WHAT ELSE SAYS "ACOUSTICAL CHARACTERIZATION SYSTEM" LIKE RON JEREMY?

Andrew Greenwood

Stanford University
Center for Computer Research in Music and Acoustics (CCRMA)
Aeg165@ccrma.stanford.edu

ABSTRACT

An interface was created for the generation and basic analysis of acoustical test signals. The interface was created using Matlab's GUIDE. The available test signals are linear and logarithmic sine sweeps, Golay pulse trains and cascaded allpass smear. These signals can be used to generate impulse response, t60 decay, frequency response and frequency specific polar response. The system was designed to function with a minimum learning curve and maximum practical function.

1. INTRODUCTION

The purpose of this experiment was to generate a user friendly way of generating and processing acoustical characterization signals. These signals can be used to characterize audio processing equipment, reverberant spaces, acoustical insulation, microphones and loudspeakers.

Traditional acoustical measurements are carried out by playing a test signal containing all frequencies and recording the response. Therefore, there are three variables in a traditional acoustical measurement setup: the sound generation device (usually a speaker), the sound capture device (usually a microphone) and the space the sound travels through in between (almost always a room). By holding any two of these variables constant, the third can be experimentally measured.

This system generates and plays stimulus signals while concurrently recording the system response. The responses can be processed to give impulse response, frequency response, t60 decay and frequency specific polar response.

2. IMPLEMENTATION

2.1. Stimuli Synthesis

2.1.1. Dirac-Delta Impulses

The traditional Dirac-Delta impulse [1] (eq. 1) is not available as a fundamental stimulus type in this interface although it is used in generating some of the other stimulus signals. The unit impulses used to generate other stimulus types are easily constructed in Matlab using the code in equation 1.

```
h= [1, zeros(1, 1024)];        (1)
```

2.1.2. Cascaded All-Pass Impulse Smear

The cascaded all-pass filter [2] is calculated according to equation 2

$$\left(\frac{\rho + Z^{-1}}{1+ \rho Z^{-1}}\right)^{n} \qquad (2)$$

where n is the filter order and ρ is the all-pass filter coefficient. An impulse (eq. 1) is used as the filter input. This filter can be implemented in Matlab according to equation 3.

```
GnA = [1 p];
GnB = [p 1];
Gn = filter(GnB, GnA, h);              (3)
for i = 2:n;
    Gn = filter(GnB, GnA, Gn);
end;
```

An example all-passed impulse is given below in figure 1.

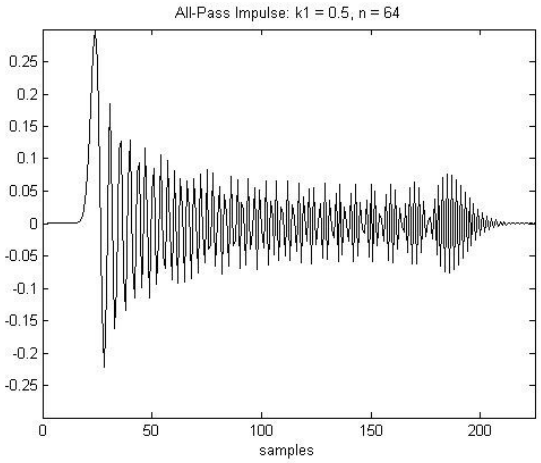

Figure 1. All-pass impulse k1 = 0.5, n = 64

The smeared all-pass signals generated in this program are volume maximized to give the highest RMS amplitude. This involves finding a ρ corresponding to the minimum of the maximum impulse response tap level [2]. This is achieved via equation 4.

```
minGain = max(abs(Gn));
minP = 0.001;

for p = 0.001: 0.001:0.999
    GnA = [1 p];
    GnB = [p 1];
    h = [1, zeros(1, 1024)];
    Gn = filter(GnB, GnA, h);
                                                    (4)
    for i = 2:n;
        Gn = filter(GnB, GnA, Gn);
    end;
    Gain = max(abs(Gn));
    if(Gain < minGain)
        minGain = Gain;
        minP = p;
    end;

end;
```

2.1.3. Swept Sine Measurements

Frequency swept sine waves achieve greater volume maximization using a fundamentally different set of equations [4]. Both logarithmic (eq. 5, fig. 2) and linear (eq. 6, fig. 3) sine sweeps are available to generate impulse response measurements.

```
f=exp(log(20*(2/fs))*(1-[0:abins-
1]'/abins));
```

```
phi = cumsum(pi*f);                                 (5)
X = sin(phi);
```

```
f = [0:abins-1]'/(abins);
phi = cumsum(pi*f);                                 (6)
X = sin(phi);
```

Where *fs* is the sampling rate and *abins* is the number of samples or the duration of the sweep.

