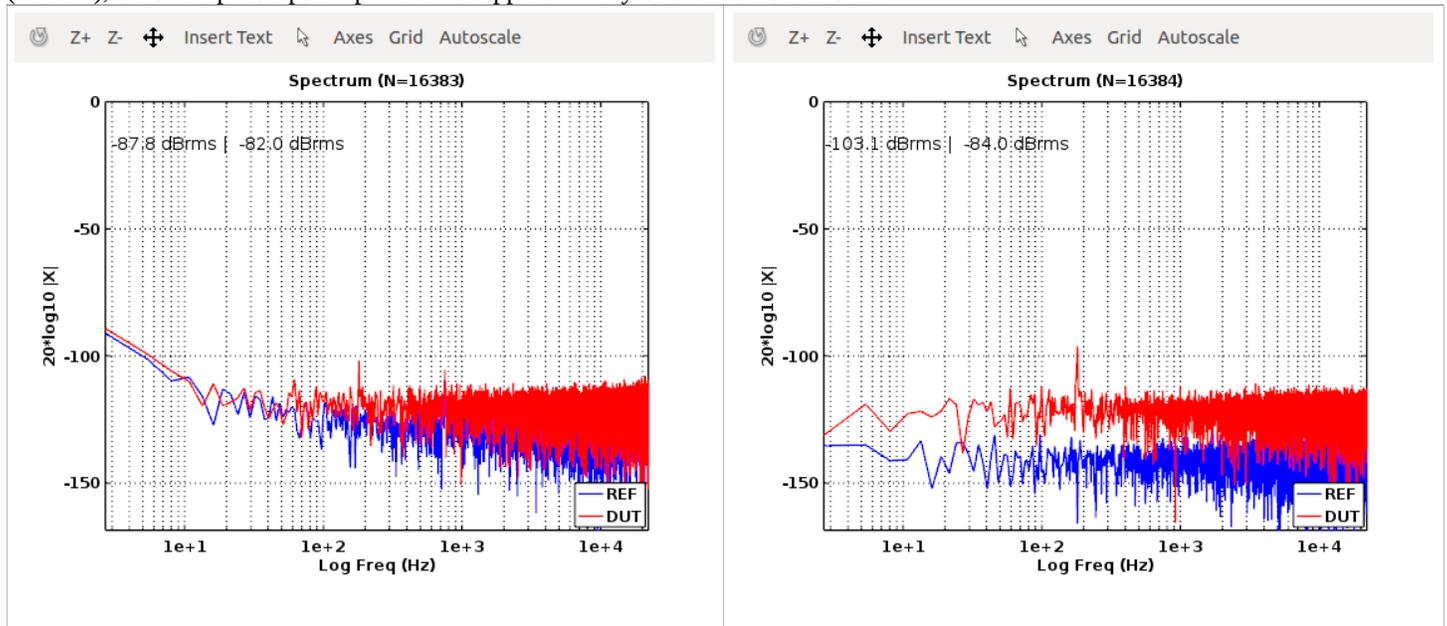


Figure: 9. Noise floor measurements for Focusrite (left) and ART USB DualPre (right).

Measurement Examples: Electronic Devices

In addition to speaker measurements, described in Part 1, spectral analysis reveals the performance of electronic devices. Here we consider their testing and show a few examples.

First consider we consider a “line matching” transformer. This particular device, Shure Model No A95, takes a high impedance (unbalanced line level) on a 1/4” TS connector, and transforms it to a low impedance, balanced input on an XLR connector. The spectrum appears to be very flat, except for a slight loss at low frequencies. The insertion loss is ~22 to 23 dB. Of course, the user must be careful when testing such devices. The test here is done at only low voltage levels (one volt or so) on consumer line level outputs. Performance may change if the input voltages are higher and/or input or output impedances change.

