Lab 04 - Distortion and Up-sampling

Ron Guglielmone

INTRODUCTION

A distortion plug-in has been designed with the following signal-flow block diagram:

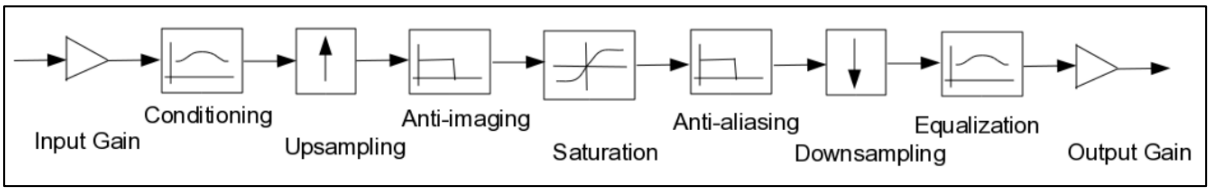

Figure 1, processing chain for distortion plug-in.

The saturation non-linearity can be implemented in two different ways: via hard clipping, or soft clipping (described by the following two equations).

Hard Clip

$$\xi(x) = \begin{cases} 1, & x > 1, \\ x, & |x| \leq 1, \\ -1, & x < -1, \end{cases}$$

Soft Clip

$$\xi(x) = \frac{x}{1 + |x|}.$$

PROBLEM 1(a)

Figure 2 shows a spectrogram of the provided sweep signal before it has been passed through the stock version of our distortion plug-in.

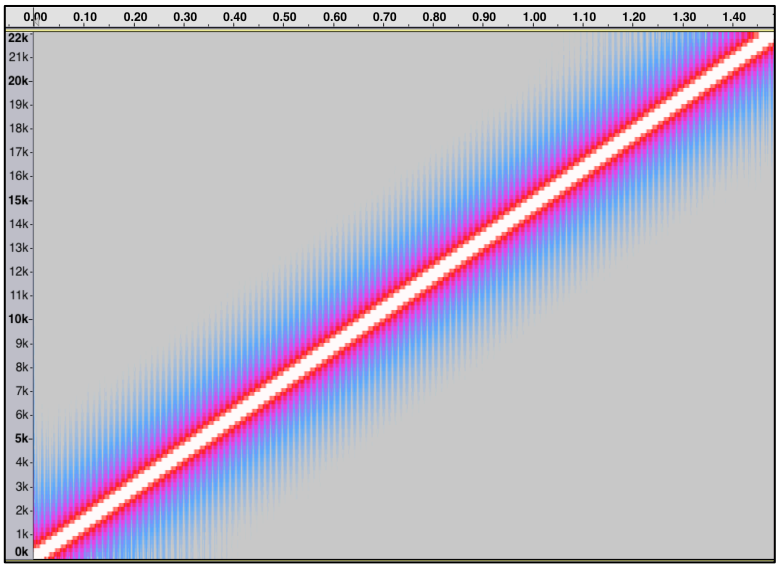

Figure 2, spectrogram of original signal.

After processing, aliasing is both audible in the sound and visible in the spectrogram (Figure 3). Without the plug-in, we only see and hear one ascending frequency sweep, but after processing, we can see and hear a series of other sweeps. The additional ascending sweeps are desirable harmonics, but the descending sweeps are due to aliasing.

