Course Name: Digital Signal Processing Design

Course Number and Section: **14:332:447:01**

Dynamic Range Control Compressors, Limiters, Expanders, and Noise Gates

Submitted by: Lance Darragh

Introduction

Here we are going to learn how to design and implement a dynamic range control system. The **dynamic range** of a signal is the ratio of the largest value a signal has to the smallest value that signal has. Altering the dynamic range of a signal has many applications, especially in music, television, and film applications, where the range of the audio signals need to be appropriate for the right playback devices. If the audio is too loud it will clip and cause pops and clicks in the speakers during playback, and if it's too soft it can be difficult to hear. To address the former problem we'll be designing **compressors** and **limiters**, which reduce the dynamic range of the signal, and to address the latter we'll design **expanders** and **gates**, which increase the dynamic range of the signal.

Some useful quantities for these applications will be the **root-mean-square** and **absolute average** of a signal x(n). The root-mean-square is defined as

(1) $$\overline{x^2(t)} = \frac{1}{T}\int_0^T \mid x(t) \mid^2 \, dt$$

For our experiments we'll be using sinusoids of the form:

(2) x(t) = Acos(2πft)

so we have

(3) $$\overline{x^2(t)} = \frac{1}{T}\int_0^T \mid Acos(2\pi ft) \mid^2 \, dt$$

(4) $$= \frac{1}{T}\int_0^T A^2 cos^2(2\pi ft)dt$$

(5) $$= \frac{A^2}{2T}\int_0^T 1 - cos(2\pi ft)dt$$

(6) $$= \frac{A^2}{2T}\left[t - \frac{sin(2\pi ft)}{2\pi f}\right]_0^T$$

(7) $$= \frac{A^2}{2T}\left\{\left[T - \frac{sin(2\pi)}{2\pi f}\right] - \left[0 - \frac{sin(0)}{2\pi f}\right]\right\}$$

(8) $$\overline{x^2(t)} = \frac{A^2}{2}$$

The absolute average of a signal is defined as

(9) $$\overline{\mid x(t) \mid} = \frac{1}{T}\int_0^T \mid x(t) \mid dt$$

giving us

(10) $$\overline{\mid x(t) \mid} = \frac{1}{T}\int_0^T \mid Acos(2\pi ft) \mid dt$$

(11) $$= \frac{1}{T}\left[\int_{-T/4}^{T/4} Acos(2\pi ft)dt - \int_{T/4}^{3T/4} Acos(2\pi ft)dt\right]$$

(12) $$= \frac{A}{T}\left\{\left[\frac{sin(2\pi ft)}{2\pi f}\right]_{-T/4}^{T/4} - \left[\frac{sin(2\pi ft)}{2\pi f}\right]_{T/4}^{3T/4}\right\}$$

(13) $$= \frac{A}{T}\left\{\left[\frac{sin(\pi/2)}{2\pi f} - \frac{sin(-\pi/2)}{2\pi f}\right] - \left[\frac{sin(3\pi/2)}{2\pi f} - \frac{sin(\pi/2)}{2\pi f}\right]\right\}$$

(14) $$= \frac{A}{T}\left(\frac{1}{2\pi f} + \frac{1}{2\pi f} + \frac{1}{2\pi f} + \frac{1}{2\pi f}\right)$$

(15) $$= \frac{A}{T}\left(\frac{4T}{2\pi}\right)$$

(16) $$\overline{\mid x(t) \mid} = \frac{2A}{\pi}$$

A central concept in dynamics processors is that of a **threshold**. This is a user-definable value to determine which signal amplitude values are attenuated and which are allowed to pass through the processor unaffected. Another fundamental property is the amount of attenuation. This is known as the **ratio**. For example, if a signal is 4 dB above the threshold we can reduce it to being only 1 dB above the threshold by attenuation it with a 4:1 ratio. This is an example of a compressor. A compressor with an extremely high ratio is a known as a limiter, and can be designed to prevent signals from passing beyond a certain value altogether, hence the name limiter. By choosing to attenuate samples with amplitude values below the threshold we increase the dynamic range, designing an expander. An expander with an extremely high ratio is known as a gate, and it attenuates signals to be imperceptibly small, not allowing them to pass through the "gate" altogether.

Having signals instantly jump from one value to another can sound unnatural and these results can be undesirable. So we allow the samples that cross the threshold to reach their desired output values over a small window of time and similarly to return to their normal values. These time values are called the the **attack time** and **release time**, respectively. For a compressor, the attack time is the time it takes for a signal that's above the threshold to reach it's desired value, and the release time is the time is takes for a signal value that goes below the threshold to go back to its desired value, the value of the input signal (remember that the compressor is inactive here). Typical values of attack and release times are on the order of a few milliseconds.

The input/output relationship for dynamics processors is

(17) $y = y_0(x/x_0)^{\rho}$

where x_0 is the threshold. In dB we have

(18) $20\log_{10}(y/y_0) = \rho*20\log_{10}(x/x_0)$

so that ρ can be viewed as linear parameter in relating the input signal in dB to the output signal in dB, and it is the reciprocal of the ratio. So for a 4:1 ratio, we have $\rho = 0.25$. A compressor is active only if $x \geq x_0$, while $\rho < 1$. For $x < x_0$ the signal is unaffected (except during release times). Similarly, an expander is active only if $x \leq x_0$, while $\rho > 1$.

The input output relationship of equation (17) is only to demonstrate the properties of a dynamics processor and is valid for constant signals, but since our signal varies in time we need to write this equation more generally as

(19) $y = Gx$,

where G is the gain applied to the signal and is a nonlinear function of the input.

(20) $G = G_0(x)^{\rho-1}$

Since this equation is nonlinear and the amount of attenuation changes over time, we have a non-linear, time-varying system. Dynamics processors are a type of **adaptive signal processing**. To compute the appropriate gain to apply to the signal we first take a local average of the signal because using instantaneous peak values would lead to jitter in the output levels. To do this, first we need to detect the input signal's level and use that to determine a control signal for each sample, c_n, which will be used to determine the gain for each sample, g_n. We can use either the instantaneous peak values $|x_n|$, the envelope of x_n, or the root-mean-square value of x_n. To program the envelope detector we first take the absolute value of the signal and input that into an exponentially-weighted moving average (EMA) filter. By taking the local average we eliminate high-frequency jitter, so this is a type of lowpass filter. Using a first order filter we have the difference equation for the control signal.

(21) $c_n = \lambda c_{n-1} + (1 - \lambda)|x_n|$

Where λ is known as the "forgetting" parameter. It's between 0 and 1, and is related to the attack and release times. We'll define this parameter shortly. Because the control signal will need to be computed differently for the attack and release times, we have the following relationship between the input signal and the control signal.

(22) $$c_n = \begin{cases} \lambda_a c_{n-1} + (1 - \lambda_a) \mid x_n \mid & , \mid x_n \mid \geq c_{n-1} \\ \lambda_r c_{n-1} + (1 - \lambda_r) \mid x_n \mid & , \mid x_n \mid \leq c_{n-1} \end{cases}$$

So if the signal is rising we use λ_a, related to the attack time, or if the signal is falling we use λ_r related to the release time. The transfer function of this filter is

(23) $H(z) = (1 - \lambda)/(1 - \lambda z^{-1})$,

the impulse response and unit step response are,

(24) $h_n = (1 - \lambda)\lambda^n u(n)$, $(h * u)_n = (1 - \lambda^{n+1})u(n)$

From the impulse response we can see that the filter decays exponentially once enacted by the step input, which corresponds to when the input signal being examined crosses the threshold value. λ is related to the effective time constant of this exponential decay, t_{eff}. To translate this to discrete time, we let

(25) $t_{eff} = n_{eff}T_s$

n_{eff} is the effective number of samples for the signal to converge to within an arbitrarily small value, ϵ.

In our processor, we'll be designing it to operate at an 8 kHz sampling rate with a 20-dB attack time of 2 ms and a 20-dB release time of 10 ms. As this implies, these will be the times required to reach 20 dB of attenuation or for the signal to return from being attenuated by 20 dB. Dynamics processors can be designed to reach practically any amount attenuation, typically by allowing a variable ratio parameter.

To determine ϵ for a 20-dB attenuation, we compute the following,

(26) $20\log_{10}(\epsilon) = -20$

(27) $\log_{10}(\epsilon) = -1$

(28) $\epsilon = 10^{-1} = 0.1 \approx e^{-2.3}$

To match this to the desired attack or release time, t_{eff}, and map it to the appropriate λ, we note that

(29) $\lambda^{neff} = \epsilon$

(30) $\lambda = e^{-2.3Ts/teff}$

So that,

(31) $\lambda_a = e^{-2.3/(8000*0.002)} \approx 0.8661$, and

(32) $\lambda_r = e^{-2.3/(8000*0.01)} \approx 0.9717$

Now we have everything we need to calculate our control signal for each sample of the input. After we calculate the control signal we'll use that to calculate the appropriate gain to be applied to each sample, g_n. Letting c_0 denote the threshold, for a compressor/limiter they are defined as:

(33) $$g = F(c) = \begin{cases} \left(\frac{c}{c_0}\right)^{\rho-1} & , c \geq c_0 \\ 1 & , c \leq c_0 \end{cases}$$

and for an expander/gate:

(34) $$g = F(c) = \begin{cases} 1 & , c \geq c_0 \\ \left(\frac{c}{c_0}\right)^{\rho-1} & , c \leq c_0 \end{cases}$$

We'll be passing these gain values through a smoothing filter because we don't want the gain changing dramatically between peaks of the input signal. It's the overall amplitude of the signal that we want to change. Here we will use two types of averaging filters, FIR and EMA.

(35) $$G_n = \frac{1}{L}[g_n + g_{n-1} + ... + g_{n-L+1}] \qquad \text{(FIR)}$$

(36) $$G_n = \lambda G_{n-1} + (1-\lambda)g_n \qquad \text{(EMA)}$$

We'll be using both to demonstrate that different filters can be used, but these two filters act equivalently if we let

(37) $$L \approx \frac{1+\lambda}{1-\lambda}$$

where we round L up to the nearest integer, or equivalently,

(38) $$\lambda \approx \frac{L-1}{L+1}$$

In our examples we'll be using our value of $\lambda_a = 0.8661$ for λ (corresponding to a value of L = 14 for the FIR filter), but this EMA filter can be defined similar to that of the control signal's to respond differently depending on whether the signal is rising or falling.

