LINEAR SYSTEM MODELING

1. DIGITAL SYSTEMS

Both in the discrete and continuous domain, the effects of linear systems can be observed and ultimately emulated when introduced to appropriate reference signals. When measuring the system responses of hardware, a single-sample signal (called an “impulse”) is synthesized to trigger the effect of some dynamic system. Similarly, for acoustic environments, a complimentary signal (called a “sine sweep” or “chirp”) is synthesized to play back an array of frequencies within the general range of hearing (20Hz to 20kHz) or until the Nyquist frequency of a given discrete-system sampling rate (e.g., 24kHz for a 48kHz sampling rate). For this project, direct impulse responses and sine sweeps were used to analyze both audio software plugins, audio hardware, and acoustic spaces. The analysis of these spaces also includes convolution in the frequency-domain using the discrete Fourier transform (DFT) and deconvolution to recover impulse responses from the recorded sine sweeps.

For consistency, a sampling rate of 48kHz was maintained for all signal synthesis and track printing in ProTools. For the direct impulse response (IR) method, a method utilizing a single-sample IR, MATLAB code was generated to pad an impulse with zeros so that, when concatenated with the impulse, the entire impulse signal would be equal to one second. Note that one second of audio is 48,000 samples and the impulse sample is represented by a value of one (e.g., [1 0 0 0 … 0_{N-1}]). Because the IR signal never exceeds a linear amplitude of one, the digital system being measured was able to process the signal without clipping, verifying a transparent maximum output without distortion. The audio software plugins measured were EQ III and Mod Delay III, both stock ProTools plugins. For the filter analysis, EQ III’s parameters were altered to

include a high-pass filter (300Hz, 12dB/oct), a “high-mid frequency” bell curve (2kHz, -5dB), and a “high frequency” shelf (12kHz, +3dB). The input and output gain were not changed and the phase was maintained relative to the output of the equalizer. To collect the IR of the filter, the plugin was inserted onto a new mono audio track in the ProTools edit bay without any other processing or gain manipulation. The track with the filter and impulse signal was printed and saved.

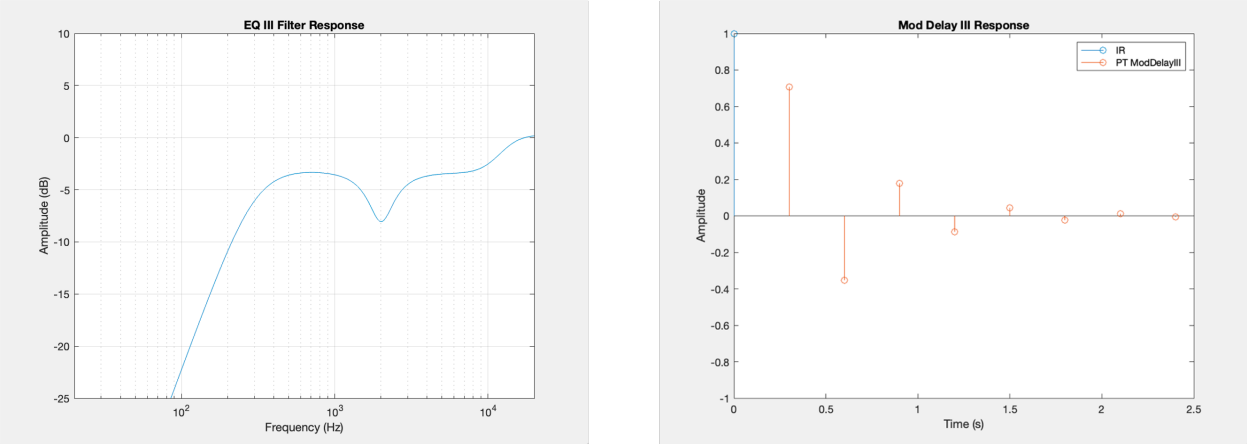

Figure 1. Plotted responses for both digital systems (see Appendix B)

The delay plugin was then inserted onto its own separate audio track with the unprocessed impulse response signal with the following parameters: feedback of -50%, mono delay output of 300ms, and the wet/dry set to 100%. The same print and save methods were used. Because the processed IR signal acted as a control for each system, a dry vocal track was passed through the filter and delay plugins separately. The same processes were used for the impulse collection. The processed vocals were used for null testing to assess the accuracy of each system’s impulse response. It should be noted that all plugin parameters were maintained between the software IR collection and the processed vocal printing. To output the responses of the filter, MATLAB’s built-in *freqz()* function processed the impulse response of a given digital

filter, *h*. For the delay, a stem plot of the impulse response reacting with the IR showed the initial and subsequent sample delays with linear amplitude as a function of time (seconds) (Figure 1).