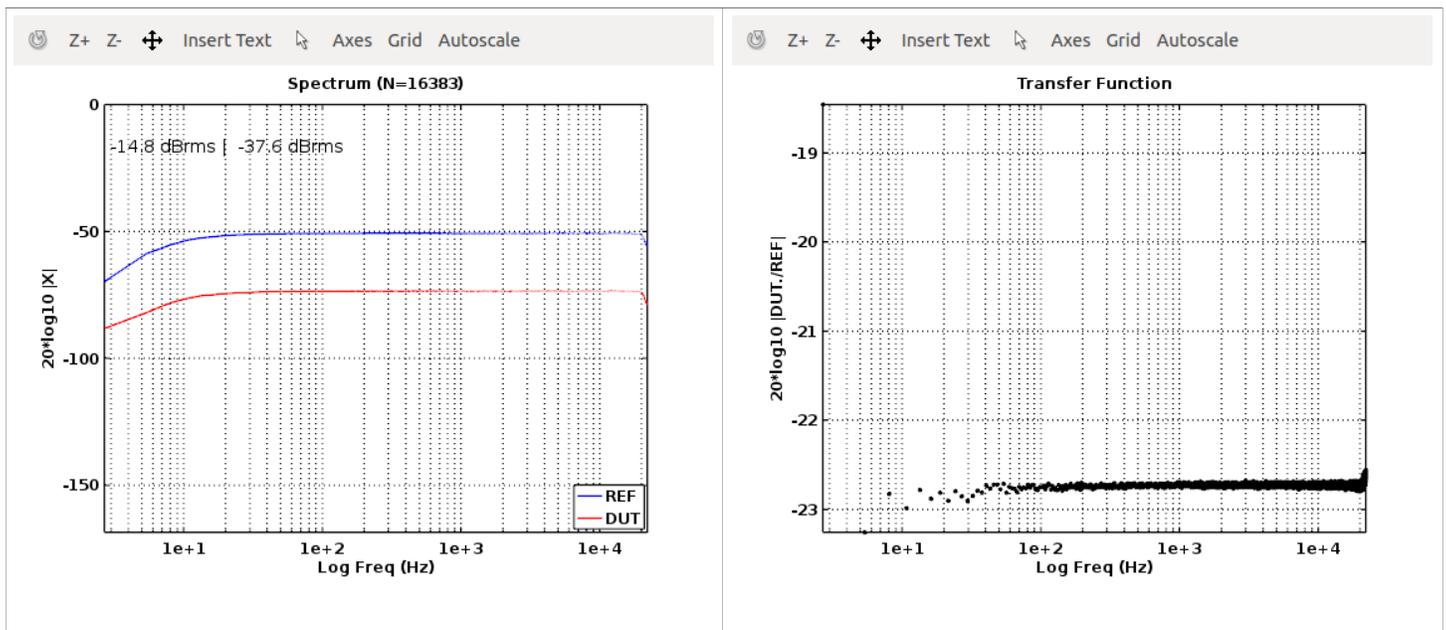


Figure 10. Transfer function for Shure Model No A95 matching transformer.

Next we test a 32-band analog equalizer (Klark Teknik DN 30/30 Graphic Equalizer). The equalizer has 31 sliders, at the standard 1/3 octave frequencies, from 25 Hz all the way up to 20 kHz.

Figure 11 (left plot) shows the spectrum for the equalizer set to “bypass”. Note that REF (blue) and DUT (red) are both flat, and both fall off at low frequencies. The falloff of the REF channel is due to the high pass filter (HPF) in the sound card. The fall off of the DUT is due to both the sound card and the equalizer itself. The right plot shows the transfer function (TF). Each dot of the transfer function corresponds to a frequency bin of the DFT. Ideally the equalizer has unity gain, approx. +1dB is measured. Note that the TF becomes unreliable at high frequencies. This could be fixed by adapting the threshold on the input levels, if desired.