(39) $$G_n = \begin{cases} \lambda_a G_{n-1} + (1-\lambda_a) g_n & , g_n \geq g_{n-1} \\ \lambda_r G_{n-1} + (1-\lambda_r) g_n & , g_n < g_{n-1} \end{cases}$$

The benefit of using the EMA filter is that it is slightly more computationally efficient with less data shifts and additions, but due the relatively slow sampling rate of audio applications the FIR filter will work just fine too, even if you use a linear buffer. For our experiments we'll be using the following signals.

(40) $$x(t) = \begin{cases} 2cos(600\pi t) & , 0 \leq t < 25msec \\ 4cos(1200\pi t) & , 25 \leq t < 50msec \\ (1/2)cos(2400\pi t) & , 50 \leq t < 75msec \end{cases}$$

Using equation (16), the mean-absolute values of these signals are, (2/π), (4/π), and (1/2π).

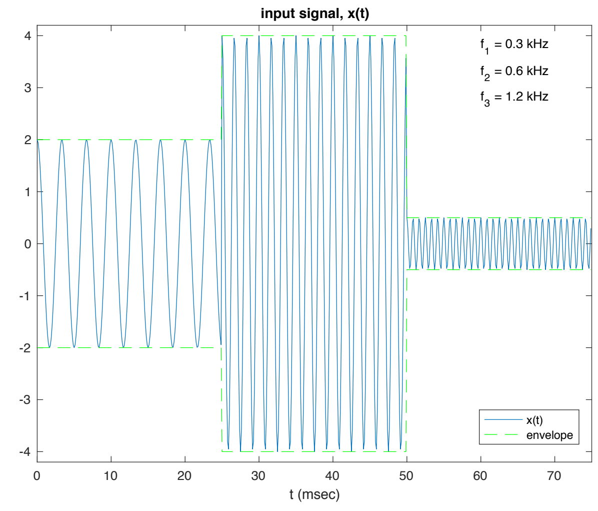


Fig. 1 Input Signal

The MATLAB code to design the compressor with an FIR gain-smoothing filter is in Appendix A.7. Running this signal through the compressor results in the following output. The internal signals described above are shown here as well.

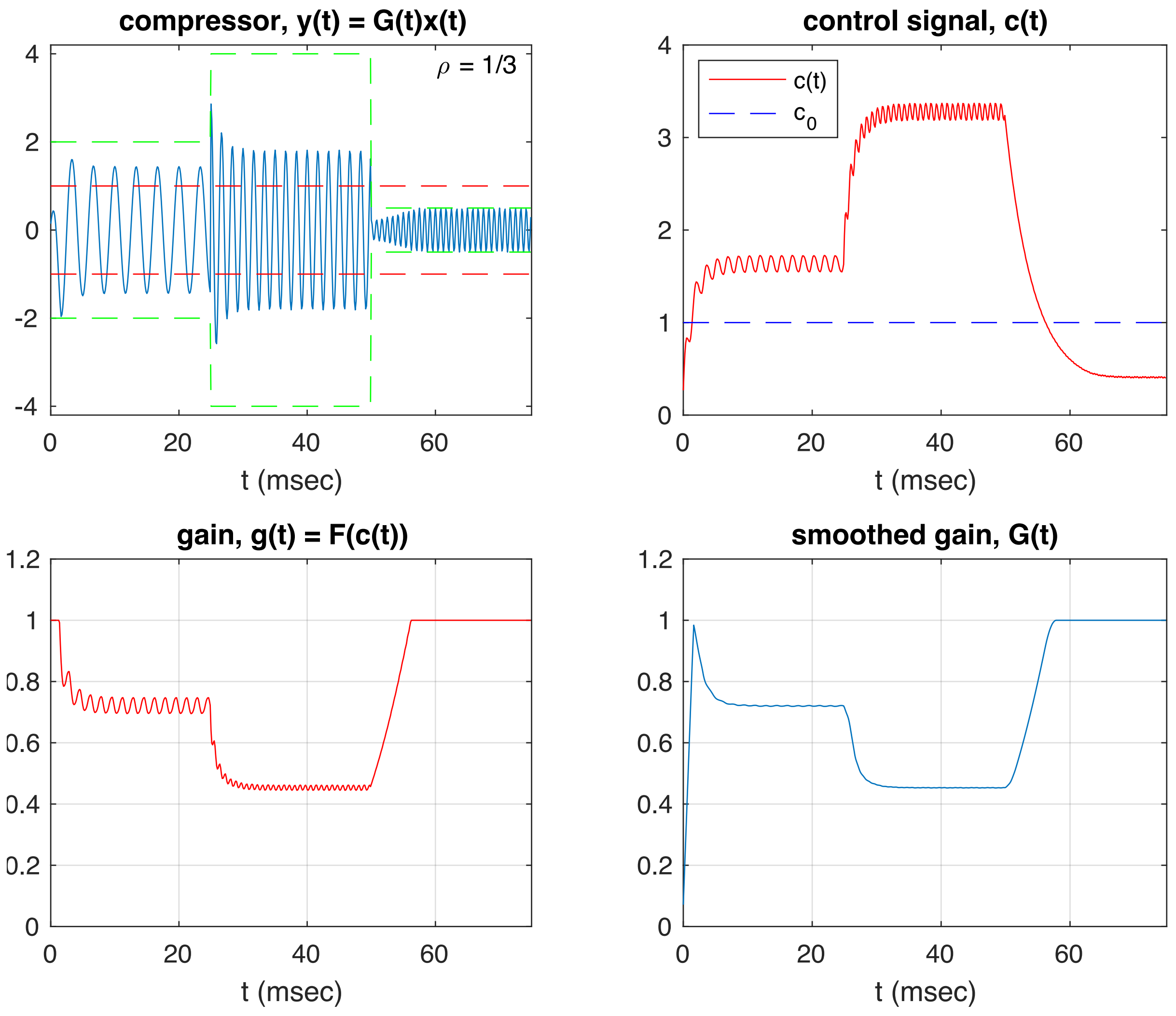


Fig. 2 Compressor, ρ = 1/3, c_0 = 1, using FIR smoother

In the picture of the compressor's output we can see the transient behavior of each region of the signal. Once the signal goes above the threshold it takes a few milliseconds to compress the signal. This is the attack time. Similarly, once the signal goes below the threshold it takes a while to return to it's uncompressed value. This is the release time. The threshold is set at 1. In the region in which the compressor is active, the signal is at a value of 4, which is above the threshold by an amount of 3 times the threshold. It is reduced to being above the threshold by an amount of only 1 times the threshold. This is a compression ratio of 3:1, with ρ = 1/3.

Now we'll modify the compressor design to act as a limiter by increasing the ratio to be practically infinity by setting $\rho = 1/100$. This is a rather extreme setting, typical settings are more like $\rho = 1/10$, but more extreme settings can be used depending on how much you want to prevent a signal from passing a certain level. We'll use an EMA gain-smoothing filter in this design, and the code can be found in Appendix A.8. Running the same input signal through this limiter with the threshold set at 2.2, as to only effect the middle region, we obtain the following signals.

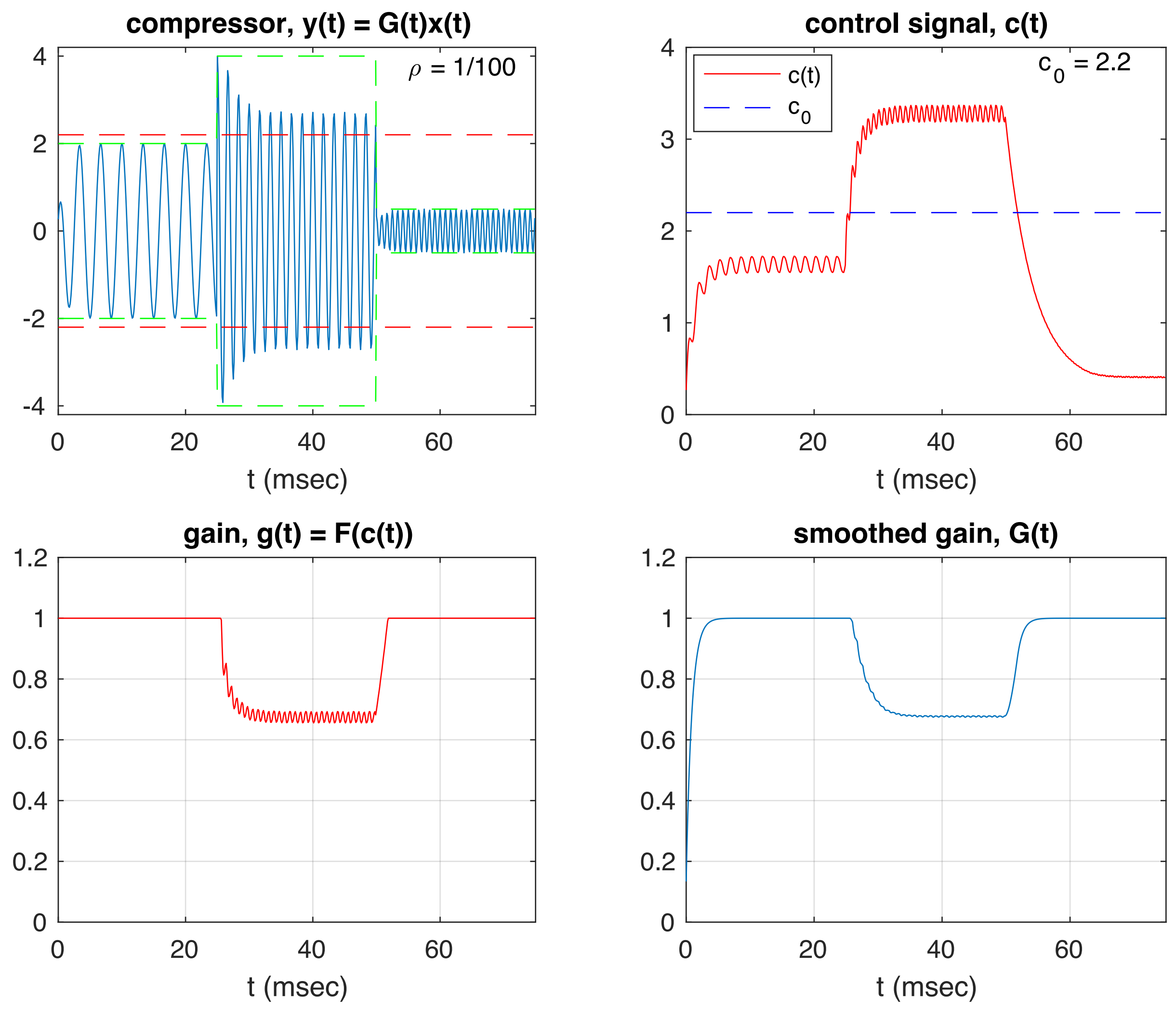


Fig. 3 Limiter, $\rho = 1/100$, $c_0 = 2.2$, using EMA smoother

Next, we'll modify the design to work as an expander by changing the conditional statement that assigns the gain values g_n to be according to equation (34). We also need to change the use of the attack and release time constants, so that the expander "attacks" the signal when it goes below the threshold and "releases" when it goes back above it. This can be done by changing the roles of the inequalities in equation (39). To only effect the lowest amplitude signal, we'll set the threshold at 0.75, we'll set $\rho = 3$ (this results in a 3:1 ratio for an expander), and we'll keep using the EMA filter for now. The MATLAB code is in Appendix A.9, and here are the output and internal system signals.