Sine sweeps increase SNR by around 3dB per doubled sweep length. This program allows for variable sweep length to accommodate for specific experimental conditions.

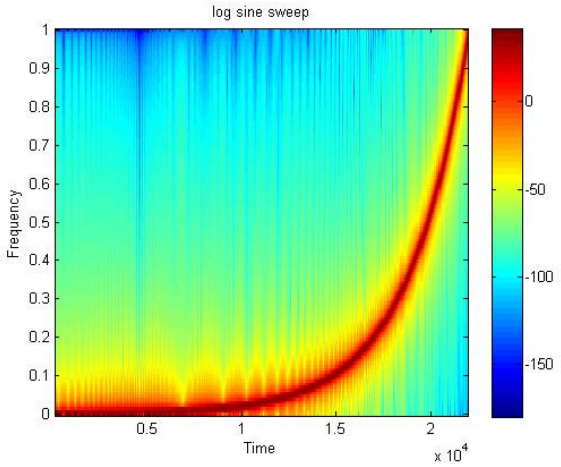

Figure 2. Logarithmic sine sweep spectrogram

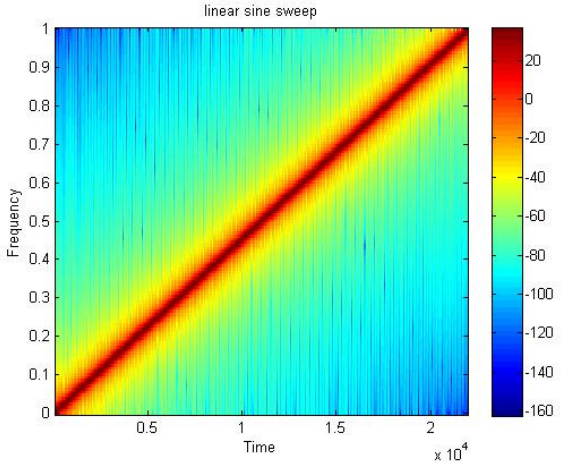

Figure 3. Linear sine sweep spectrogram

Swept sines are sometimes used in consecutive pairs which allows for the energy put into the room during the first sweep to decay during the second excitation. This option is available using the “Number of Sweeps” text box on the

program interface.

2.1.4. Golay Codes

Golay codes [5] are also available and are generated according to the Matlab code given in equation 7 [6].

```
a = [1 1];
b = [1 -1];
while (N>1)
    olda = a;
    oldb = b;                                   (7)
    a = [olda oldb];
    b = [olda -oldb];
    N = N - 1;
End;
```

The program displays the last N samples of the order N Golay code (figure 4).

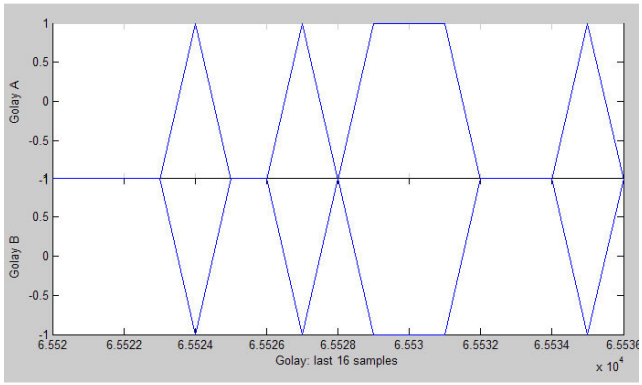

Figure 4. Last 16 samples of order 16 Golay codes

The Golay order is user specified on the program interface. Additionally, the room relaxation time or pause time in between excitation signals is also specified by the user.

2.2. Signal Processing

The processing routines are very similar across different test signals. The basic idea is that the original signal is cross-correlated with the measured response signal (eq. 8).

$$IFFT\left(\frac{FFT(response)}{FFT(input)}\right) = input \star \boldsymbol{response} \qquad (8)$$

This processing routine is experimentally accomplished for swept sine and cascaded all-pass signals according to either equation 9 or 10.

```
IR =
 real(ifft(fft(response)./fft(input)));                (9)
```

```
IR = fliplr(fftfilt(response, input));
                                                      (10)
```

Once the impulse response is calculated, the average energy is then calculated in order to give a smoother representation of the signal energy over time. This is achieved by convolving the impulse response squared by a weighted 10 ms window to give an average over the window length. The square root of the convolved signal is then taken to preserve relative magnitude (eq. 11).

```
window_length = 10*fs/1000;
window = ones(window_length, 1) /
 window_length;                                       (11)
avg_energy = sqrt(fftfilt(window,
 IR.^2));
```