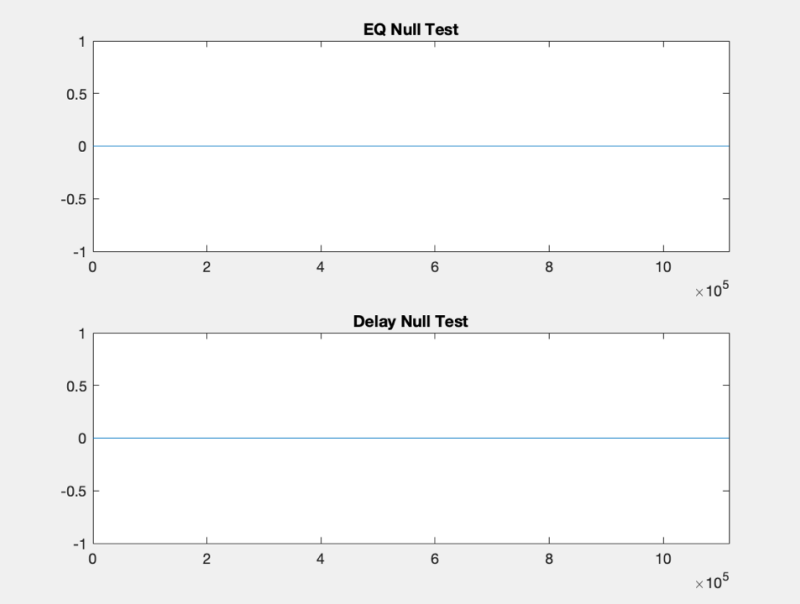

Figure 2. Digital plug-ins null test results

To test the accuracy of the new digital impulse responses, we first needed to convolve the individual filter and delay IRs with the unprocessed vocal signal. Using MATLAB's *conv()* function to convolve the individual impulse responses with their own dry vocal, we were able to use that output in a null test against the ProTools vocal prints (Figure 2). All following null tests followed the same processes even if deconvolution was needed prior to the null difference. Because the output arrays after the null test were all zeros, we assumed complete cancellation occurred, the single-sample IR was an accurate representation of those systems, and no level of memoryless randomization was introduced during plugin processing.

2. ACOUSTIC AND HARDWARE SYSTEMS

As noted in the project guidelines, the direct IR method is not an ideal approach for measuring acoustic and hardware spaces; rather, the sine sweep method is significantly more effective given its ability to record a system's response to a set frequency range over time. Through the process of deconvolution, the impulse response of that system can be recovered for use similar to that of digital system measurement. For our measurements, a 10 second logarithmic sine sweep was generated using MATLAB's built-in *chirp()* function. While linear sweeps are also an option, its logarithmic counterpart tends to sweep through the frequencies more evenly on this scale, resulting in a more accurate response of the system. A linear fade of 10,000 samples was added to the beginning and end of the sweep to ensure the bookends of the signal did not abruptly start or cut off.