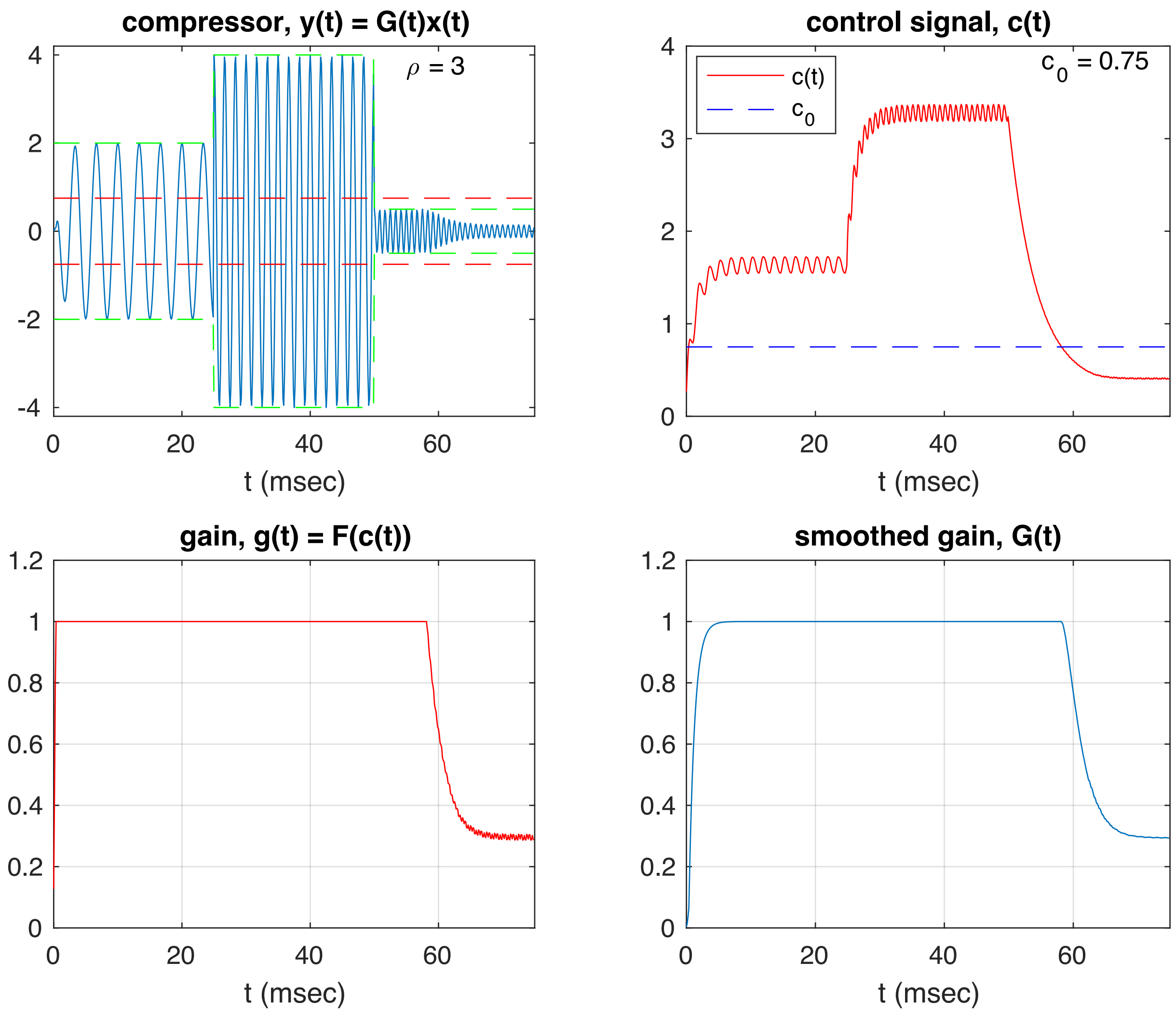


Fig. 4 Expander, $\rho = 3$, $c_0 = 0.75$, using EMA smoother

Now we'll change ρ to 10 and set the threshold to 2.2 so that it acts as a noise gate, suppressing the first and last sinusoids, allowing the 0.6 kHz signal to stand out more.

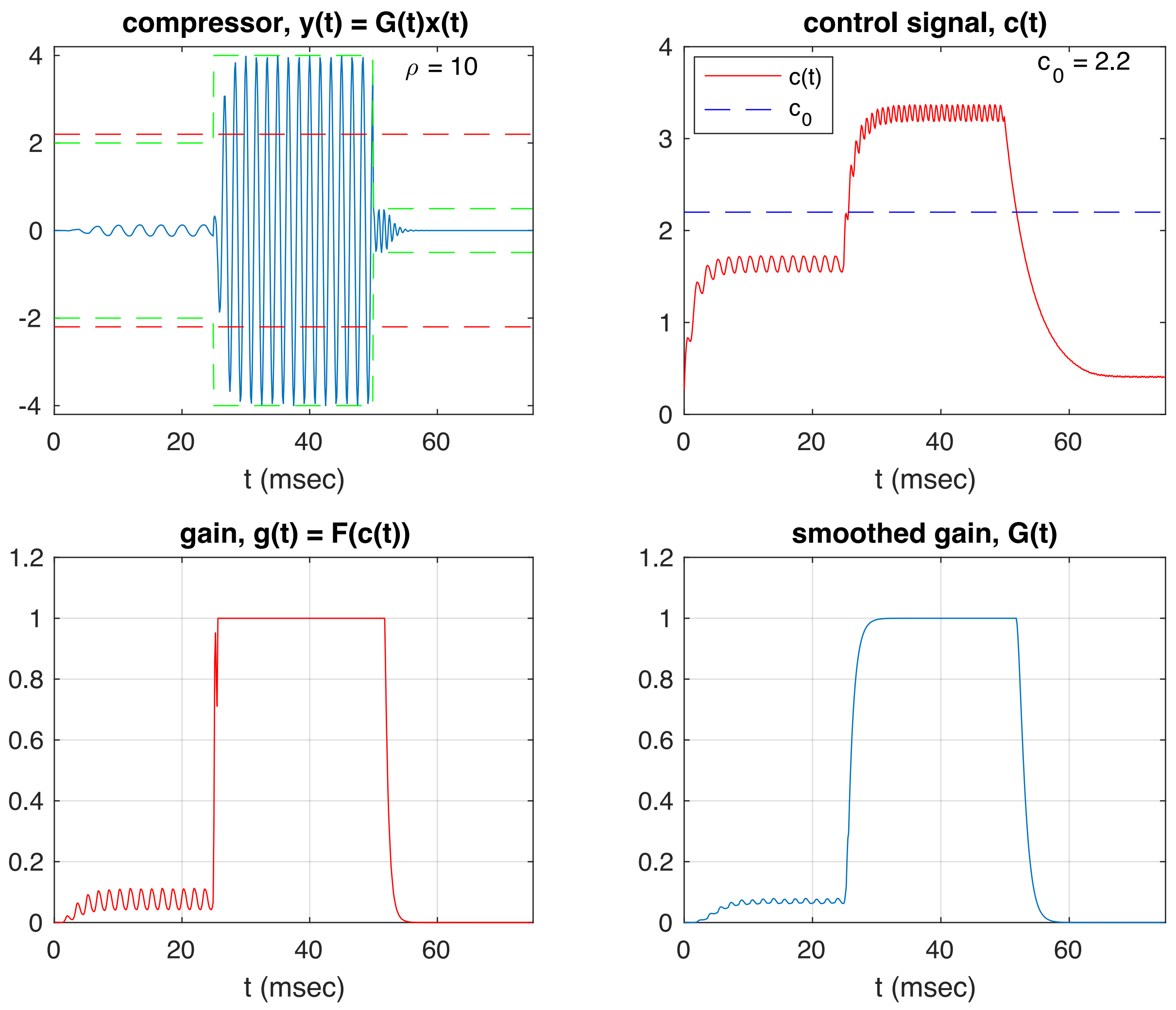


Fig. 6 Noise gate suppressing f_1 and f_3, $\rho = 10$, $c_0 = 2.2$

The MATLAB code for this design is in Appendix A.10. If we change the threshold back to 0.75 we can see how the gate acts on just the smallest parts of the signal This is typical in applications where low-level noise like interference or background noise should be removed. Note that this can be used in other applications besides audio, such as removing 60 Hz electrical hum from a medical imaging signal. We'll set $\rho = 100$ to be a bit more extreme. The MATLAB code is in Appendix A.11