The two signals are then plotted together (fig. 5)

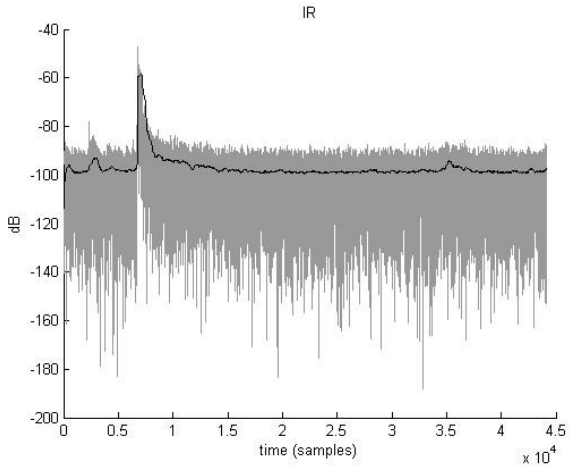

Figure 5. Example impulse response in dB

When correlating responses to Golay codes, a slightly different processing routine is required. Golay codes require that the signals be convolved, added and normalized by the energy put into the system. This is achieved according to equation 12.

```
IR =
 flipud((fftfilt(flipud(inputA),                      (12)
 responseA)) + (fftfilt(flipud(inputB),
 responseB)))/(2*length(inputA));
```

2.2.1. T60 Determination

In order to process t60 information, the decay region must be selected by the user. This is accomplished using the *ginput* function in Matlab. Once an impulse response is calculated, a plot similar to figure 5 is displayed. The user is asked to select two points on the x-axis so that the plot can be redrawn. Once presented with the new plot, the user is asked to select two points on the average energy function to use as bounds for t60 estimation. The selected region is then fit with a straight line and the t60 is calculated using the slope, m, from the line fit according to equation 13.

```
X = slope_vector;
Y = db(abs(avg_energy(slope_vector)));
 m = polyfit(X', Y, 1);                    (13)
t60 = (1/(m(1)*fs))*-60;
```

The *slope_vector* is the vector of samples between the range of t60 calculation selected by the user.

Once calculated, the t60 value is displayed on the interface and the plot along with the selected t60 calculation region and fit line are plotted (fig 6).

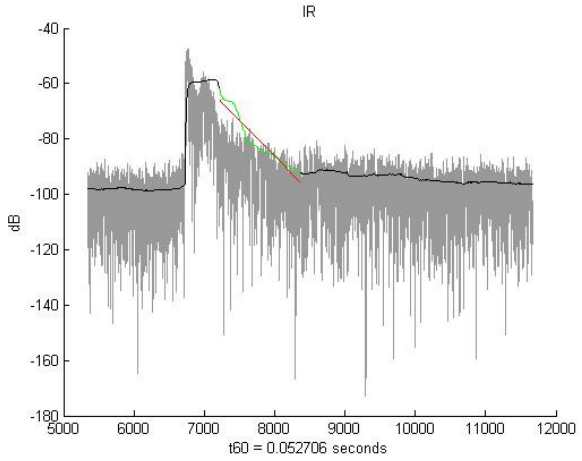

Figure 6: T60 with averaged data and fit line

2.2.2. Frequency Response Determination

The frequency response is generated by taking the FFT of the calculated impulse response (eq. 14).

```
freq_bins = 4028;
x = fs/2 * linspace(0, 1, freq_bins);
H = db(real(fft(IR, 2*freq_bins)));
                                          (14)
```

The rolling average of the frequency response is calculated similarly to equation 13.

Both the raw and averaged frequency responses are then displayed on the interface (fig. 7).

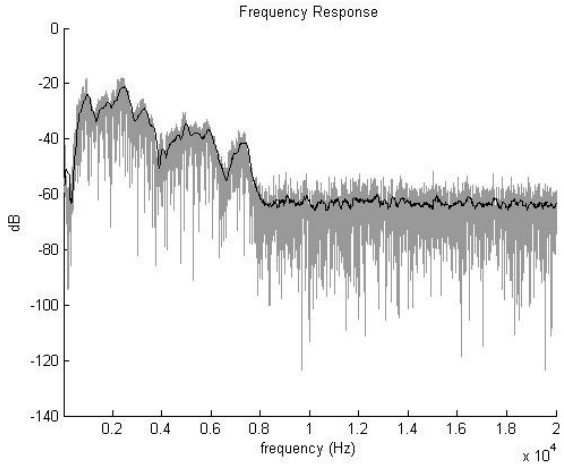

Figure 7: Example frequency response

2.2.3. Polar Response

The polar response is generated by calculating energy per octave frequency band per angle. The experimental variables of ending degree, degree increment, mirroring, number of sweeps and stereo are given by the user via the *Polar Response* panel (fig. 8).