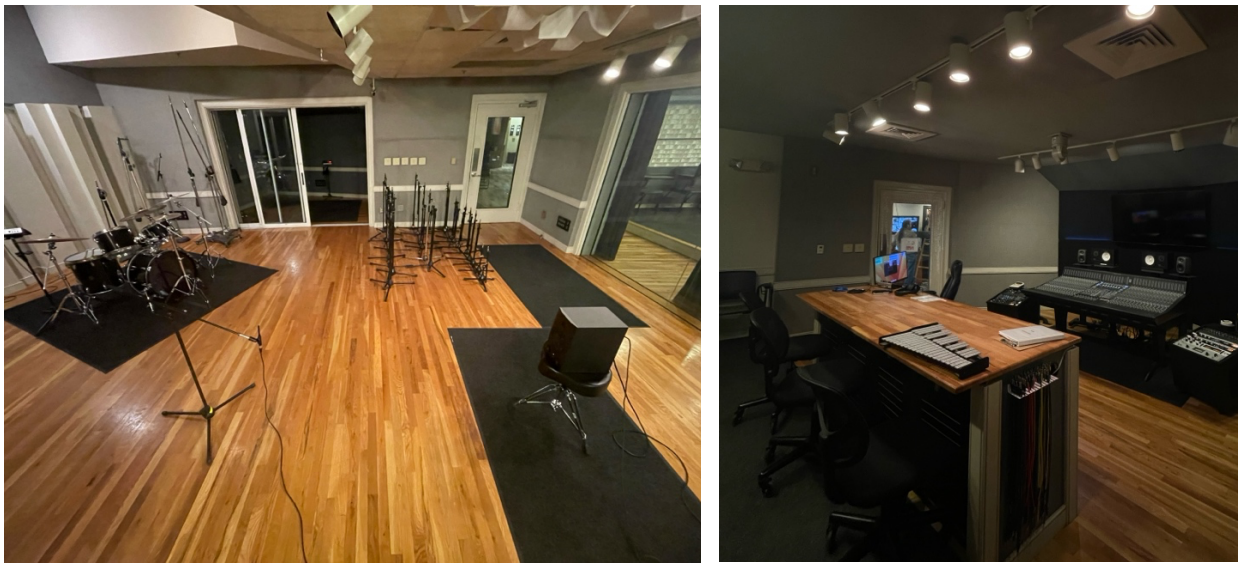

Figure 3. REM Studio A – tracking room

For the acoustic measurement, we used the live room of Belmont University's REM Studio A (Figure 3). To acquire the room response, an Adam Audio A7X monitor was placed on a padded drum throne acting as our source. An omnidirectional DBX RTA-M was used to record

the signals with the capsule pointing towards the ceiling, perfectly flush with the monitor's tweeter. The microphone was placed six feet from the speaker, and the distance never changed during the measurement process. Multiple versions of the sine sweep and unprocessed vocal were captured for quality control.

Similarly, audio hardware takes to logarithmic sine sweeps more effectively than the direct IR method used in the digital measurements. We chose to measure a Maag EQ-4 for its 650Hz bell curve (-4dB) and its famous "air band" high-shelf at 20kHz (+6dB). The same 10 second logarithmic sine sweep was used to measure the EQ-4 and, similarly, two dry vocal tracks were passed through each EQ setting individually and printed. This process was possible by patching the output of our dry vocal into the EQ and routing the output of that hardware into the input of a blank record-enabled track (Figure 4). This process was used for all sine sweeps and vocal prints.

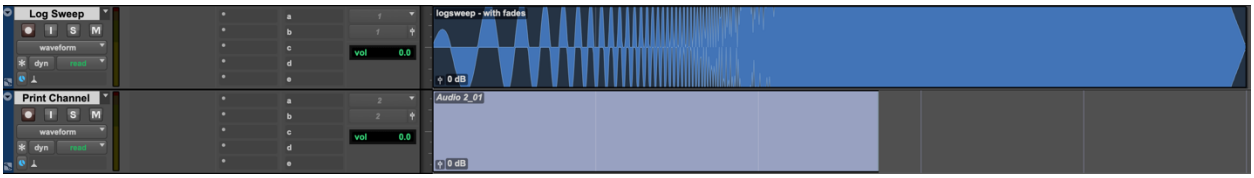

Figure 4. ProTools routing example for hardware measurement

In order to do null tests on our acoustic and hardware systems, we first had to deconvolve the sweep responses from each system. The method used was deconvolution with Tikhonov regularization (Figure 5), a constant that both low-passes an output signal and keeps the deconvolution function from becoming undefined. This process, while difficult in the time domain, is much more easily computed in the frequency domain, allowing us to use Fourier transform functions within MATLAB. The dry sine sweep signal *x[n]* and sine sweep recorded through our system *y[n]* were converted into the frequency domain via MATLAB's built-in *fft()*

function. Then, we iterated over each frequency bin, *k[n]*, computing the equation seen in Figure 5. Note that G_k is *x[n]* in the frequency domain, G^*_k represents the conjugate (MATLAB's *conj()* function) of G_k, D_k represents *y[n]* in the frequency domain, and Δt is the sampling period related to the sampling rate of our input signals.