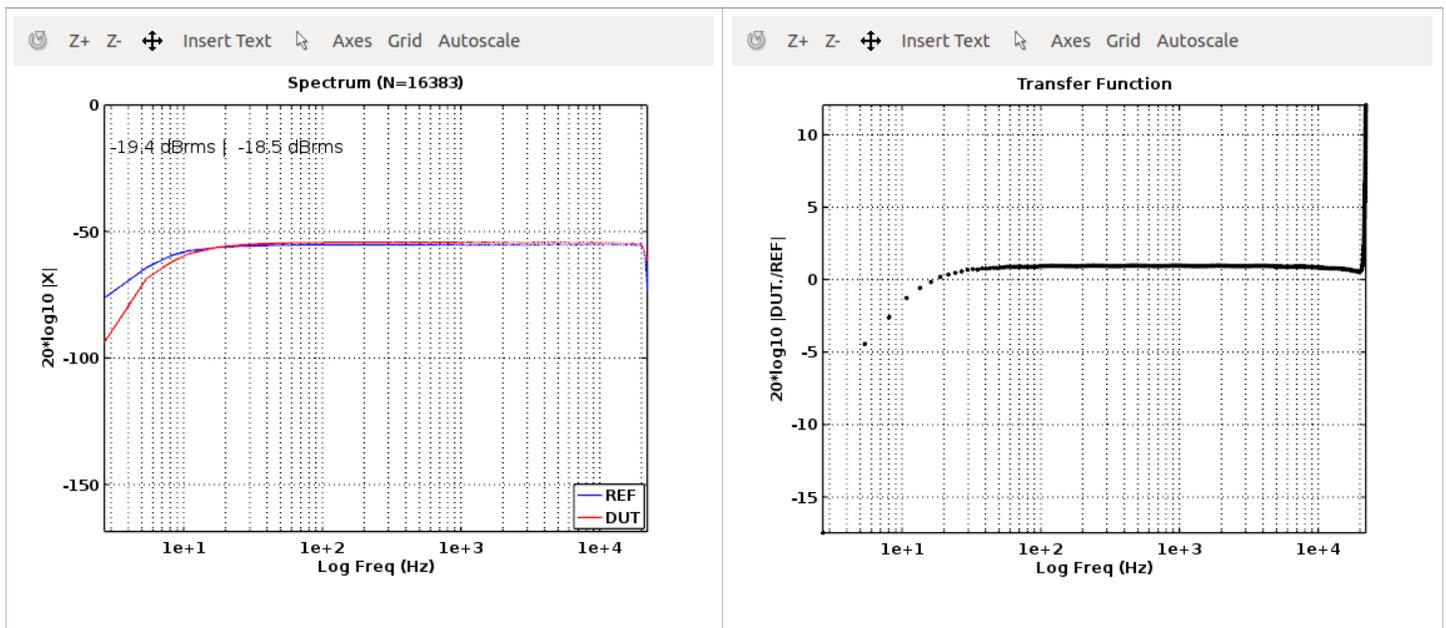


Figure 11. Transfer function for the Klark Teknik DN 30/30 Graphic Equalizer (all sliders set to 0dB).

Figure 12 shows the response for a single slider, 1kHz, set to +12dB gain. Figure 13 shows the response for every third slider set to +12dB gain. The peaks correspond to the standard octave frequencies (31.5, 63, 125, 250, 500, 1000, ...). Note that both the gain and bandwidth of the filters are consistent over the entire frequency range. The equalizer is performing correctly.