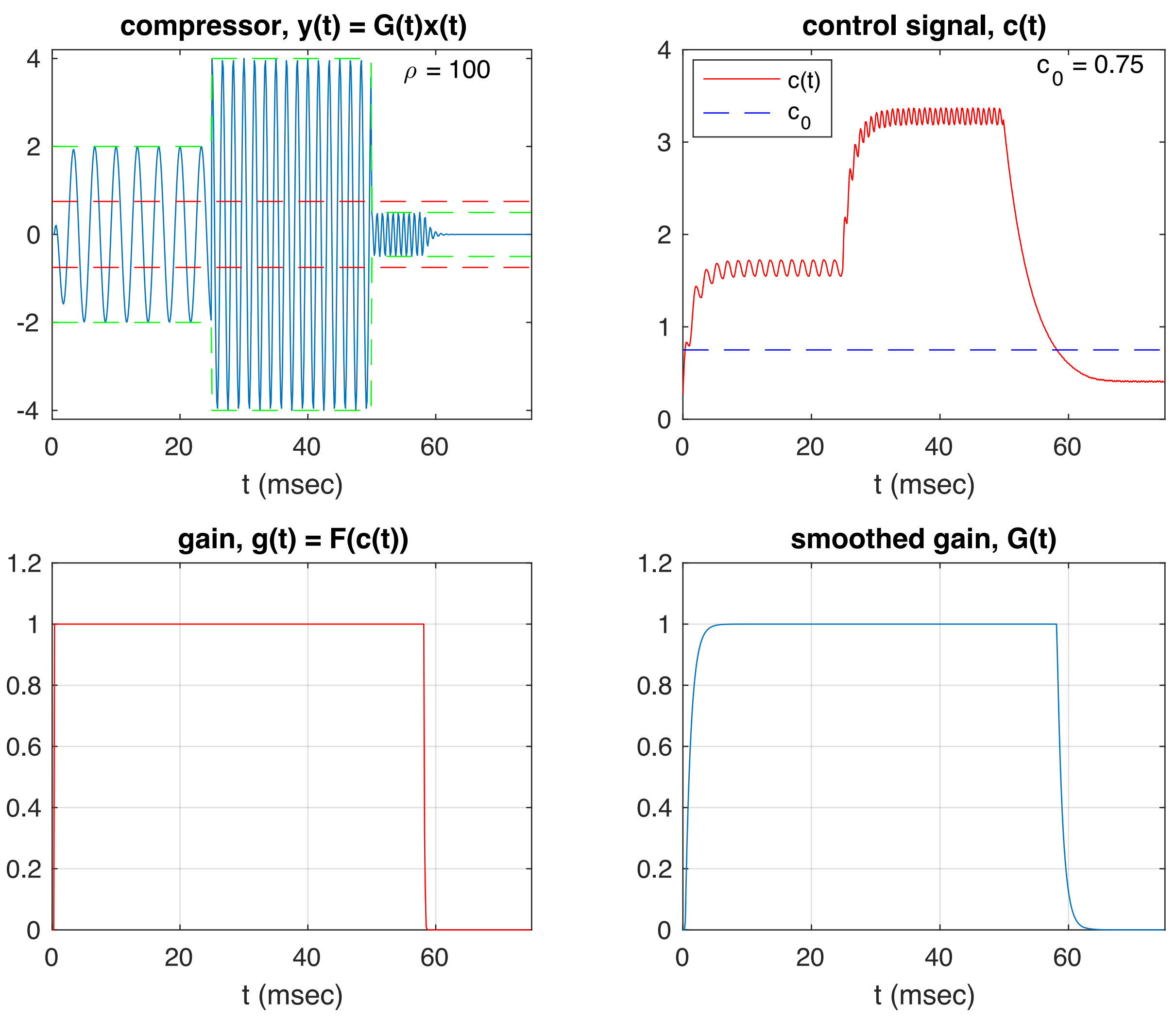


Fig. 7 Noise gate suppressing f_3 only, $\rho = 10$, $c_0 = 0.75$

For our last example we'll be looking at a common application in the audio industry, **ducking**. Imagine working as a mixer making a commercial at a film studio, and you have music that's to be mixed in with a person speaking. Turning the dialogue up sounds unnatural in the scene, but turning the music down makes it too low. The answer is to use adaptive signal processing to compress the music signal only when the person is speaking, making the music signal "duck" when the person talks. To do this we use the person speaking as the input to our control signal to compute the applied gain but apply that gain to the music instead of the speaker. This is known as using a **side-chain input**. With the appropriate choice of threshold, ratio, and attack and release times we can mix these two audio signals in a way that sounds better for the viewer, allowing them to hear the speech more clearly while still being able to hear the music in the background. Letting $x_s(t)$ denote the speech signal, $x_m(t)$ denote the music signal, and x(t) denote the combination of the two, $x_s(t) + x_m(t)$, we have the following input signals.

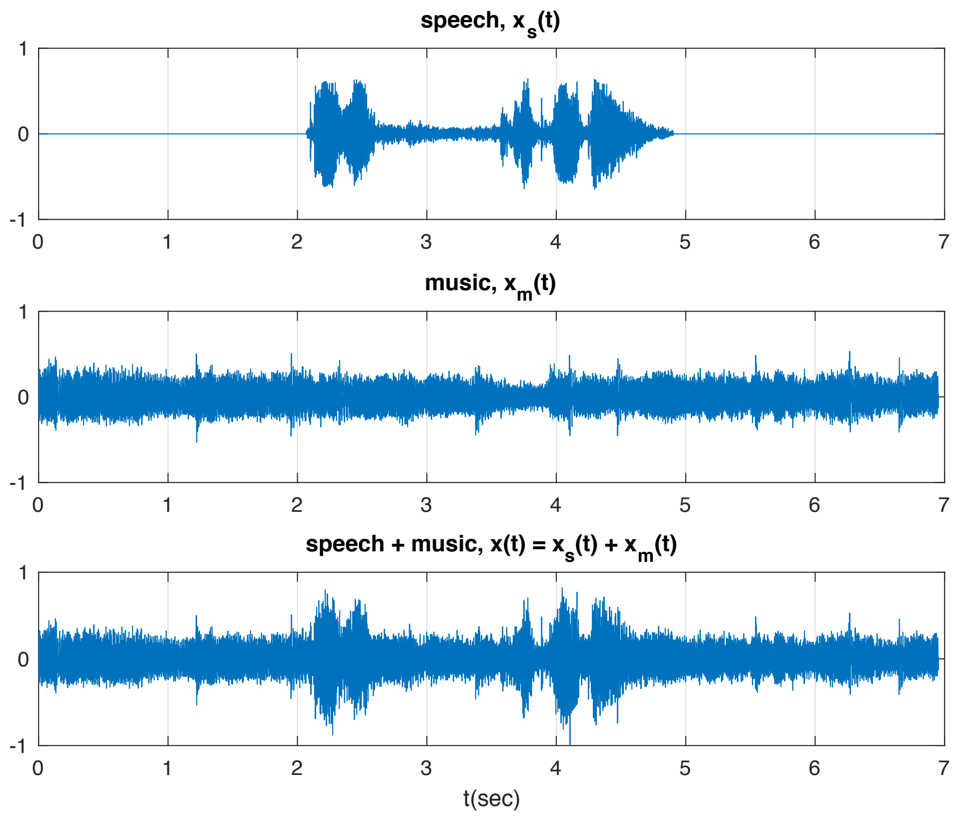


Fig. 7 Input signals for ducker

The input/output relationship is described by

(41) $y(t) = x_s(t) + G(t)x_m(t)$

Letting $y_m(t)$ denote the compressed music signal we have

(42) $y(t) = x_s(t) + y_m(t)$

The sampling rate of these signals is 44.1 kHz. Using an attack time of 50 msec, a release time of 500 msec, a ratio of 5:1, and a threshold corresponding to a 40 dB below the maximum value of the speech signal, we have $\lambda_a = 0.997916$, $\lambda_r = 0.999791$, $\rho = 1/5$, and $c_0 = 0.06428$ (0.01 * 0.6428). The MATLAB code is in Appendix A.12. Using these settings we obtain the following plots,

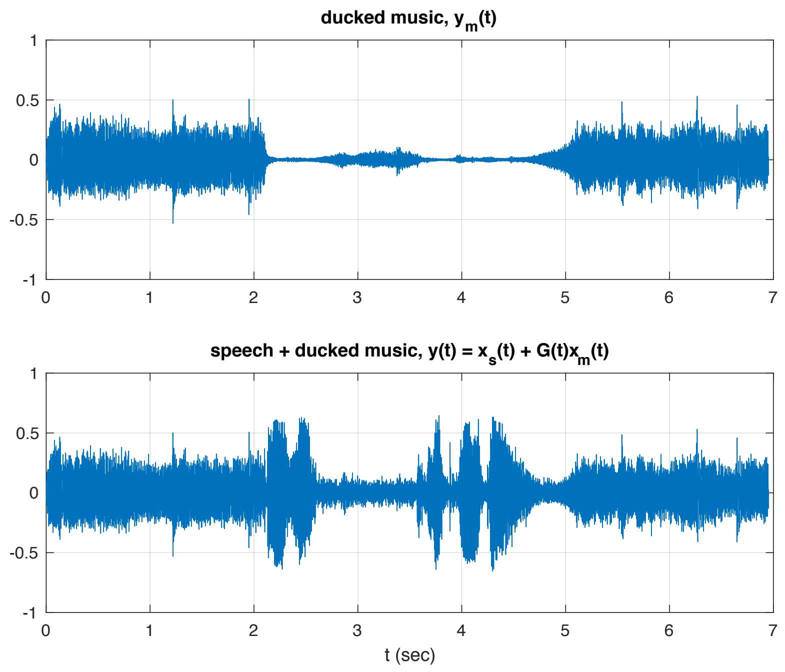


Fig. 7 Ducked music and output signal for ducker

and these are the plots of the internal signals.