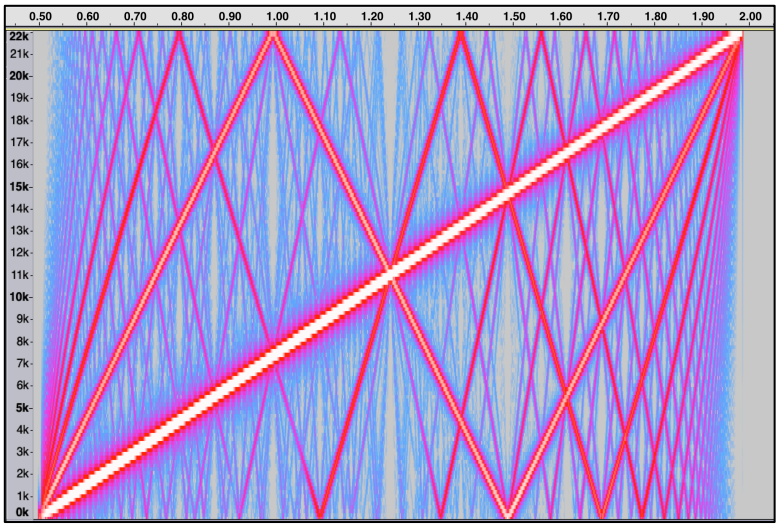

Figure 3, spectrogram of stock plug-in with heavy aliasing.

PROBLEM 1(b)

The following MATLAB script was written to help develop a new set of filter coefficients:

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford University
% May 3, 2017
%
% Problem 1 Part B

% Filter parameters:
order = 6;
ripple = 2;
stopAtten = 100;
omega = 1/8 - 2/32;

% Design filter:
[b,a] = ellip(order, ripple, stopAtten, omega);

% Look at mag-spec
fvtool(b,a);

% Transform to bi-quads
s = tf2sos(b,a);
```

After lots of unfruitful experimentation, a filter with the following magnitude spectrum was settled upon (Figure 4). This is a sixth order elliptic filter.

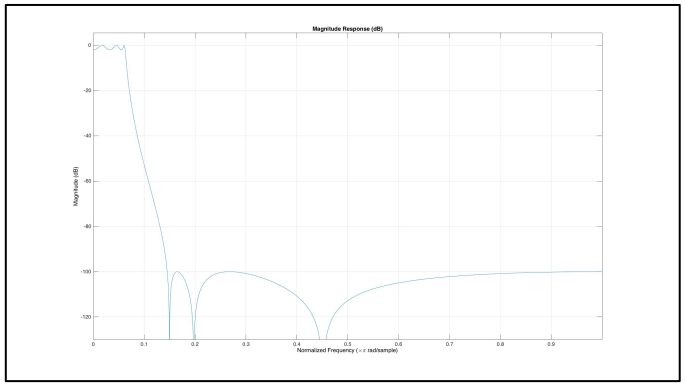

Figure 4, magnitude spectrum of elliptic filter.

While aliasing was reduced significantly in terms of audio perception, the spectrogram still reveals quite a bit of undesired frequency mirroring (Figure 5).

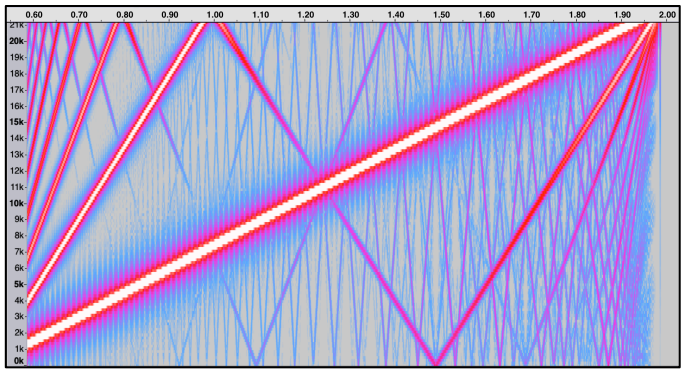

Figure 5, aliasing from elliptic, 6th order, -100 dB filter.

I would have continued trying to improve this, but higher order filters consistently 'broke' the plugin, as did cutoffs at lower frequencies. A quick analysis of Apple's AUDistortion plug-in revealed that even professional products have significant aliasing (Figure 6).

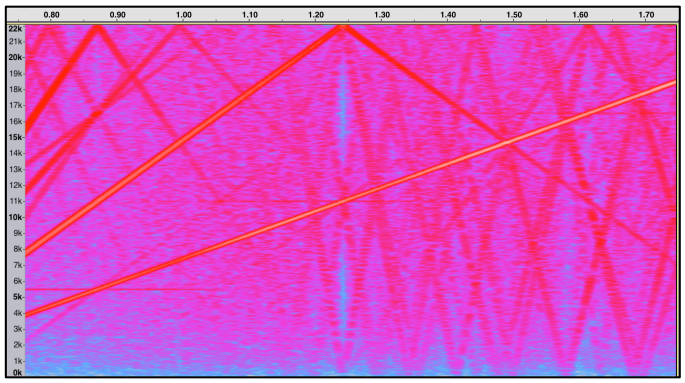

Figure 6, spectrogram of Apple distortion (aliased).

In terms of C++ code modifications, a partial list of the different coefficients tested is presented below in Figure 7. Most of these did not work, either resulting in clipped output, or no output at all.

