Lab 02 - Compression Architectures

Ron Guglielmone

PROBLEM 1: GAIN COMPUTATION

Modify the basic compressor gain computer to produce the following static compression characteristics:

$$\ell_{\text{out}} = \begin{cases} \ell_{\text{in}} & \ell_{\text{in}} < \ell_{\text{T}}, \\ \ell_{\text{T}} + \frac{1}{\rho}(\ell_{\text{in}} - \ell_{\text{T}}) & \ell_{\text{in}} \geq \ell_{\text{T}}, \end{cases}$$

This can be accomplished by implementing the following dB gain function in C++:

$$g(\ell_I) = (\ell_I - \ell_T)(1/\rho - 1)$$

The following code modification accomplishes this (Figure 1).

```
// Above threshold:
if (threshold < level_estimate)
{
    float rho = comp_ratio;
    dbgainval = (log_level - dB(threshold)) * ((1 / rho) - 1);
}

// Below threshold:
else {
    dbgainval = 0.0; // becomes 1 after dB2lin
}
```

Figure 1: code for problem 1.

Three test signals are available as input (Figure 2).

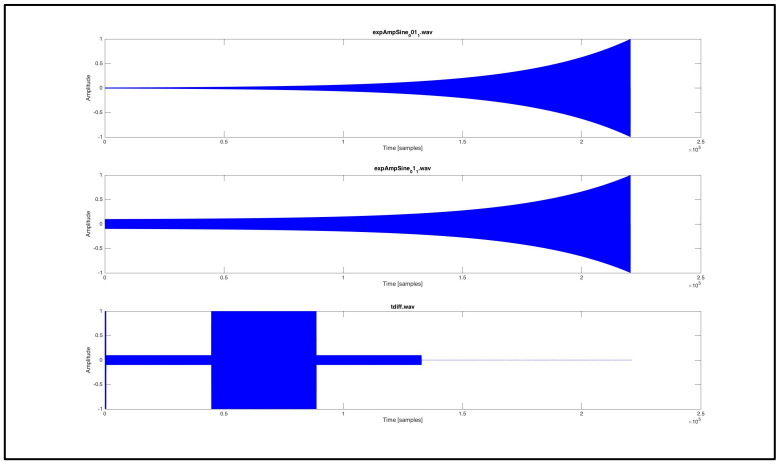


Figure 2: available test signals.

The second signal, "expAmpSine_01.wav", has been used with a compression ratio of 5, threshold of 0.1 (-20 dB), and the fastest attack time (0.09999 mS). The output after processing is plotted below in Figure 3.

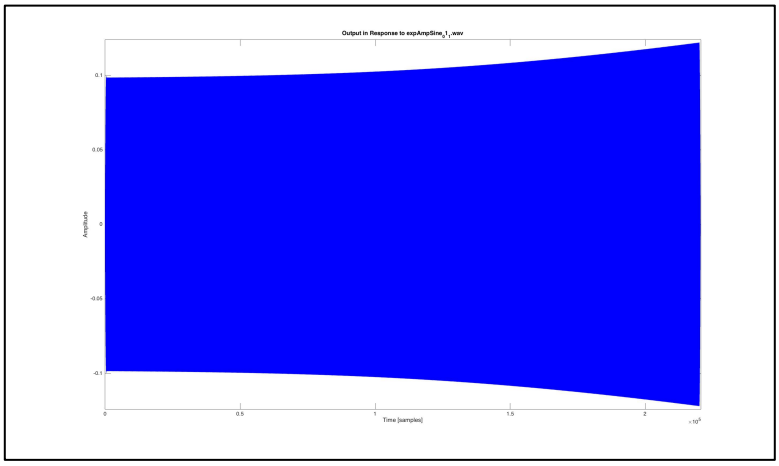


Figure 3: ratio of 5, threshold of 0.1, a.t. 0.09 mS, r.t. 100 mS

PROBLEM 2: ANALOG-STYLE GAIN COMPUTER

The following MATLAB script was used to model both a feed forward and feedback analog compressor (Figures 4,5).