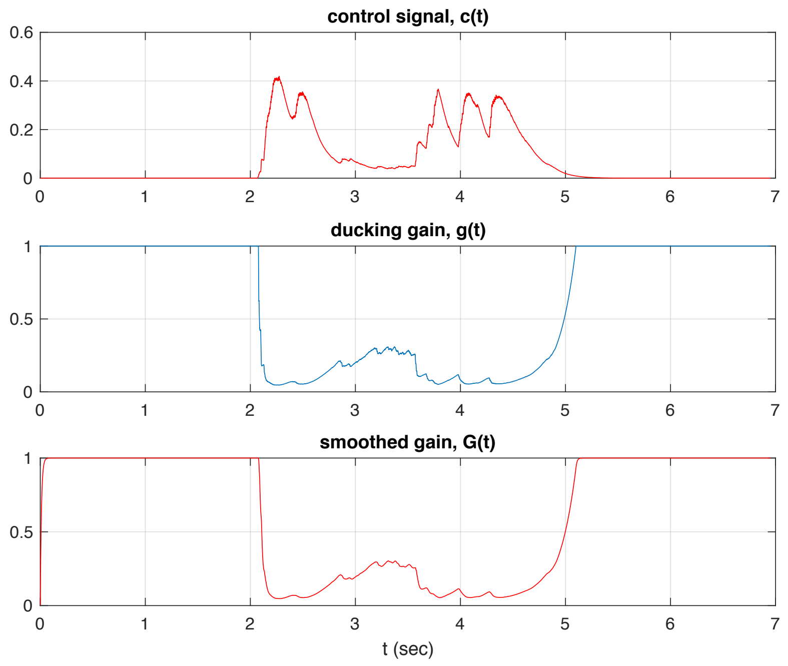


Fig. 9 Control and gain signals for ducker

Table of Contents

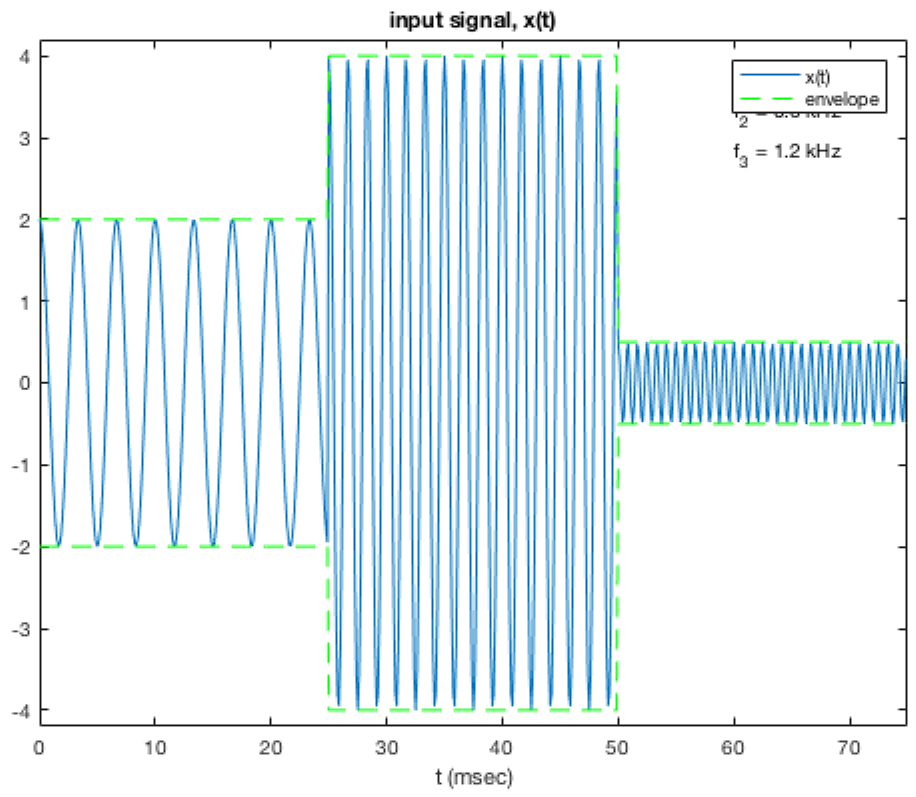
Appendix A.7: Dynamic Range Controllers

```
% Define input signals and compressor initial parameters
fs = 8000; % sampling rate
Ts = 1/fs; % sampling period
ta = 0.002; % attack time
tr = 0.01; % release time
Aa = exp(-2.3*Ts/ta); % attack forgetting factor
Ar = exp(-2.3*Ts/tr); % release forgetting factor

% Defining three sinusoids to be played consecutively
A1 = 2; % Amplitude of first sinusoid
f1 = 300; % f1 = 0.3 kHz
t1 = [0:Ts:0.025 - Ts]; % t1 from 0 to 25 msec
A2 = 4; % Amplitude of second sinusoid
f2 = 600; % f2 = 0.6 kHz
t2 = [0.025:Ts:0.05 - Ts]; % t2 from 25 msec to 50 msec
A3 = 0.5; % Amplitude of third sinusoid
f3 = 1200; % f3 = 1.2 kHz
t3 = [0.05:Ts:0.075 - Ts]; % t3 from 50 msec to 75 msec
x = [A1*cos(2*pi*f1*t1),A2*cos(2*pi*f2*t2),A3*cos(2*pi*f3*t3)];


% For plotting
envInTop = [A1*ones(1,200),A2*ones(1,200),A3*ones(1,200)];
envInBot = [-A1*ones(1,200),-A2*ones(1,200),-A3*ones(1,200)];
t = [t1,t2,t3];
```

Plot Input Signals

```
close all
```

```
plot(t*1000,x,t*1000,envInTop,'--g',t*1000,envInBot,'--g');
axis([0,75,-4.2,4.2]); legend('x(t)','envelope'),
text(60,3.8,'f_1 = 0.3 kHz');text(60,3.3,'f_2 = 0.6 kHz');
text(60,2.8,'f_3 = 1.2 kHz'); xlabel('t (msec)'), title('input signal,
 x(t)')
```



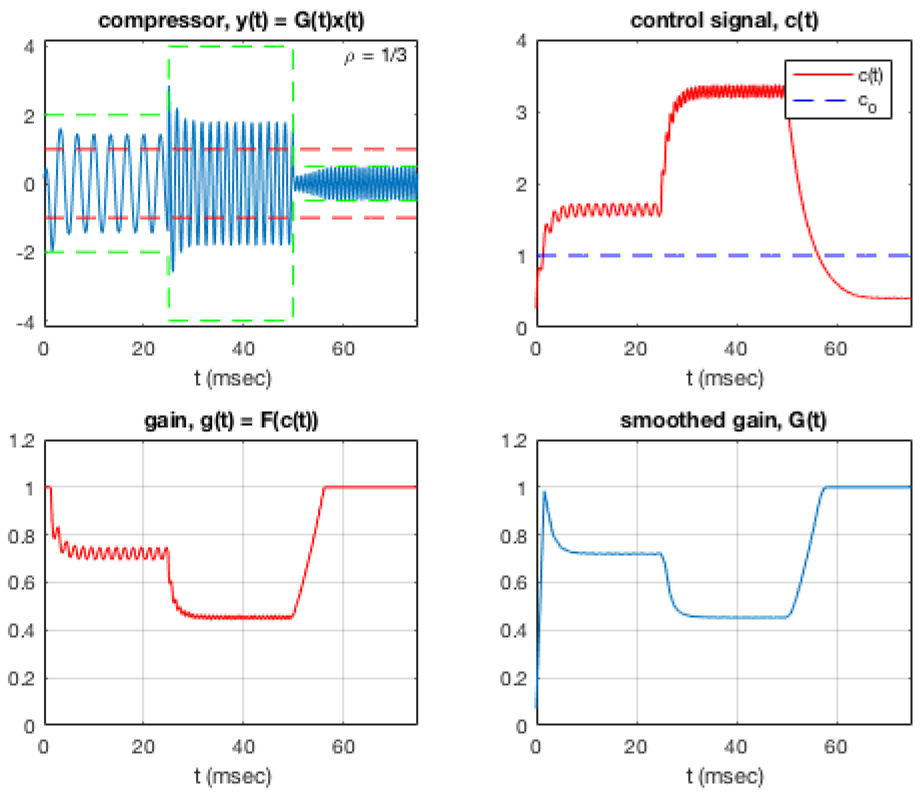
Appendix A.8: Compressor Design

```
Aa = 0.8661; % lambda_a
Ar = 0.9717; % lambda_r
p = 1/3;     % rho
c0 = 1;      % threshold
L = ceil((1 + Aa)/(1 - Aa)); % Length of FIR gain-smoothing filter
g = zeros(1,L); % Internal buffer for FIR gain-smoothing filter
q = 1;                 % Internal buffer index
c = 0;             % Control signal
G = 0;             % Smoothed gain
n = 1;                 % Index variable
N = length(x);     % For limit on n

for n = 1:N
if abs(x(n)) >= c                     % calculate control signal
    c = Aa*c + (1 - Aa)*abs(x(n));
else
    c = Ar*c + (1 - Ar)*abs(x(n));
end
```

```
cPlot(n) = c;    % For plotting

if c == 0        % Calculate gain signal
    g(q) = 1;
elseif c >= c0
    g(q) = (c/c0)^(p-1);
else
    g(q) = 1;
end

gPlot(n) = g(q);    % For plotting

q = q + 1;
if q > L
    q = 1;
end

G = (1/L)*sum(g); % FIR averaging filter

GPlot(n) = G;    % For plotting

y(n) = G*x(n); % Output
end
```

Plot Compressor Results

```
close all

subplot(2,2,1)
plot(t*1000,y,t*1000,envInTop,'--g',t*1000,envInBot,'--g',...
t*1000,c0*ones(1,N),'--r',t*1000, -c0*ones(1,N),'--r')
axis([0,75,-4.2,4.2]),xlabel('t (msec)'),
title('compressor, y(t) = G(t)x(t)'), text(60,3.8,'\rho = 1/3')


subplot(2,2,2)
plot(t*1000,cPlot,'r',t*1000, c0*ones(1,N),'--b')
axis([0,75,0,4]),xlabel('t (msec)'),title('control signal, c(t)'),
legend('c(t)','c_0')

subplot(2,2,3)
plot(t*1000,gPlot,'r')
axis([0,75,0,1.2]),xlabel('t (msec)'),title('gain, g(t) = F(c(t))')
grid on

subplot(2,2,4)
plot(t*1000,GPlot)
axis([0,75,0,1.2]),xlabel('t (msec)'),title('smoothed gain, G(t)')
grid on
```