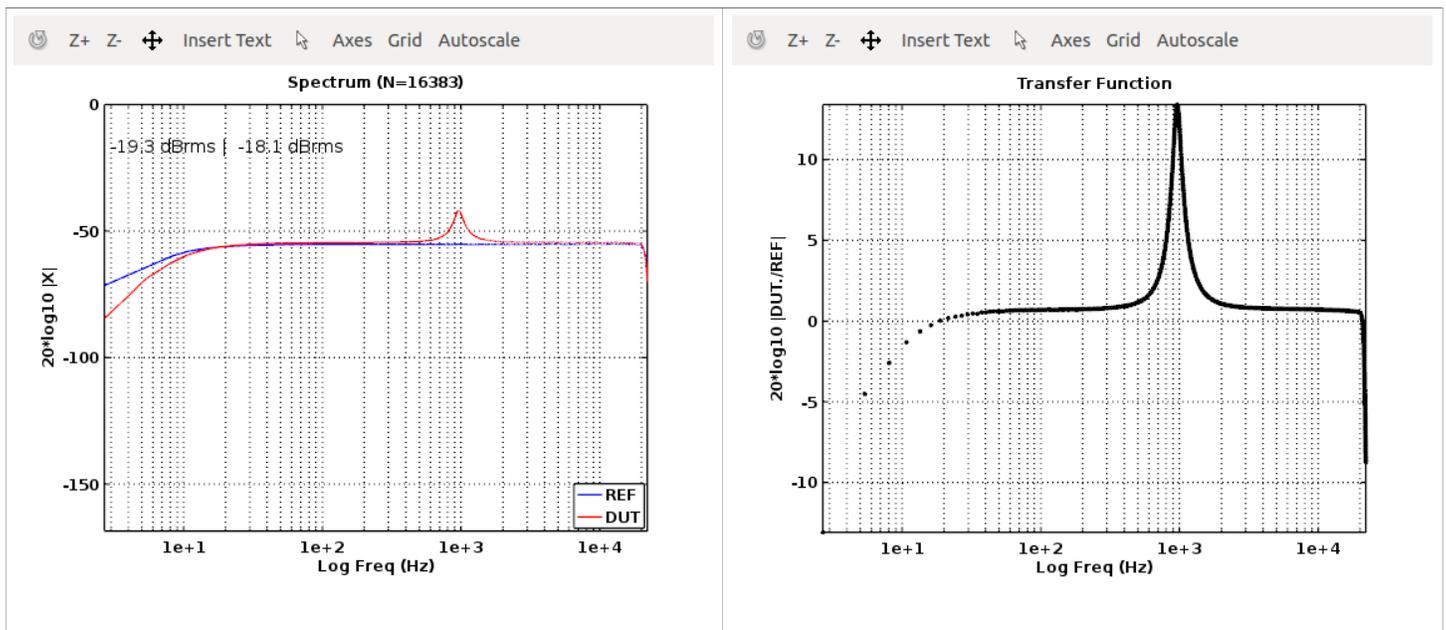


Figure 12. Transfer function for the Klark Teknik DN 30/30 Graphic Equalizer (1kHz band at maximum, +12dB).

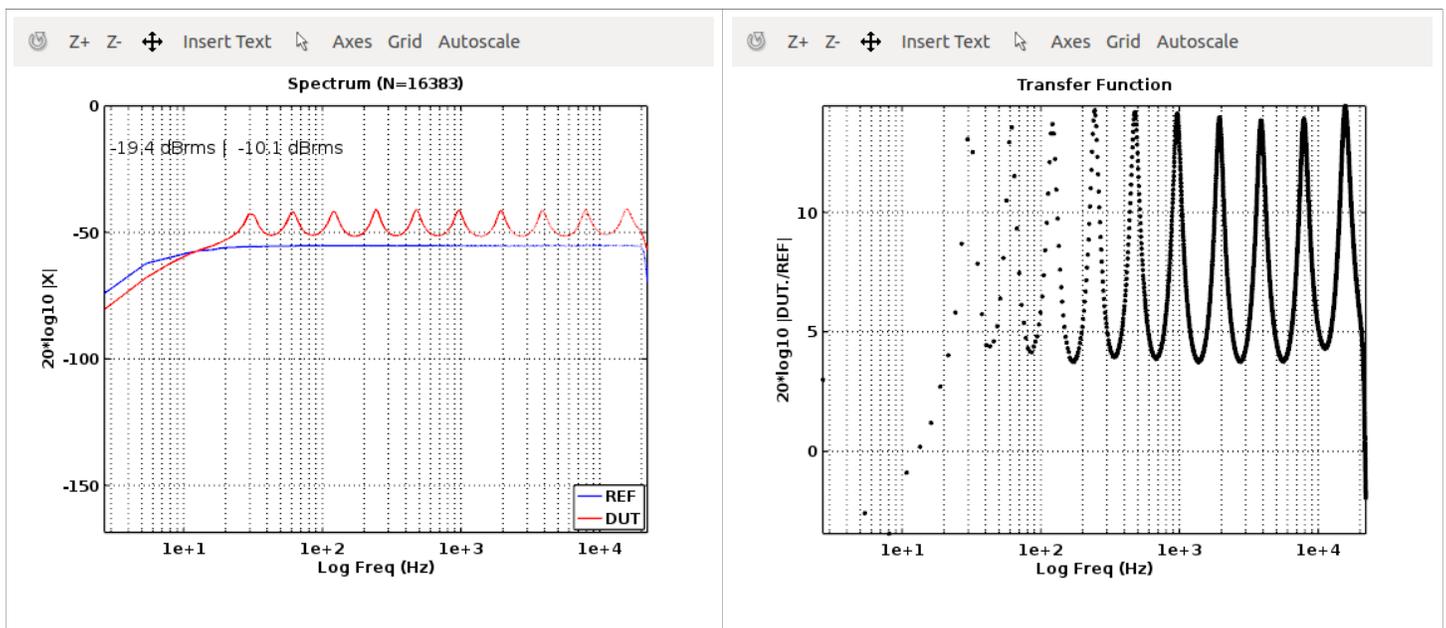


Figure 13. Transfer function for the Klark Teknik DN 30/30 Graphic Equalizer (1kHz band at maximum, +12dB).

Finally we test a Digitech WHAMMY Guitar effects pedal. This is a digital processor that takes an input and shifts it up or down by one or two octaves, depending on the setting. To test the pedal we input a pure tone (440Hz) and compare the REF and DUT spectrum. Figure 14 shows the input (blue) and output (red) signals for one octave up, two octaves up, one octave down and two octaves down. In each case the pure tone is shifted in the spectrum. The magnitude of the peak remains approximately the same. Note the peak is no longer sharp after going through the device.