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford University
% April 19, 2017
%
% HW 2 - Problem 2
%

% Feed Forward Compressor Model:

R0 = 100e3;
signalLevel = (1:0.01:100e3);
Rp = R0 .* ( signalLevel ) .^ (-0.75);
filterGain = 2 * Rp ./ ( R0 + Rp );
filterOutput = signalLevel .* filterGain;
plot(signalLevel , filterOutput);
xlabel('Input Signal Level');
ylabel('Output Signal Level');
title('Analog Compressor Models');


% Feedback Compressor Model:

hold on;
filterOutput2 = ((signalLevel + 1^(-0.75)...
            * signalLevel .^ (1.75)) ./ 2);
plot(filterOutput2,signalLevel);
legend('Feed Fwd', 'Feedback');
```

Figure 4, MATLAB script for models.

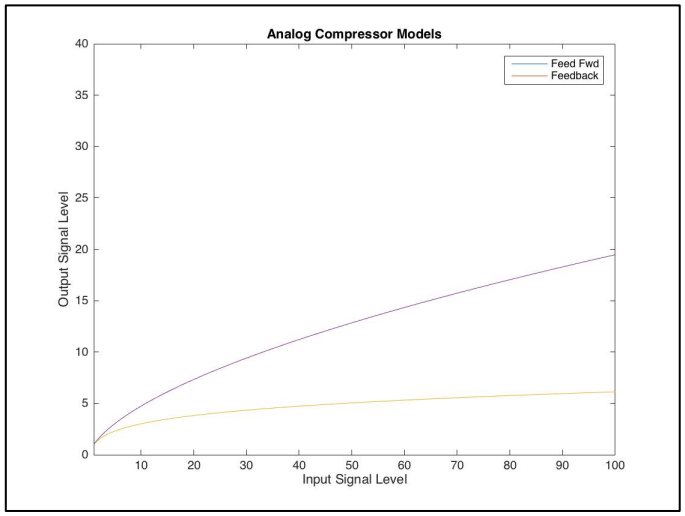


Figure 5, MATLAB output for models.

PROBLEM 3: LEVEL DETECTOR

3(a) The peak-detector has been modified to implement release-to-threshold (Figure 6).

```
// Process one block (one sample):
void process (float input, float threshold, float& output) {

    // Test (?) above or below threshold:
    if ( fabs( input ) > levelEstimate ) {

        // "Attack-state" update equation:
        levelEstimate += b0_a * ( fabs( input ) - levelEstimate );

    }

    else {

        // "Release-state" update equation(s):

        // Release to signal:
        // levelEstimate += b0_r * ( fabs( input ) - levelEstimate );

        // Release to zero:
        // levelEstimate += b0_r * (0 - levelEstimate );

        // Release to threshold:
        levelEstimate += b0_r * ( threshold - levelEstimate );

    }

    // Update output:
    output = levelEstimate;
}
```

Figure 6, release-to-threshold update equation.

3(b) Expressions for release-to-signal and release-to-threshold models have been derived and plotted in MATLAB (Figures 7,8).

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford University
% April 22, 2017
%
% HW 2 - Problem 3(b)

% Constants:
thresh = 0.1;                   % -20 dB
estimateThresh = [1];           % R.T.Thresh
estimateSig = [1];              % R.T.Signal
fs = 441e3;                     % 44.1kHz
dur = 5;                        % seconds
signal = zeros(dur*fs,1)';      % null
tau = 0.1;                      % release

% Update equation constant:
b0 = (1 - exp( -1.0 / ( tau * fs ) ));

% Main loop:
for n = 2 : length(signal)

    estimateThresh(n) = estimateThresh(n-1)...
        + b0*(thresh - estimateThresh(n-1));

    estimateSig(n) = estimateSig(n - 1)...
        + b0*(signal(n-1) - estimateSig(n-1));
end

% Invert estimate to find 'gain':
estimateThresh = 1 - estimateThresh;
estimateSig = 1 - estimateSig;
```

Figure 7, MATLAB model for different release patterns.