```
// First try coefficients (not enough):
// {0.0005,    0.0008,    0.0005,    -1.6773,    0.7417},
// {1.0000,    0.5781,    1.0000,    -1.5078,    0.8276},
// {1.0000,    0.0133,    1.0000,    -1.3879,    0.9411},

// Doesn't work:
//{0.0000,    0.0000,    0.0000,    -1.8912,    0.8996},
//{1.0000,    1.1646,    1.0000,    -1.8670,    0.9157},
//{1.0000,    0.5197,    1.0000,    -1.8392,    0.9442},
//{1.0000,    0.2053,    1.0000,    -1.8348,    0.9804},

// Doesn't work:
//{0.0000,    0.0000,    0.0000,     1.8766,    0.8876},
//{1.0000,   -1.2484,    1.0000,    -1.8575,    0.9159},
//{1.0000,   -1.5865,    1.0000,    -1.8391,    0.9527},
//{1.0000,   -1.6724,    1.0000,    -1.8385,    0.9852},

// Doesn't work (only slightly lower cutoff than abv):
//{0.0000,   -0.0000,    0.0000,    -1.9266,    0.9307},
//{1.0000,   -1.6278,    1.0000,    -1.9312,    0.9523},
//{1.0000,   -1.7831,    1.0000,    -1.9465,    0.9835},

// Didn't work:
//{0.0000,   -0.0000,    0.0000,   -1.7333,    0.7523},
//{1.0000,   -1.5340,    1.0000,   -1.7993,    0.8183},
//{1.0000,   -1.7360,    1.0000,   -1.9122,    0.9317},

// Didn't work:
//{1.0000,   -0.0160,    1.0000,   -1.7333,    0.7523},
//{1.0000,   -1.5340,    1.0000,   -1.7993,    0.8183},
//{1.0000,   -1.7360,    1.0000,   -1.9122,    0.9317},

// Didn't work.
//{1.0000,    0.8768,    1.0000,   -1.5856,    0.6312},
//{1.0000,   -0.9781,    1.0000,   -1.6778,    0.7243},
//{1.0000,   -1.3787,    1.0000,   -1.8435,    0.8929},

// Doesn't work...
//{0.0000,    0.0000,    0.0000,    -1.5887,    0.6326},
//{1.0000,   -1.2960,    1.0000,    -1.6506,    0.6938},
//{1.0000,   -1.6518,    1.0000,    -1.7551,    0.7978},
//{1.0000,   -1.7435,    1.0000,    -1.8838,    0.9274},

// Best I could do after 20 or so tries:
{0.0001,    0.0001,    0.0001,    -1.8503,    0.8662},
{1.0000,   -0.8335,    1.0000,    -1.8267,    0.9080},
{1.0000,   -1.2518,    1.0000,    -1.8249,    0.9680},
```

Figure 7, different test coefficients for the filter matrix.

PROBLEM 1(c)

A parametric section has been designed from the following transfer function:

$$H(s) = \frac{\left(\frac{s}{\omega_0}\right)^2 + \frac{1}{Q} \cdot \gamma \left(\frac{s}{\omega_0}\right) + 1}{\left(\frac{s}{\omega_0}\right)^2 + \frac{1}{Q} \left(\frac{s}{\omega_0}\right) + 1}$$

The bilinear transform was applied, resulting in the following coefficients:

$$b_0 = 1,\ b_1 = \frac{\gamma}{2Q\pi w},\ b_2 = \frac{1}{4Q\pi^2 w^2}$$

$$a_0 = 1,\ a_1 = \frac{1}{2Q\pi w},\ a_2 = \frac{1}{4\pi^2 w^2}$$

This resulted in the following C++ code segments (Figure 8).