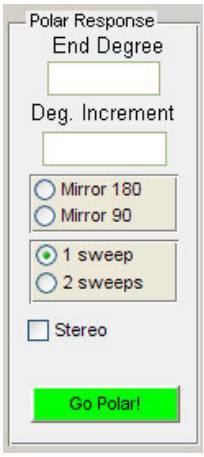

Figure 8: Polar response user interface

Swept sine tones are used as the stimulus signal. The impulse response is calculated using equation 10 and the signal is then split into four octave wide bands at 500 Hz,

1000 Hz, 2000 Hz and 4000 Hz using second order Butterworth filters according to equation 15.

```
f0 = 250;
for i = 1:4

    f = f0*2^i;
    f1 = (f*2^(1/2))/(fs/2);
    if f1 > 1
        f1 = 0.99999;                    (15)
    end;
    f2 = (f*2^(-1/2))/(fs/2);
    [B A] = butter(2, [f1 f2]);
    [h w] = freqz(B, A, F, fs);
    IR_bp(:,i) = filter(fliplr(B),
        fliplr(A), IR);

end;
```

The energy per band is then calculated according to equation 16 .

```
theta_500 = IR_bp(:,1);
theta_1000 = IR_bp(:,2);
theta_2000 = IR_bp(:,3);
theta_4000 = IR_bp(:,4);
                                         (16)
mag_500 = sqrt(sum(theta_500.^2,1);
mag_1000 = sqrt(sum(theta_1000.^2,1));
mag_2000 = sqrt(sum(theta_2000.^2,1));
mag_4000 = sqrt(sum(theta_4000.^2,1));
```

The calculated energy at each frequency band is collected into a vector containing radius values for each angle. An angle vector is concurrently built for plotting purposes. Once measurements have been taken at all of the specified angles, the system checks to see if mirroring has been enabled. If so, the signal is mirrored about 90 or 180 degrees depending on the ending angle parameter according to equations 17 and 18 respectively.

```
mag_500_vector = [mag_500_vector
fliplr(mag_500_vector)];
    mag_1000_vector = [mag_1000_vector
fliplr(mag_1000_vector)];
    mag_2000_vector = [mag_2000_vector
fliplr(mag_2000_vector)];
    mag_4000_vector = [mag_4000_vector
fliplr(mag_4000_vector)];
    t_rad_theRest = t_rad +
index_end*interval*pi/180;
    t_rad = [t_rad t_rad_theRest];
                                         (17)

mag_500_vector = [mag_500_vector
fliplr(mag_500_vector)
        fliplr([mag_500_vector
fliplr(mag_500_vector)])];
    mag_1000_vector = [mag_1000_vector
fliplr(mag_1000_vector)
        fliplr([mag_1000_vector
fliplr(mag_1000_vector)])];
    mag_2000_vector = [mag_2000_vector
fliplr(mag_2000_vector)
        fliplr([mag_2000_vector
fliplr(mag_2000_vector)])];
    mag_4000_vector = [mag_4000_vector
fliplr(mag_4000_vector)
         fliplr([mag_4000_vector
fliplr(mag_4000_vector)])];
    t_rad_theRest1 = t_rad +
index_end*interval*pi/180;
    t_rad_theRest2 = t_rad +
2*index_end*interval*pi/180;
    t_rad_theRest3 = t_rad +
3*index_end*interval*pi/180;
    t_rad = [t_rad t_rad_theRest1
t_rad_theRest2 t_rad_theRest3];
                                         (18)
```

The polar plot is then displayed to the user (fig. 9)