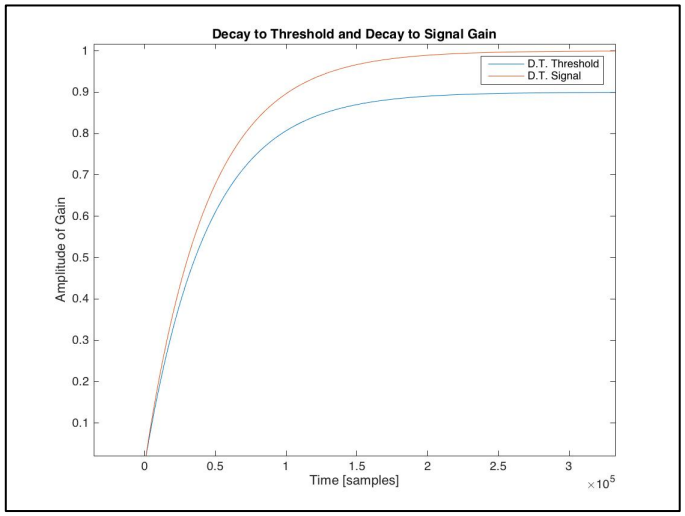


Figure 8, MATLAB output for gain vs. time.

3(c) Program dependent level detector:

To implement a program dependent detector, we essentially build two separate signal trackers: one running slowly, and another that is quicker. We use the quick one as our primary detector, but allow it to release towards the slower one. This has been implemented in the attached C++ code files.

PROBLEM 4: MYSTERY COMPRESSOR

4(a) A MATLAB script was written to generate test signals (Figure 9).

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford University
% April 22, 2017
%
% HW 2 - Problem 4

% Test Signal for Threshold

clear all
close all

% Globals:
fs = 44100;
dur = 10;
f = 1000;
dt = 1/fs;
t = (0:dt:dur-dt);

% Signal construction:
signal = cos(2*pi*f*(t));

l = length(t);
scaler = zeros(length(t),1)';

for i = 0 : 10
    for k = 1:fs
        scaler(i*fs + k) = (0.1)*i;
    end
end

scaler = scaler(44101:end);
signal = signal .* scaler;