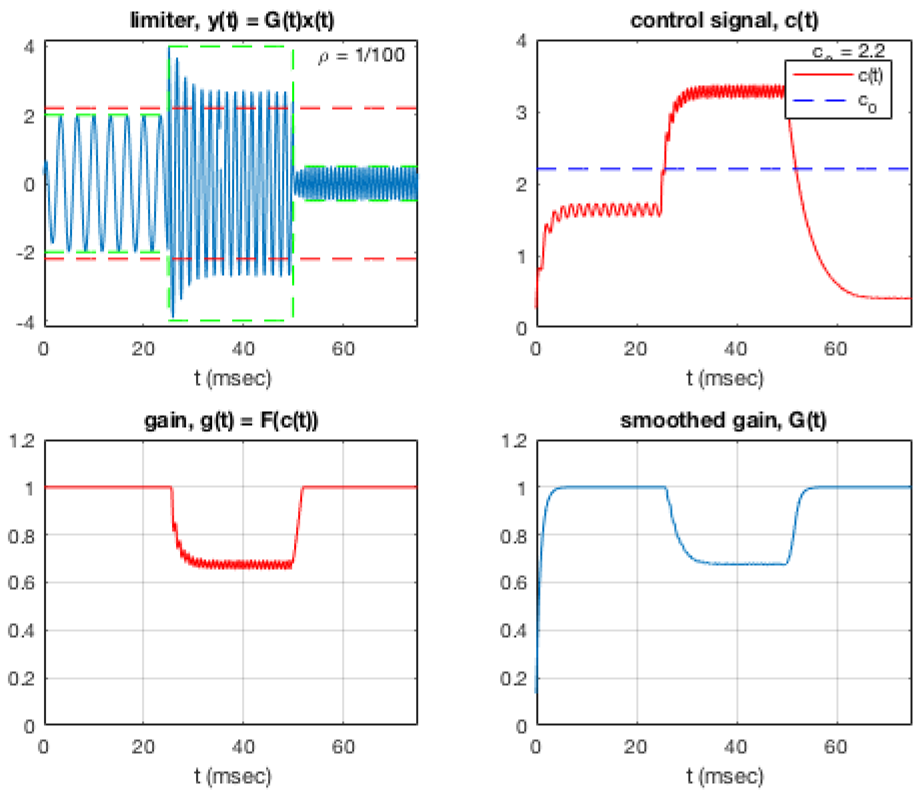
Appendix A.9: Limiter Design

```
Aa = 0.8661; % lambda_a
Ar = 0.9717; % lambda_r
p = 1/100;     % rho
c0 = 2.2;      % threshold
g1 = 0;     % for EMA gain-smoothing filter, g_n
g0 = 0;             % g_{n-1}
c = 0;          % Control signal
G = 0;          % Smoothed gain
n = 1;              % Index variable
N = length(x);    % For limit on n

for n = 1:N
if abs(x(n)) >= c                    % calculate control signal
    c = Aa*c + (1 - Aa)*abs(x(n));
else
    c = Ar*c + (1 - Ar)*abs(x(n));
end

cPlot(n) = c;    % For plotting

g0 = g1;

if c == 0         % Calculate gain signal
```

```
    g1 = 1;
elseif c >= c0
    g1 = (c/c0)^(p-1);
else
    g1 = 1;
end

gPlot(n) = g1;     % For plotting

if g1 >= g0      % EMA smoothing filter
    G = Aa*G + (1 - Aa)*g1;
else
    G = Ar*G + (1 - Ar)*g1;
end

GPlot(n) = G;     % For plotting

y(n) = G*x(n); % Output
end
```

Plot Limiter Results

```
close all

subplot(2,2,1)
plot(t*1000,y,t*1000,envInTop,'--g',t*1000,envInBot,'--g',...
t*1000,c0*ones(1,N),'--r',t*1000, -c0*ones(1,N),'--r')
axis([0,75,-4.2,4.2]),xlabel('t (msec)'),
title('limiter, y(t) = G(t)x(t)'), text(55,3.8,'\rho = 1/100')


subplot(2,2,2)
plot(t*1000,cPlot,'r',t*1000, c0*ones(1,N),'--b')
axis([0,75,0,4]),xlabel('t (msec)'),title('control signal, c(t)'),
legend('c(t)','c_0'), text(55,3.8,'c_0 = 2.2')

subplot(2,2,3)
plot(t*1000,gPlot,'r')
axis([0,75,0,1.2]),xlabel('t (msec)'),title('gain, g(t) = F(c(t))')
grid on

subplot(2,2,4)
plot(t*1000,GPlot)
axis([0,75,0,1.2]),xlabel('t (msec)'),title('smoothed gain, G(t)')
grid on
```



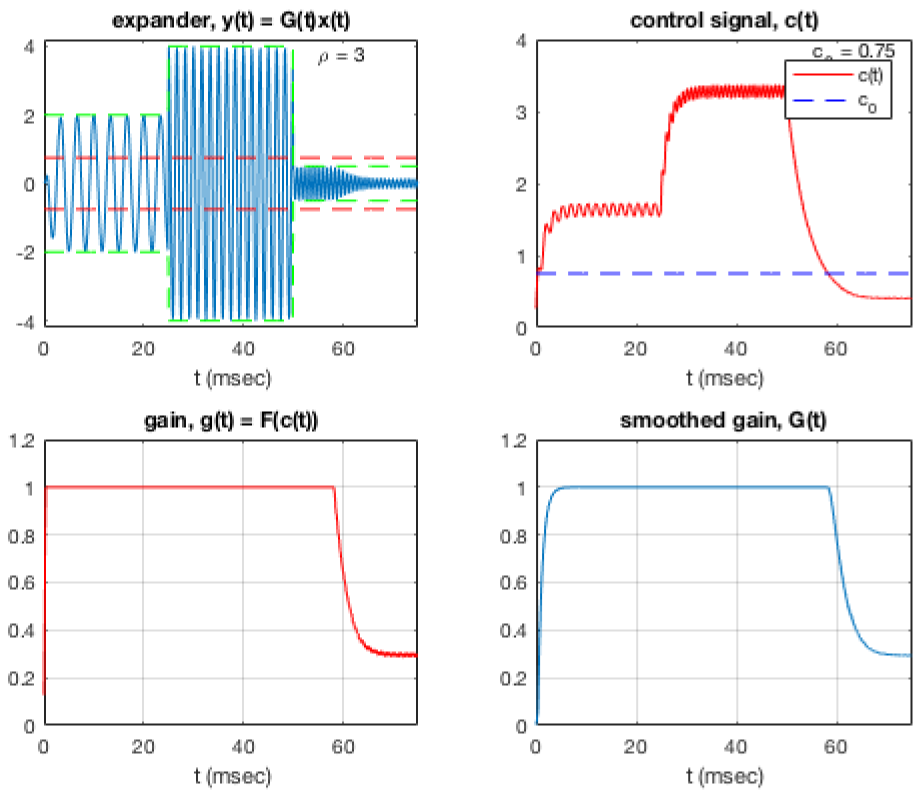
Appendix A.10 Expander Design

```
Aa = 0.8661; % lambda_a
Ar = 0.9717; % lambda_r
p = 3;     % rho
c0 = 0.75;      % threshold
g1 = 0;     % for EMA gain-smoothing filter, g_n
g0 = 0;             % g_{n-1}
c = 0;          % Control signal
G = 0;          % Smoothed gain
n = 1;              % Index variable
N = length(x);    % For limit on n

for n = 1:N
if abs(x(n)) >= c                   % calculate control signal
    c = Aa*c + (1 - Aa)*abs(x(n));
else
    c = Ar*c + (1 - Ar)*abs(x(n));
end

cPlot(n) = c;    % For plotting

g0 = g1;

if c == 0        % Calculate gain signal
```

```
    g1 = 1;
elseif c <= c0
    g1 = (c/c0)^(p-1);
else
    g1 = 1;
end

gPlot(n) = g1;    % For plotting

if g1 <= g0      % EMA smoothing filter
    G = Aa*G + (1 - Aa)*g1;
else
    G = Ar*G + (1 - Ar)*g1;
end

GPlot(n) = G;    % For plotting

y(n) = G*x(n);
end
```

Plot Expander Results

```
close all

subplot(2,2,1)
plot(t*1000,y,t*1000,envInTop,'--g',t*1000,envInBot,'--g',...
t*1000,c0*ones(1,N),'--r',t*1000, -c0*ones(1,N),'--r')
axis([0,75,-4.2,4.2]),xlabel('t (msec)'),
title('expander, y(t) = G(t)x(t)'), text(55,3.8,'\rho = 3')


subplot(2,2,2)
plot(t*1000,cPlot,'r',t*1000, c0*ones(1,N),'--b')
axis([0,75,0,4]),xlabel('t (msec)'),title('control signal, c(t)'),
legend('c(t)','c_0'), text(55,3.8,'c_0 = 0.75')

subplot(2,2,3)
plot(t*1000,gPlot,'r')
axis([0,75,0,1.2]),xlabel('t (msec)'),title('gain, g(t) = F(c(t))')
grid on

subplot(2,2,4)
plot(t*1000,GPlot)
axis([0,75,0,1.2]),xlabel('t (msec)'),title('smoothed gain, G(t)')
grid on
```