```
////////////// BILINEAR /////////////
///////////////START/////////////////

double T = 1/fs;

az0 = ( a0*T*T + 2*a1*T + 4*a2 );
az1 = ( 2*a0*T*T - 8*a2 ) / az0;
az2 = ( a0*T*T - 2*a1*T + 4*a2 ) / az0;

bz0 = ( b0*T*T + 2*b1*T + 4*b2 ) / az0;
bz1 = ( 2*b0*T*T - 8*b2 ) / az0;
bz2 = ( b0*T*T - 2*b1*T + 4*b2 ) / az0;

az0 = 1;
///////////////END/////////////////
```

```
////////////// FILTER ///////////////
///////////////START/////////////////

double P2C2 = (pi*pi)*(center*center);
double PC = pi*center;

b0 = 1.0;
b1 = dB2mag(gain)/(qval*2*PC);
b2 = 1/(4*P2C2);
a0 = 1.0;
a1 = 1/(qval*2*PC);
a2 = 1/(4*P2C2);
///////////////END/////////////////
```

Figure 8, code segments for bilinear transform and parametric filter.

PROBLEM 2(a)

A non-linearity described by the following equation was implemented:

$$\xi(x) = \frac{x + \gamma}{1 + |x + \gamma|}$$

The following MATLAB script (Figure 9) was written to test its effects on a simple signal.

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford
University
% May 6, 2017
%
% Problem 2 Part A

% Constants:
freq = 440;
fs = 44100;
dur = 5;
T = 1/fs;
t = 0:T:dur;
gamma = 0;

% Signal:
x = cos(2*pi*440*t);

% Non-Linearity:
x = (x + gamma)./(1 + abs(x + gamma));

% Plot spectrum:
X = fftshift(fft(x));
N = length(t);
dF = fs/N;
f = -fs/2:dF:fs/2-dF;
plot(f,abs(X)/N);
% sound(x,fs)
```

Figure 9, MATLAB script for problem 2.

With gamma set equal to zero, the following output spectrum was seen (Figure 10).

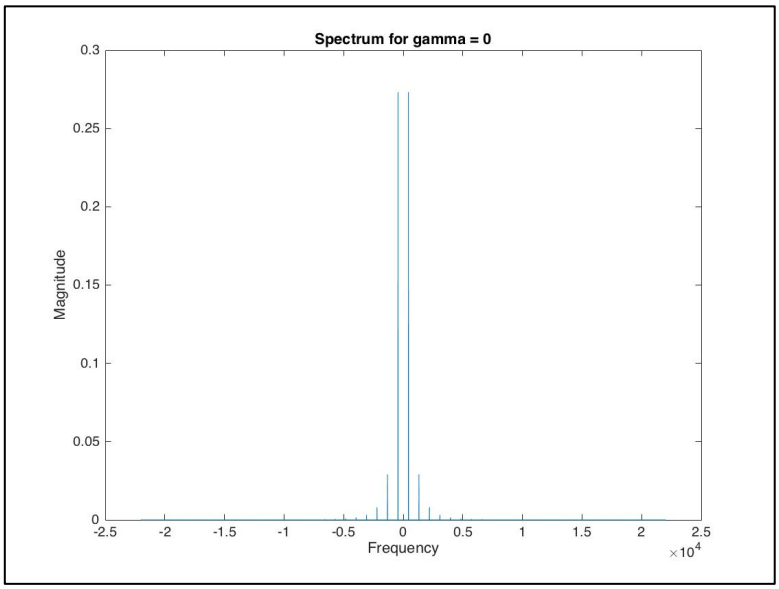

Figure 10, spectrum for gamma = 0.