% Save as .wav file:
audiowrite('Stepped_Sines.wav',signal,fs);
```

Figure 9, MATLAB script for first test signal.

The test signal waveform was then compared before and after processing (Figure 10).

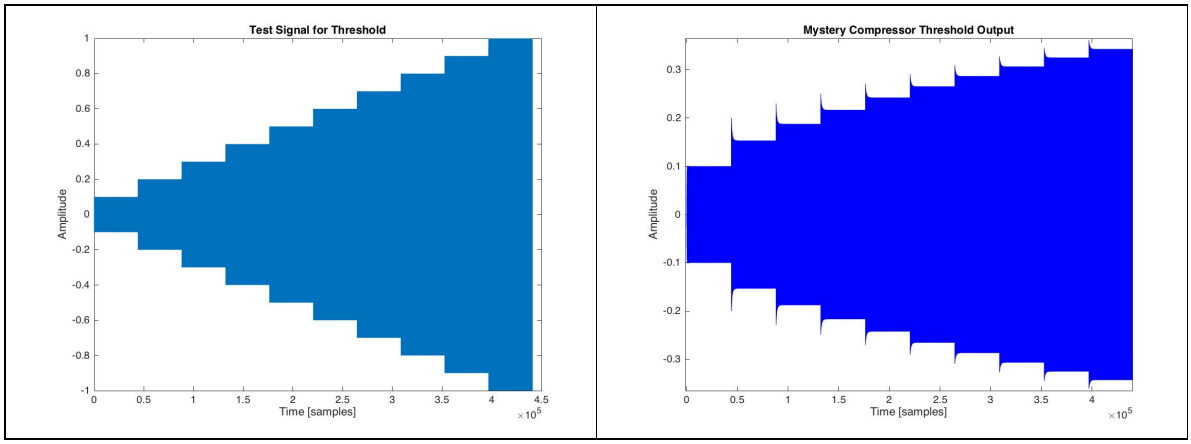


Figure 10, output from test signal vs. input to system.

It was clear that the threshold resided somewhere between 0.1 and 0.2, so another test signal was coded to sweep in this range (Figure 11).

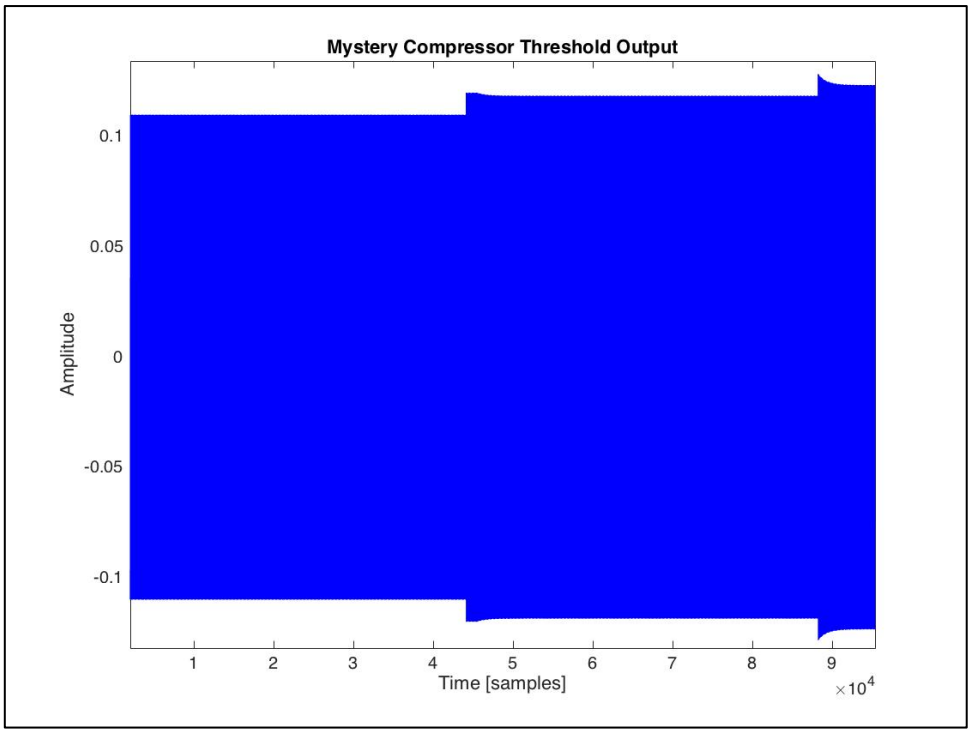


Figure 11, fine-grain test near 0.1 (-20 dB).

After this second test, it can be said with some confidence that the threshold is very close to 0.1, or -20 dB. It is certainly between 0.1 and 0.01.

Although we don't know anything about the gain computer, an estimate of the compression ratio can be deduced by comparing the input signal level to the output signal level (Figure 12).

Input Level	dB In	Output Level	dB Out
0.1	-20	0.1	-20
0.2	-14	0.15	-16.5
0.3	-10.5	0.19	-14.5
0.4	-8	0.22	-13
0.5	-6	0.24	-12
0.6	-4.5	0.26	-11.5
0.7	-3	0.28	-11
0.8	-2	0.30	-10.5
0.9	-1	0.32	-10
1	0	0.34	-9.5

Figure 12, input, output relation through compressor.

Plotting this and fitting a line, we find a slope of about 0.2 (Figure 13).

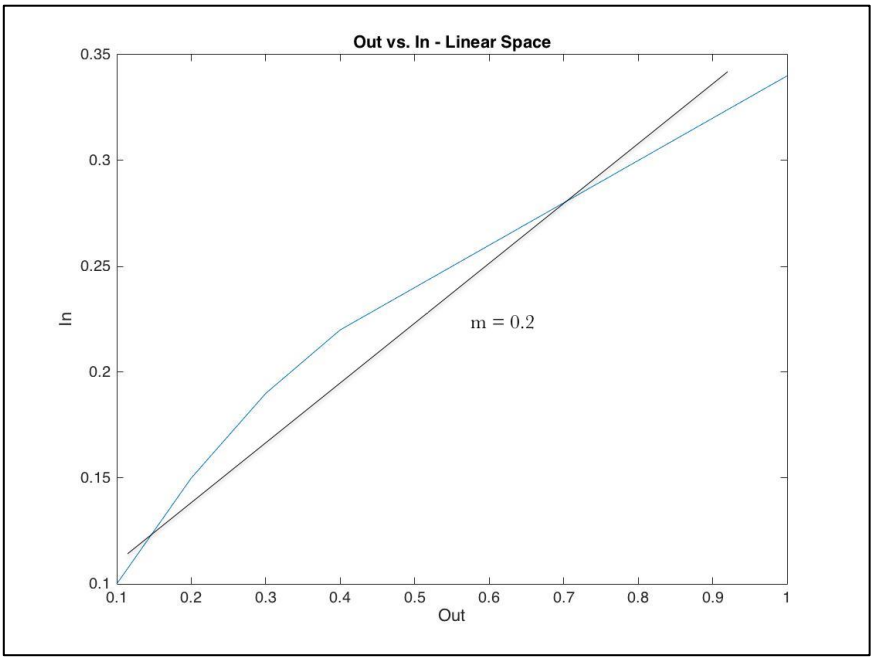


Figure 13, slope of input vs. output for mystery compressor.

This suggests that the compression ratio could be somewhere near 5:1.

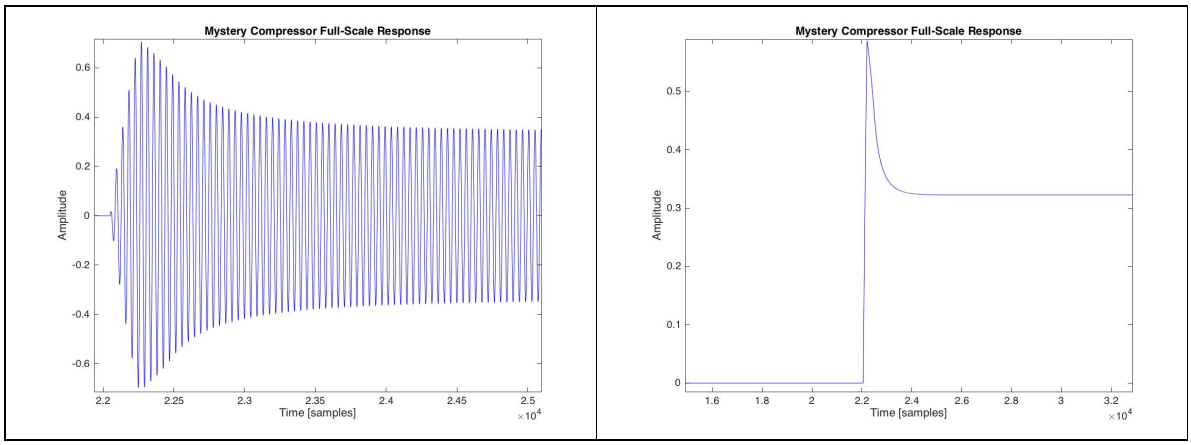
4(b) Possible pseudo-code is presented in Figure 14.