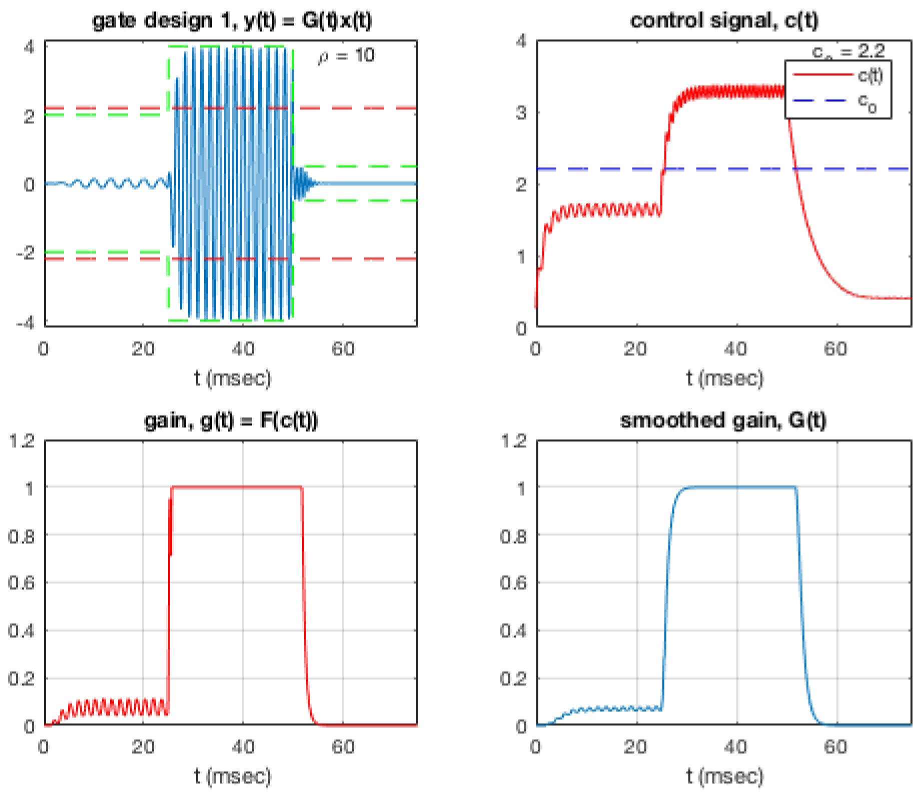
Appendix A.11 Gate Design 1

```
Aa = 0.8661; % lambda_a
Ar = 0.9717; % lambda_r
p = 10;      % rho
c0 = 2.2;       % threshold
g1 = 0;      % for EMA gain-smoothing filter, g_n
g0 = 0;              % g_{n-1}
c = 0;           % Control signal
G = 0;           % Smoothed gain
n = 1;               % Index variable
N = length(x);     % For limit on n

for n = 1:N
if abs(x(n)) >= c                   % calculate control signal
    c = Aa*c + (1 - Aa)*abs(x(n));
else
    c = Ar*c + (1 - Ar)*abs(x(n));
end

cPlot(n) = c;     % For plotting

g0 = g1;

if c == 0          % Calculate gain signal
```

```
    g1 = 1;
elseif c <= c0
    g1 = (c/c0)^(p-1);
else
    g1 = 1;
end

gPlot(n) = g1;     % For plotting

if g1 <= g0      % EMA smoothing filter
    G = Aa*G + (1 - Aa)*g1;
else
    G = Ar*G + (1 - Ar)*g1;
end

GPlot(n) = G;     % For plotting

y(n) = G*x(n); % Output
end
```

Plot Gate Design 1 Results

```
close all

subplot(2,2,1)
plot(t*1000,y,t*1000,envInTop,'--g',t*1000,envInBot,'--g',...
t*1000,c0*ones(1,N),'--r',t*1000, -c0*ones(1,N),'--r')
axis([0,75,-4.2,4.2]),xlabel('t (msec)'),
title('gate design 1, y(t) = G(t)x(t)'), text(55,3.8,'\rho = 10')


subplot(2,2,2)
plot(t*1000,cPlot,'r',t*1000, c0*ones(1,N),'--b')
axis([0,75,0,4]),xlabel('t (msec)'),title('control signal, c(t)'),
legend('c(t)','c_0'), text(55,3.8,'c_0 = 2.2')

subplot(2,2,3)
plot(t*1000,gPlot,'r')
axis([0,75,0,1.2]),xlabel('t (msec)'),title('gain, g(t) = F(c(t))')
grid on

subplot(2,2,4)
plot(t*1000,GPlot)
axis([0,75,0,1.2]),xlabel('t (msec)'),title('smoothed gain, G(t)')
grid on
```



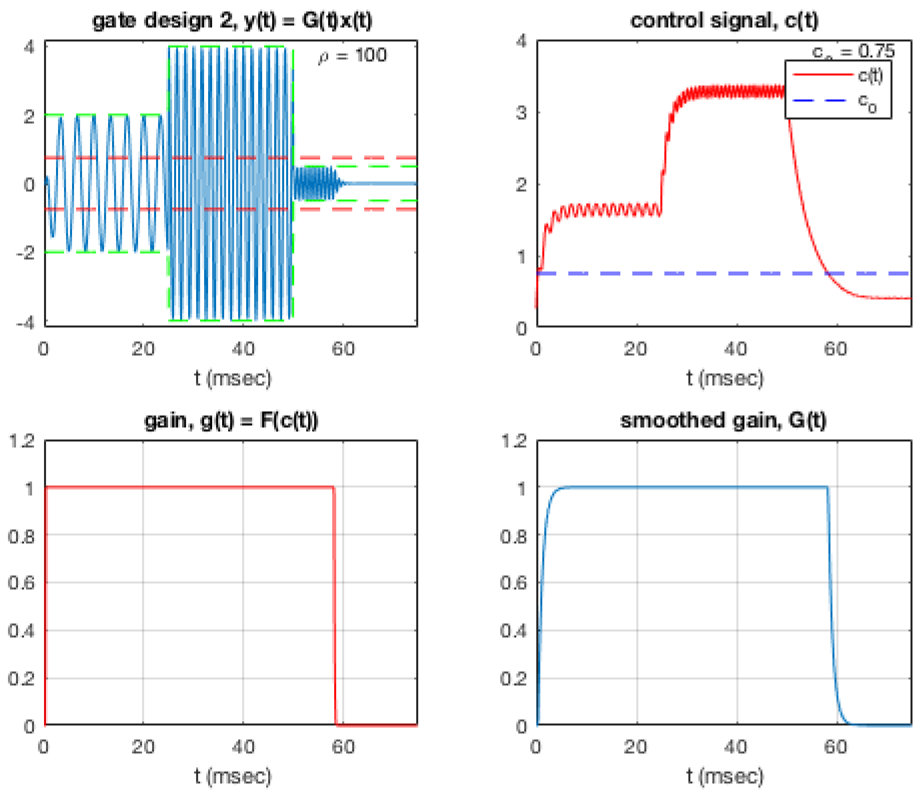
Appendix A.12 Gate Design 2

```
Aa = 0.8661; % lambda_a
Ar = 0.9717; % lambda_r
p = 100;     % rho
c0 = 0.75;      % threshold
g1 = 0;     % for EMA gain-smoothing filter, g_n
g0 = 0;             % g_{n-1}
c = 0;          % Control signal
G = 0;          % Smoothed gain
n = 1;              % Index variable
N = length(x);    % For limit on n

for n = 1:N
if abs(x(n)) >= c                   % calculate control signal
    c = Aa*c + (1 - Aa)*abs(x(n));
else
    c = Ar*c + (1 - Ar)*abs(x(n));
end

cPlot(n) = c;    % For plotting

g0 = g1;

if c == 0        % Calculate gain signal
```

```
    g1 = 1;
elseif c <= c0
    g1 = (c/c0)^(p-1);
else
    g1 = 1;
end

gPlot(n) = g1;    % For plotting

if g1 <= g0     % EMA smoothing filter
    G = Aa*G + (1 - Aa)*g1;
else
    G = Ar*G + (1 - Ar)*g1;
end

GPlot(n) = G;    % For plotting

y(n) = G*x(n); % Output
end
```

Plot Gate Design 2 Results

```
close all

subplot(2,2,1)
plot(t*1000,y,t*1000,envInTop,'--g',t*1000,envInBot,'--g',...
t*1000,c0*ones(1,N),'--r',t*1000, -c0*ones(1,N),'--r')
axis([0,75,-4.2,4.2]),xlabel('t (msec)'),
title('gate design 2, y(t) = G(t)x(t)'), text(55,3.8,'\rho = 100')


subplot(2,2,2)
plot(t*1000,cPlot,'r',t*1000, c0*ones(1,N),'--b')
axis([0,75,0,4]),xlabel('t (msec)'),title('control signal, c(t)'),
legend('c(t)','c_0'), text(55,3.8,'c_0 = 0.75')

subplot(2,2,3)
plot(t*1000,gPlot,'r')
axis([0,75,0,1.2]),xlabel('t (msec)'),title('gain, g(t) = F(c(t))')
grid on

subplot(2,2,4)
plot(t*1000,GPlot)
axis([0,75,0,1.2]),xlabel('t (msec)'),title('smoothed gain, G(t)')
grid on
```

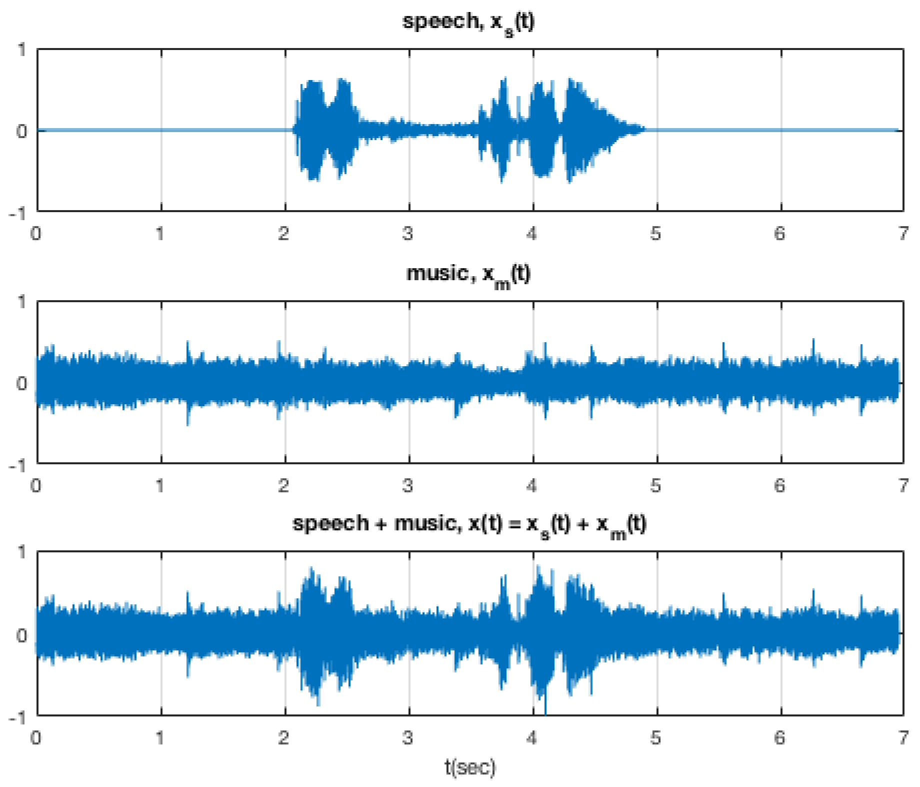


Define New Input Signals for Ducker

```
[xs, fs] = audioread('speech.wav');  % speech signal
[xm, fs] = audioread('music.wav');   % music signal
xms = xm + xs;                       % speech plus music
t = (1:length(xs))/fs;               % time in sec
```