$$M_k = \frac{G_k^* D_k}{(G_k^* G_k + \lambda)\Delta t}$$

Figure 5. Deconvolution equation with Tikhonov regularization

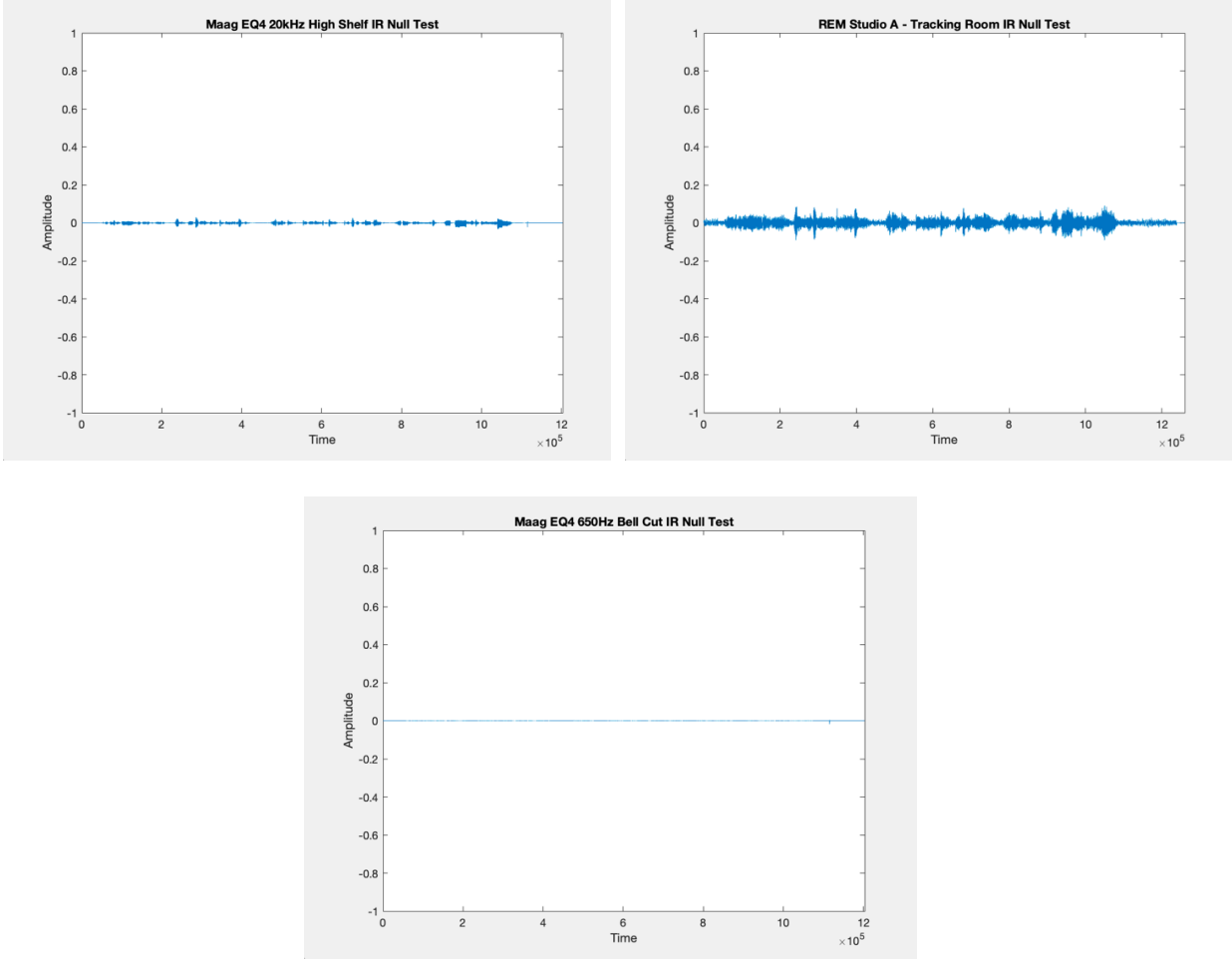

Figure 6. Null tests of acoustic and hardware systems

As mentioned before, the presence of the Tikhonov regularization constant (λ) allows for us to handle noisy data more effectively. Aster and Borchers note that noise in the frequency domain can alter the stability of spectral division and that the Tikhonov constant evens out these tendencies [1]. The value also tends to act as a low-pass filter given that noise tends to exist more at higher frequencies. After computing the deconvolution on our dry sweep and recorded sweep, we are left with an impulse response that can be convolved with our dry vocal signal to then be null tested against the vocal that was recorded through our system. The same process was done for the EQ-4 hardware for null testing.