With gamma set equal to one, this alternative spectrum resulted (Figure 11).

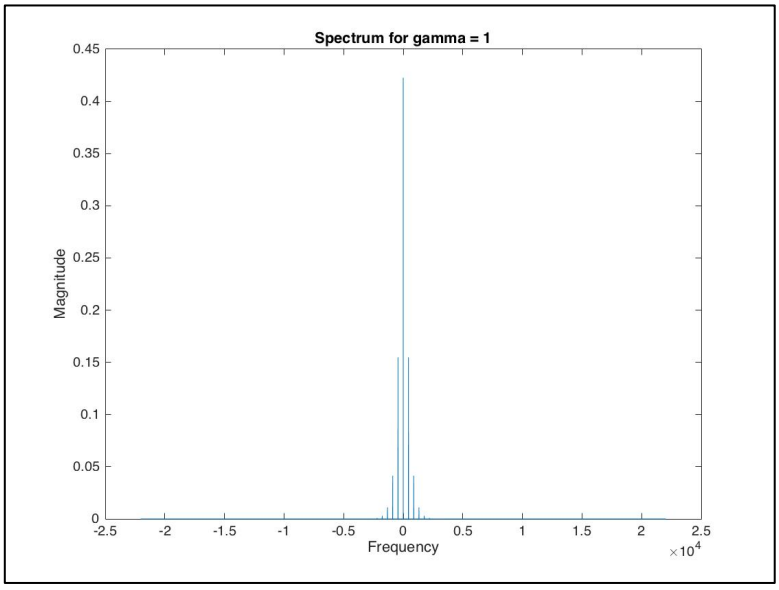

Figure 11, spectrum for gamma = 1.

The spectrum seems condensed for gamma equal to 1. We gain a DC component at the expense of our previous base-bands.

PROBLEM 2(b)

A DC blocking filter was designed from the following transfer function:

$$H(s) = \frac{s}{s + \omega_c}$$

This filter can be implemented at output using the following difference equation:

$$y[n] = x[n] - x[n\text{-}1] + a * y[n\text{-}1]$$

Values of ***a*** around 0.999 work best for effectively blocking DC without affecting higher frequencies. An example impulse response with ***a*** = 0.999 is shown below (Figure 12).

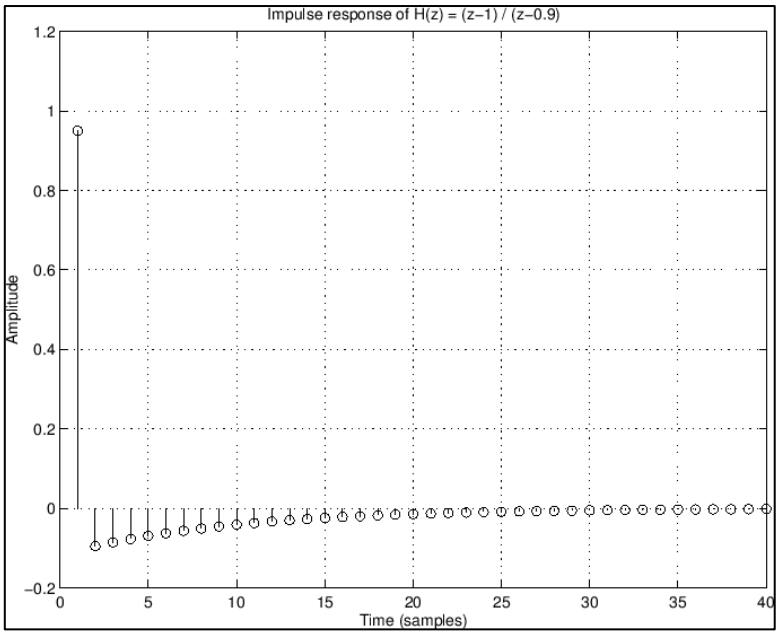

Figure 12, impulse response from JOS' online book.