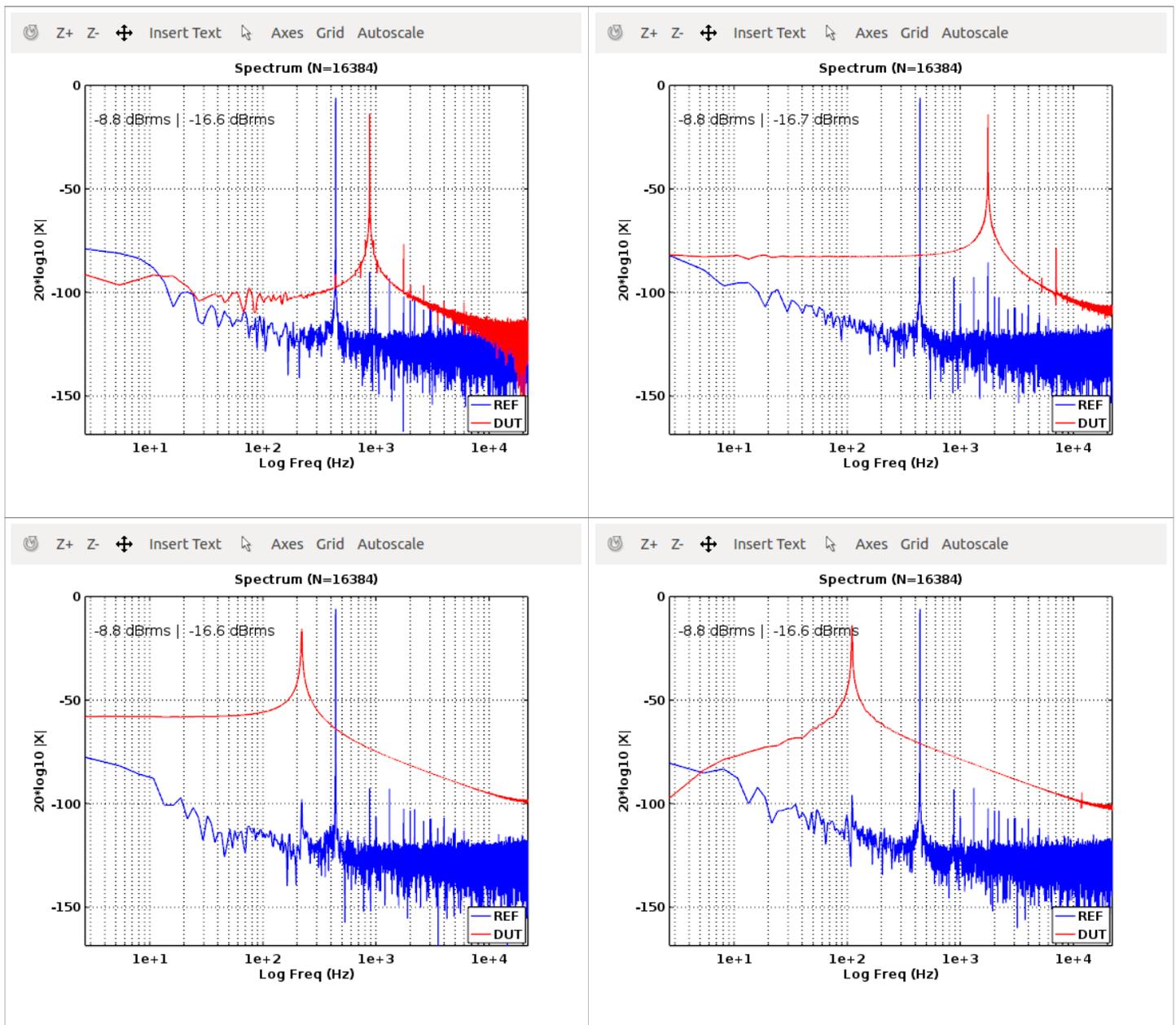


Figure 14. Transfer function for Digitech WHAMMY Guitar effects pedal. Upper left: no effect, Upper right: double frequency, Lower left: half frequency, Lower right: quarter frequency.

Acknowledgements

The authors would like to thank Kevin Krauel for helpful comments and Dan Recoskie and Heming Wang for comments and for testing the software.

References

- [1] GNU Octave. <https://www.gnu.org/software/octave/>
- [2] Matlab. <https://www.mathworks.com/products/matlab.html>
- [3] D.Rife and J. Vanderkooy. MLSSA Software. <http://www.mlssa.com/>