As seen in Figure 6, the acoustic null test presented decent but not total cancellation. The resulting signal (after subtracting the MATLAB convolved signal with the vocal signal played through the system) had some remanence of the dry vocal but was mostly high frequency noise with the max amplitude of the null signal peaking at 0.0908 on the linear scale. REM Studio A is known for having notable HVAC noise during recordings and, when removing the Tikhonov constant from our computation ($\lambda = 0$), the impulse response on a unipolar scale is nearly unreadable. This was proof that our deconvolution method used was effective in removing a sizable amount of noise, resulting in a suitable impulse response for the space as a reverberation effect. The null tests for the Maag EQ-4 were significantly more successful with the bell curve subtractive EQ outputting a near-zero array after null testing. The 20kHz high shelf only had a maximum output value of 0.029 on the linear scale (or approximately -30.754 dB). Because the null test was not completely successful, we believed the incomplete null output was due to non-linearity in the phase response of the filter.

Because a room response is both an acoustic measurement and a reverberation effect, the RT60 can be adopted to analyze our deconvolved impulse response. Defined as the time after the

impulse that it takes for a signal *h[n]* to reach -60dB (Figure 7), this metric can help define the length of a reverberate effect. To see this in MATLAB, the IR signal was plotted in decibels as a function of time. The RT60 of this space was approximately 0.2644s.

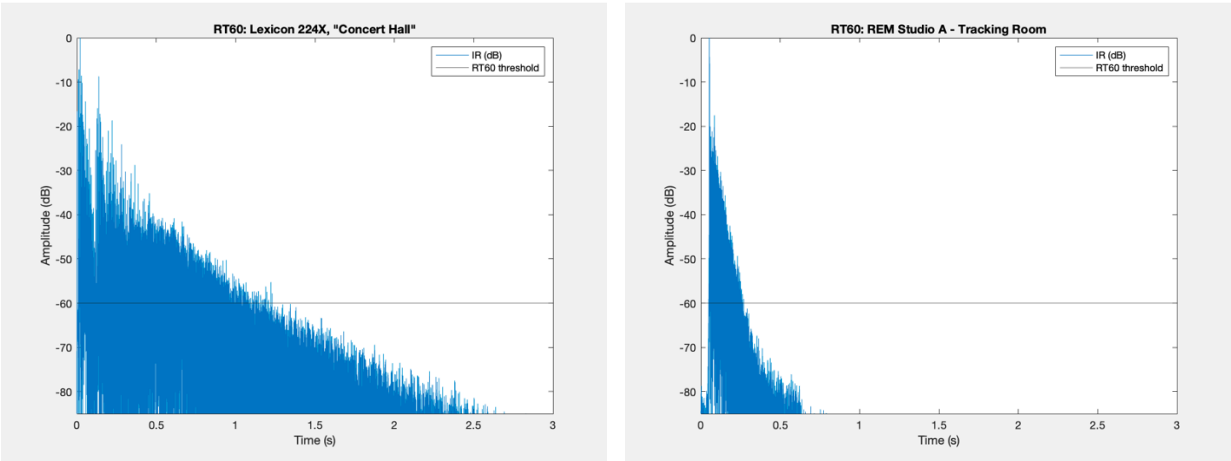

Figure 7. RT60 comparison of digital and acoustic systems

For comparison, a Lexicon 224X Digital Reverb was measured on a “Concert Hall” preset, resulting in an RT60 of approximately 1.258s. The comparison between the two RT60 graphs shows the visual cues of a lively space (hall) versus a drier space (room).