```
Feed Forward
if ( input > levelEstimate )
     levelEstimate += b0_a * ( input - levelEstimate );
 else
     levelEstimate += b0_r * ( input - levelEstimate);

Feedback
if ( output > levelEstimate )
     levelEstimate += b0_a * ( output - levelEstimate );
 else
     levelEstimate += b0_r * ( output - levelEstimate);
```

Figure 14, pseudo-code for f.f. and f.b. designs.

4(c) To determine whether the compressor uses RMS or peak detection, two signals were created in MATLAB: one full-scale sinusoid, and one full-scale square wave. If the compressor approaches a value of 1/sqrt(2) for the sine wave, and 1/2 for the square wave, then we know that this is an RMS detector. Any values close to these could also be good indicators. The observed outputs were as follows in Figure 15.



Figures 15 shows the responses to full-scale inputs.

The signal didn't approach 0.7 in the sin-wave case, nor did it approach 1/2 in response to a step input. Therefore, it is reasonable to assume that this compressor is **not** using an RMS scheme. It could be a peak detector, or some other design.

4(d) Attack and release times.

Approximate attack estimate: **5 mS** (five cycles of a 1kHz test signal).
Approximate release estimate: **500 mS** (25,000 samples / 44100 samples).

4(e) Pseudo-code to implement such a detector is presented in Figure 16.

```
void process (input, & output) {
        if ( fabs( input ) > levelEstimate )
                levelEstimate += b0_a * ( fabs( input ) - levelEstimate );
        else
                 levelEstimate += b0_r * ( fabs( input ) - levelEstimate );
 output = levelEstimate;
}
```

Figure 16, code for a peak detector.