Plot New Input Signals

```
close all
subplot(3,1,1)
plot(t,xs),title('speech, x_s(t)'),grid on
subplot(3,1,2)
plot(t,xm),title('music, x_m(t)'),grid on
subplot(3,1,3)
plot(t,xms),xlabel('t(sec)'),
title('speech + music, x(t) = x_s(t) + x_m(t)'),grid on
```



Appendix A.13: Ducker Design

```
Aa = 0.997916; % lambda_a
Ar = 0.999826; % lambda_r
p = 1/5;      % rho
c0 = 0.006428;       % threshold
g1 = 0;     % for EMA gain-smoothing filter, g_n
g0 = 0;             % g_{n-1}
c = 0;           % Control signal
G = 0;           % Smoothed gain
n = 1;               % Index variable
N = length(xs);     % For limit on n

for n = 1:N
if abs(xs(n)) >= c                    % calculate control signal
    c = Aa*c + (1 - Aa)*abs(xs(n));
else
    c = Ar*c + (1 - Ar)*abs(xs(n));
end

cPlot(n) = c;     % For plotting

g0 = g1;

if c == 0         % Calculate gain signal
```

```
    g1 = 1;
elseif c >= c0
    g1 = (c/c0)^(p-1);
else
    g1 = 1;
end

gPlot(n) = g1;     % For plotting

if g1 >= g0      % EMA smoothing filter
    G = Aa*G + (1 - Aa)*g1;
else
    G = Ar*G + (1 - Ar)*g1;
end

GPlot(n) = G;     % For plotting

ym(n) = G*xm(n);

y(n) = xs(n) + ym(n);
end
```

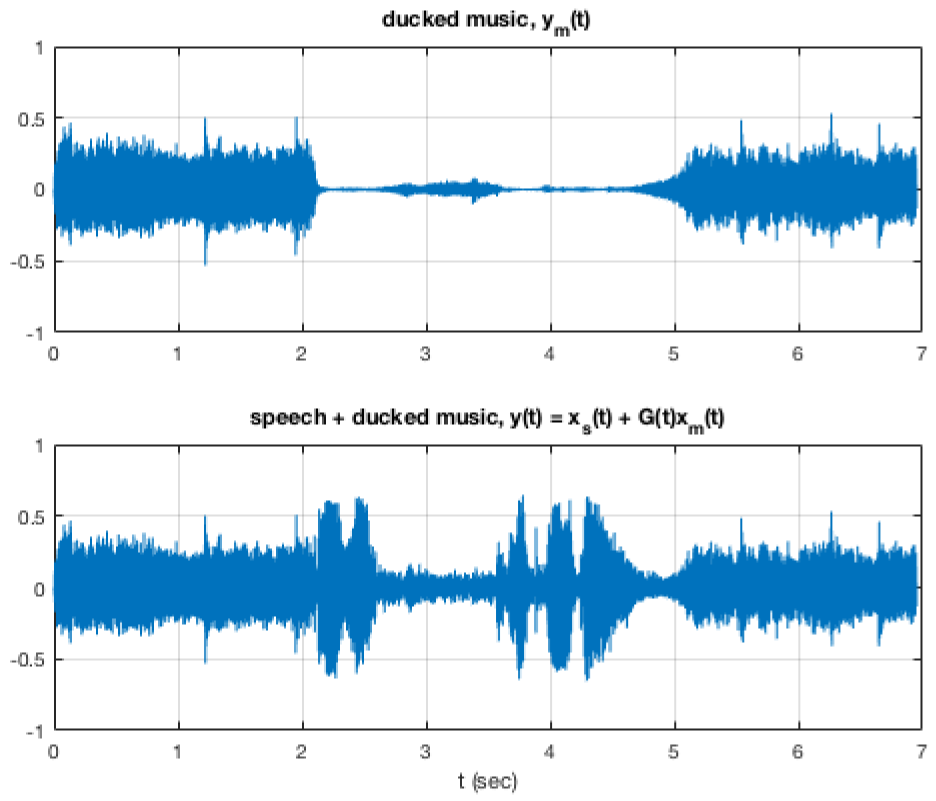
Compare Results

```
sound(xms,fs)

sound(y,fs)

audiowrite('speech with ducked music.wav',y,fs)

xsmax = max(xs)
```

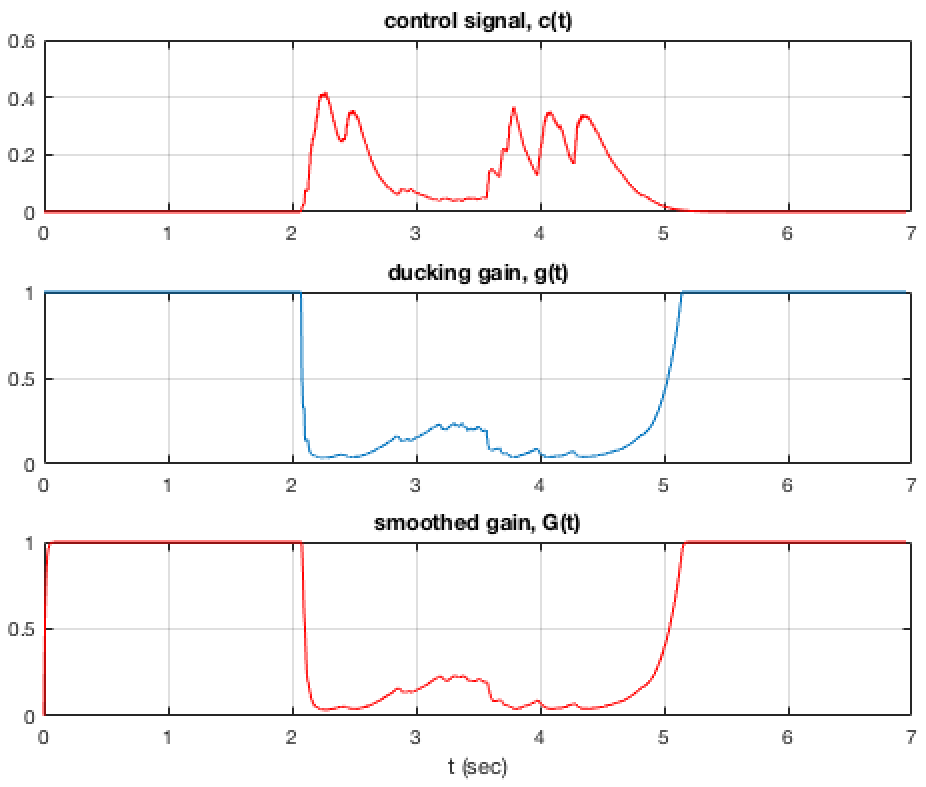
```
xsmax =

    0.6428
```

Plot Ducker Results

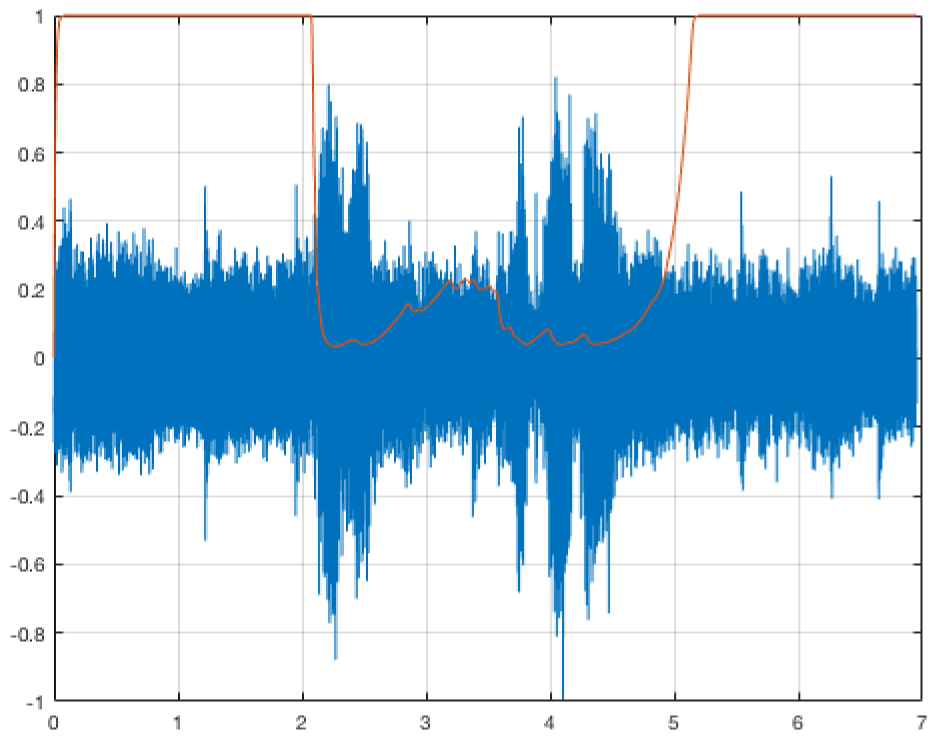
```
close all
subplot(2,1,1)
plot(t,ym),title('ducked music, y_m(t)'),axis([0,7,-1,1]),grid on
subplot(2,1,2)
plot(t,y),title('speech + ducked music, y(t) = x_s(t) + G(t)x_m(t)'),
axis([0,7,-1,1]),grid on,xlabel('t (sec)')
```



```
close all
subplot(3,1,1)
plot(t,cPlot,'r')
title('control signal, c(t)'),
grid on

subplot(3,1,2)
plot(t,gPlot),title('ducking gain, g(t)')
grid on

subplot(3,1,3)
plot(t,GPlot,'r')
xlabel('t (sec)'),title('smoothed gain, G(t)')
grid on
```



```
close all
plot(t,xms,t,GPlot),grid on
```



Published with MATLAB® R2017b