3. CONCLUSION

After synthesizing test signals and collecting impulse responses from software, hardware, and acoustic spaces, we discovered that, with the absence of randomization in a software’s algorithm, digital systems take well to single-impulse (i.e., direct IR) analysis and are the most likely to pass null testing with near-zero output. Similarly, when dealing with non-digital systems that have the potential of interacting with noise, the deconvolution (with regularization) of a sine sweep is a reliable enough process to remove unwanted interference when collecting general impulse responses.

Bibliography

[1] R. Aster and B. Borchers, “Time Series/Data Processing and Analysis (MATH 587/GEOP 505).” Oct. 10, 2008.

Appendix A: Code Snippets

The following are all notable code scripts from this project's MATLAB library. All scripts will be included with the project along with all pertinent audio files used and synthesized.

impulseSignal.m

Zachary Miller, Alek Weidman, John Sweeney

```
clear; clc; close all;
% Synthesize impulse (~1sec)
Fs = 48000;
impulse = [1;zeros(Fs-1,1)];
audiowrite('ImpulseResponse_48k.wav',impulse,Fs);
```

Published with MATLAB® R2023b

sineSweep.m

Zachary Miller, Alek Weidman, John Sweeney

```
% 21 March 2024
clear; clc; close all;
% Linear
Fs = 48000;
Ts = 1/Fs;
t = [0:Ts:10];
xLin = chirp(t,0,10,22000).';

% Logarithmic
xLog = chirp(t,1,10,22000,'logarithmic').';
N = length(xLog);

% Apply fade-in/fade-out
fade = [linspace(0,1,10000).';ones(N-20000,1);linspace(1,0,10000).'];
y = xLog .* fade;

audiowrite('logsweep - with fades.wav',y,Fs);
```

Published with MATLAB® R2023b

irPlot.m

Zachary Miller, Alek Weidman, John Sweene

```
% 21 March 2024
clear; clc; close all;

[IR,Fs] = audioread('ImpulseResponse_48k.wav');
[hEQ] = audioread('IR_EQ.wav');
[hDelay] = audioread('IR_ModDelay.wav');

% Plot parameters
Ts = 1/Fs;
N = length(hDelay);
t = [0:Ts:(N/Fs)-Ts].'; % time domain in seconds
IR = [IR;zeros(N-length(IR),1)]; % pad
IR(IR == 0) = NaN; hDelay(hDelay == 0) = NaN; % ignore zeros

% Plot log frequency, dB amplitude
[H_EQ,W_EQ] = freqz(hEQ,1,2048,Fs);

% Amplitude plots
figure;
semilogx(W_EQ,20*log10(abs(H_EQ))); % EQ response
axis([20 20000 -25 10]); grid on;
xlabel('Frequency (Hz)'); ylabel('Amplitude (dB)');
title('EQ III Filter Response');

figure; grid off;
stem(t,IR); hold on; % IR reference
stem(t,hDelay); hold off; % Delay response
axis([0 2.5 -1 1]); legend('IR','PT ModDelayIII');
xlabel('Time (s)'); ylabel('Amplitude');
title('Mod Delay III Response');
```

Published with MATLAB® R2023b

impulseDeconvolutionAcoustic.m

Zachary Miller, Alek Weidman, John Sweeney

```
% 30 March 2024
% Deconvolution, RT60, and null test
clear; clc; close all;

[x,Fs] = audioread('logsweep - with fades.wav'); % dry sine sweep
y = audioread('REM Sweep.wav'); % sine sweep in room
y = y/max(y); % normalize

% Pad and collapse "y" to mono
pad = length(y)-length(x);
x = [x;zeros(pad,1)];
y = y(:,1);

X = fft(x); Y = fft(y);
lenX = length(X);

% Deconvolution
reg = 0.99; Nyq = Fs/2; Ts = 1/Fs;

H = zeros(lenX,1);
for k=1:lenX

    Xk = X(k,1); Yk = Y(k,1); conjXk = conj(Xk);
    H(k,1) = (conjXk * Yk) / ((conjXk * Xk + reg) * Ts);

end

[jake] = audioread('REM Vocal.wav');
jake = jake(:,1);

h = real(ifft(H));
h = h/max(h);

% RT60
N = length(H); Ts = 1/Fs;
t = [0:Ts:(N/Fs)-Ts].'; % time domain (seconds)
plot(t,20*log10(abs(h))); hold on;
yline(-60); hold off; % RT60 threshold
axis([0 3 -85 0]); legend('IR (dB)', 'RT60 threshold');
xlabel('Time (s)'); ylabel('Amplitude (dB)');
title('RT60: REM Studio A - Tracking Room');

% Null
vocal = audioread('Vocal.wav');
out = conv(vocal,h);
out = out(1:length(jake),1); % cut signal length
out = out/(max(out)/max(jake)); % extra normalization
null = jake - out;

figure; plot(null); axis([0 length(jake) -1 1]);
xlabel('Time'); ylabel('Amplitude');
title('REM Studio A - Tracking Room IR Null Test');
```