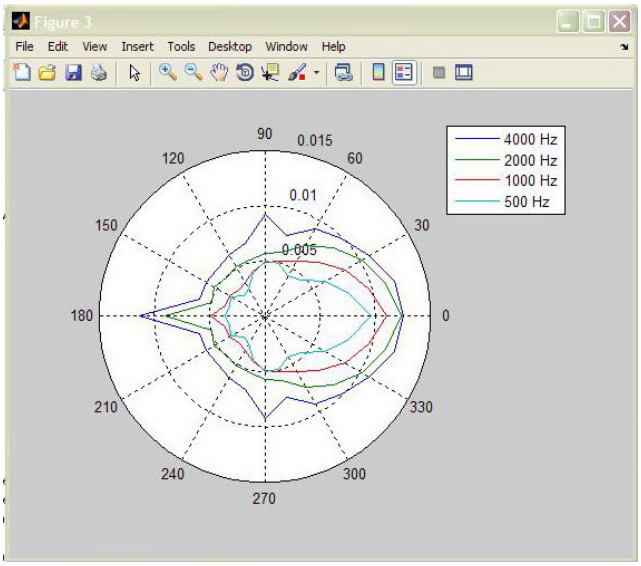

Figure 9: Polar plot of energy per octave band

2.3. Hardware Configuration

While any of these measurement signals are played response is simultaneously recorded. Duplexing is achieved through the use of MEX files (Windows only) given in the pa-wavplay bundle [7].

This package supports hardware functioning as DirectX and ASIO. The hardware environment is profiled and configured by the user in the *Hardware Setup* panel (fig. 10).

Figure 10: Hardware configuration panel

Additionally, a separate scroll window is displayed on the main interface to allow the user to set a specific sample rate. A screenshot of the main interface is given below in figure 11.

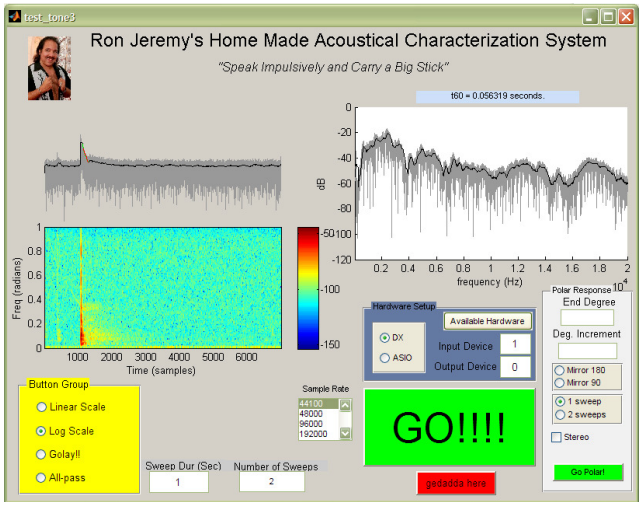

Figure 11: Current main interface design

3. DISCUSSION

An "auto-adjustment" feature could easily be included to automatically set the length of the swept sine wave signals. If a standard signal to noise ratio is set, the program could take a short sample of background noise and calculate a length assuming 3 dB of SNR improvement every time the sweep length is doubled. This should work well as long as the length doesn't get too long.

One major shortfall of the polar response system is that the process of orienting the microphone is currently left up to the user. This process could be automated by including support for an external stepper motor hooked up to a turntable. This would allow the measurement process to be automated.

If it is necessary, a profile of the speaker and microphone could be loaded into the program to allow for automatic compensation of the experimental measurement equipment.

It may also be useful to include an option to display the graphs in a standard "figure" window to allow the user to perform basic graphical transformation functions such as zooming and tagging.

Ideally, once everything is working correctly, it would be nice to translate these functions to DX, AU or VST plug-ins to allow for signal generation and acquisition within a standard DAW. I imagine that this would make many of the standard operations much simpler because the DAWs are more specifically targeted to audio applications than Matlab.

4. REFERENCES

[1] Wikipedia contributors, 'Dirac delta function', *Wikipedia, The Free Encyclopedia,* 5 May 2009, http://en.wikipedia.org/w/index.php?title=Dirac_delta_function&oldid=288017888

[2] Smith, J. O. "Nested Allpass Filters," *Physical Audio Signal Processing,* December 2008, http://www.dsprelated.com/dspbooks/pasp/Nested_Allpass_Filters.html

[3] Griesinger, D. "Impulse Response Measurements Using All-Pass Deconvolution," 1992. Retrieved from: http://www.davidgriesinger.com/dgaes92b.pdf

[4] Farina, A. "Simultaneous Measurement of Impulse Response and Distortion with a Swept-Sine Technique," *Audio Engineering Society Convention, Preprint 5093*, 2000

[5] Foster, S. "Impulse response measurement using Golay codes," *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.* , 1986, vol.11, pp. 929-932

[6] Abel, J. and Berners, D. (2005). *Signal Processing Techniques for Digital Audio Effects*, 2005. Retrieved from: http://ccrma.stanford.edu/courses/424/

[7] Frear, M. pa-wavplay, *SourceForge,* Sept. 2004. http://sourceforge.net/projects/pa-wavplay/