Published with MATLAB® R2023b

impulseDeconvolutionHardwareReverb.m

Zachary Miller, Alek Weidman, John Sweeney

```
% 2 April 2024
% Deconvolution, RT60, and null test
clear; clc; close all;

[x,Fs] = audioread('logsweep - with fades.wav');
y = audioread('Lexicon 224X Concert Hall Sin Sweep.wav');
y = y/max(y); % normalize, max(y) = max(x)

% Pad and collapse "y" to mono
pad = length(y)-length(x);
x = [x;zeros(pad,1)];
y = y(:,1);

X = fft(x); Y = fft(y);
lenX = length(X);

% Deconvolution
reg = 0.99; Nyq = Fs/2; Ts = 1/Fs;

H = zeros(lenX,1);
for k=1:lenX

    Xk = X(k,1); Yk = Y(k,1); conjXk = conj(Xk);
    H(k,1) = (conjXk * Yk) / ((conjXk * Xk + reg) * Ts);

end

[jake] = audioread('Lexicon 224X Concert Hall Vocal.wav');
jake = jake(:,1);

h = real(ifft(H));
h = h/max(h); % normalize

% RT60
N = length(H); Ts = 1/Fs;
t = [0:Ts:(N/Fs)-Ts].'; % time domain (seconds)
plot(t,20*log10(abs(h))); hold on;
yline(-60); hold off; % RT60 threshold
axis([0 3 -85 0]);
legend('IR (dB)', 'RT60 threshold');
xlabel('Time (s)'); ylabel('Amplitude (dB)');
title('RT60: Lexicon 224X, "Concert Hall"');

% Null
vocal = audioread('Vocal.wav');
out = conv(vocal,h);
out = out(1:length(jake),1); % cut signal length
out = out/(max(out)/max(jake)); % extra normalization
null = jake - out;

figure;
plot(null); axis([0 length(jake) -1 1]);
```

Published with MATLAB® R2023b

impulseDeconvolutionMaagBell.m

Zachary Miller, Alek Weidman, John Sweeney

```
% Deconvolution and null test
% 2 April 2024

clear; clc; close all;

[x,Fs] = audioread('logsweep - with fades.wav'); % dry sine sweep
y = audioread('Maag EQ4 (650Hz Bell -4dB) Sin Sweep.wav'); % sine sweep in room
y = y/max(y); % normalize, max(x) = max(y)

% Pad and collapse "y" to mono
pad = length(y)-length(x);
x = [x;zeros(pad,1)];
y = y(:,1);

X = fft(x); Y = fft(y);
lenX = length(X);

% Deconvolution
reg = 0.99; Nyq = Fs/2; Ts = 1/Fs;

H = zeros(lenX,1);
for k=1:lenX

    Xk = X(k,1); Yk = Y(k,1); conjXk = conj(Xk);
    H(k,1) = (conjXk * Yk) / ((conjXk * Xk + reg) * Ts);

end

[PTVocal_MaagBell] = audioread('Maag EQ4 (650Hz Bell -4dB) Vocal.wav');

h = real(ifft(H));
h = h/max(h); % normalize

% Null
vocal = audioread('Vocal.wav');
out = conv(vocal,h);
out = out(1:length(PTVocal_MaagBell));
out = out/(max(out)/max(PTVocal_MaagBell));
null = PTVocal_MaagBell - out;

plot(null); axis([0 length(out) -1 1]);
xlabel('Time'); ylabel('Amplitude');
title('Maag EQ4 650Hz Bell Cut IR Null Test');
```

Published with MATLAB® R2023b

impulseDeconvolutionMaagShelf.m

Zachary Miller, Alek Weidman, John Sweeney

```
% 2 April 2024
% Deconvolution and null test

clear; clc; close all;

[x,Fs] = audioread('logsweep - with fades.wav');
y = audioread('Maag EQ4 (20kHz High Shelf +6dB) Sin Sweep.wav');
y = y/max(y); % max(x) = max(y)

% Pad and collapse "y" to mono
pad = length(y)-length(x);
x = [x;zeros(pad,1)];
y = y(:,1);

X = fft(x); Y = fft(y);
lenX = length(X);

% Deconvolution
reg = 0.99; Nyq = Fs/2; Ts = 1/Fs;

H = zeros(lenX,1);
for k=1:lenX

    Xk = X(k,1);
    Yk = Y(k,1);
    conjXk = conj(Xk);
    H(k,1) = (conjXk * Yk) / ((conjXk * Xk + reg) * Ts);

end

[PTVocal_Maag20kHz] = audioread('Maag EQ4 (20kHz High Shelf +6dB) Vocal.wav');

h = real(ifft(H));
h = h/max(h); % normalize

% Null
vocal = audioread('Vocal.wav');
out = conv(vocal,h);
out = out(1:length(PTVocal_Maag20kHz));
out = out/(max(out)/max(PTVocal_Maag20kHz)); % extra normalization
null = PTVocal_Maag20kHz - out;

plot(null); axis([0 length(out) -1 1]);
xlabel('Time'); ylabel('Amplitude');
title('Maag EQ4 20kHz High Shelf IR Null Test');
```

Published with MATLAB® R2023b

digitalNull.m

Zachary Miller, Alek Weidman, John Sweeney

```
% 21 March 2024
% Null tests for ProTools EQ and delay

clear; clc; close all;

% Impulse responses
[vocal,Fs] = audioread('Vocal.wav');
[IR_EQ] = audioread('IR_EQ.wav');
[IR_ModDelay] = audioread('IR_ModDelay.wav');

% ProTools prints
[PTVocal_EQ] = audioread('PTVocal_EQ.wav');
[PTVocal_ModDelay] = audioread('PTVocal_ModDelay.wav');

% Manual convolution
vocal_EQ = conv(vocal,IR_EQ);
vocal_EQ = vocal_EQ(1:1115330); % cut short
vocal_ModDelay = conv(vocal,IR_ModDelay);
vocal_ModDelay = vocal_ModDelay(1:1115330); % cut short

N = length(vocal); % length of all files

nullEQ = vocal_EQ - PTVocal_EQ;
subplot(2,1,1); plot(nullEQ); title('EQ Null Test');
axis([0 N -1 1]); xlabel('Time'); ylabel('Amplitude');
nullDelay = PTVocal_ModDelay-vocal_ModDelay;
subplot(2,1,2); plot(nullDelay); title('Delay Null Test');
axis([0 N -1 1]); xlabel('Time'); ylabel('Amplitude');
```

Published with MATLAB® R2023b

Appendix B: Extra figures

Figure 8. ProTools EQ III preset

The response from this plugin can be seen in Figure 1.

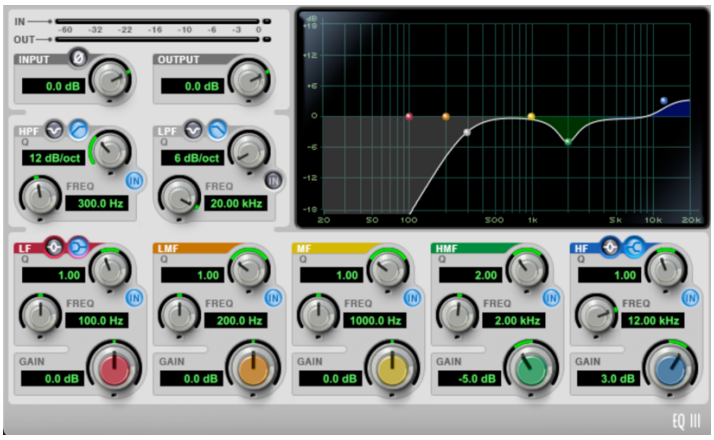

Figure 9. ProTools Mod Delay III preset

The response from this plugin can be seen in Figure 1.