Homework #4: Distortion and Upsampling [30 points]
Due Date: May 2, 2019

Submission Instructions

Submit via Canvas. Create **a single compressed file** (`.zip`, `.tar` or `.tar.gz`) containing all your submitted files. Upload the compressed file to the homework submission dropbox. Name the file using the following convention:

```
<suid>_hw<number>.zip
```

where `<suid>` is your Stanford username and `<number>` is the homework number. For example, for Homework #1 my own submission would be named `cavdir_hw1.zip`.

For coding problems (either C++ or Matlab code), submit all the files necessary to compile/run your code, including instructions on how to do it. In case of theory problems, submit the solutions in **PDF format only**. LaTeX or other equation editors are preferred, but scans are also accepted. In case of scanned handwriting, make sure the scan is legible. Illegible homework will not be graded.

Lab4 - Distortion and Upsampling

Problem 1. [20 Points]

In this lab, the plug-in *Distortion* will be explored. The plug-in is posted to the class web site; look for `Lab4_MAC_VST.zip` or similar. Solutions for each exercise should include the source files `Distortion.cpp` and `Distortion.h`, suitably modified. The files must be able to be compiled. Additional write-up is required for some sections of this problem.

The plug-in processing, shown in Figure 1, is a cascade of an input gain and filter, upsampling, distortion, downsampling, and an output gain and filter. The distortion element clips its input to lie within the interval $[-1, 1]$, either with a hard clip,

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

or with a soft saturation,

$$\xi(x) = \frac{x}{1 + |x|}. \tag{2}$$

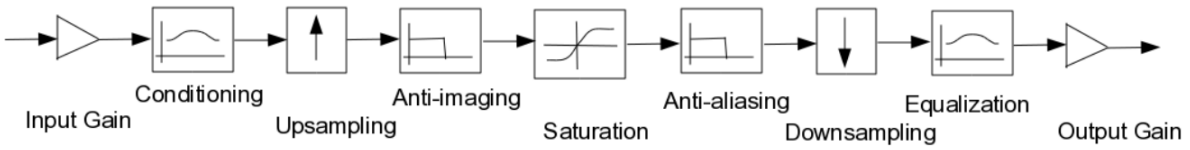

Figure 1: Processing chain

By applying parametric sections before and after the distortion element, various guitar amp-like distortion sounds can be made.

Approximating the distortion element as a Taylor series in x, and noting that a signal $x(t)$ raised to the nth power has a spectrum n times the bandwidth of the original, we see that the bandwidth of any input may be greatly multiplied by the distortion element. For this reason, the input signal is upsampled by a factor of eight by inserting seven zeros between each input sample, and then low-pass filtered to 1/8 of its new bandwidth. The 1/8-bandwith signal is distorted and low-pass filtered again to 1/8 of its bandwidth. One of every eight of these samples is output, the rest discarded.

1(a). [4 Points] As presently implemented, the low-pass filter is a first-order Butterworth with cutoff frequency of 1/8, which is insufficient to eliminate aliasing. Also, as presently implemented, the input and output filters behave as digital wires. Verify the presence of aliasing by setting the input gain to 6.0 dB and the output gain to -6.0 dB, and running the provided linear sine sweep (`linearSweep.wav` inside `audioTracks.zip`) through the plug-in. Plot a spectrogram of the output[1], and also listen to the output. Turn in your spectrogram and briefly explain what you heard that shouldn't be there if there were no aliasing.

1(b). [10 Points] Now design an improved antialiasing filter by using one of the Matlab functions `butter`, `cheby2`, or `ellip`. Adjust the cutoff frequency, filter order (although don't go higher than order 12), and other parameters so that the aliasing is barely noticeable when listening. You will need to convert the filter coefficients returned by the design function to biquads (sometimes called second-order sections); see functions such as `tf2sos` and `zp2sos`. Turn in your filter coefficients along with a plot of the magnitude of your antialiasing filter.

Do you notice any difference in aliasing in the output when the hard clip is used compared to the soft clip? Why?

Aside from aliasing, do you notice much difference in the sound of the output when the hard clip is used compared to the soft clip?

[1]You may produce the spectrogram however you like. We've provided a matlab script you could use called `ftgram` which displays a spectrogram; the matlab function call is something like `ftgram(signal, fs, 'music');`.

1(c). [**6 Points**] Implement a parametric section to compute the coefficients of the input and output filters from their center frequency, Q, and gain controls.

Turn in single project (only the `.h` and `.cpp` files) including all the implementations asked in 1(a)., 1(b). and 1(c)..

1(d). [**optional (just for fun)**]

- We have posted a couple guitar tracks on the class web site (see `lab1`); see if you can't find some filter settings which make a nice distorted guitar sound.

- Try with a nonlinearity of the form

$$\xi(x) = \frac{x}{1+|x|} + \gamma \sin(\pi x) \qquad 0 \leq \gamma \leq 1.0$$

  Play with different values of γ

- Add a compressor before the distortion (you don't need to code this: using your VST host apply your compressor from lab1 in first and then apply the distortion you just implemented). Before you listen to the results, how do you think this will affect the generated sound? Now listen to it. Is it what you expected?

- Change the nonlinearity to create a polynomial approximating a *dead-zone* nonlinearity. What does it sound like? The dead-zone can be defined as:

$$\xi(x) = \begin{cases} 0, & |x| \leq \alpha, \\ x - \alpha, & x > \alpha, \\ x + \alpha, & x < -\alpha, \end{cases} \tag{3}$$

  with a dead-zone width $2\alpha \geq 0$.

Problem 2. [**10 points**]

In this problem, we will explore the harmonic content created by memoryless nonlinearities.

A memoryless nonlinearity with odd symmetry can produce only odd harmonics of an input sinusoid. This can be seen by noting that the presence of even harmonics will create an output waveform which is not symmetric about zero, since positive-going regions of even harmonics will coincide with both positive-going and negative-going regions of the original waveform.

If we add a DC offset to the input of the saturating nonlinearity of Eq. (2) from Problem 1, it no longer maintains its odd symmetry. We therefore expect it to produce both even and odd harmonics.

2(a). [**4 points**] Implement the asymmetric saturator,

$$\xi(x) = \frac{x+\gamma}{1+|x+\gamma|}. \tag{4}$$

where γ is the DC offset applied to the input of the saturator. In Matlab, generate a unity-amplitude sinusoid at 440 Hz. Distort the sinusoid using the asymmetric saturator above. Turn in a plot showing the dB magnitudes of the two output spectra, formed using a DFT size of 16384 points and a Blackman window. Describe the differences in the spectra for $\gamma = 0$ and $\gamma = 1$.

In addition, briefly explain what happens if we add a DC offset to the *output* of the nonlinearity.

2(b). [**6 points**] The DC offset will produce an output with a DC component. Because of the nonlinearity, the magnitude of this component will depend upon the input signal. We therefore need to provide a *DC blocking filter* at the output of the nonlinearity. Design a second-order digital DC blocking filter by cascading two first-order highpass filters of the form

$$H(s) = \frac{s}{s+\omega_c}, \tag{5}$$

where $\omega_c = 2\pi(5\text{Hz})$ is the cutoff frequency of the DC blocking filter. Perform a bilinear transform on this highpass filter (at the oversampled rate!) to obtain your digital DC blocking filter. Plot the impulse response of this filter. Note the *settling time* of the filter, *i.e.* the time it takes for the impulse response to decay by 60 dB.

Implement the DC blocking filter as part of your VST plugin and apply the 440 Hz input sine wave. Again find the output spectra using $\gamma = 0$ and $\gamma = 1$. Confirm that the DC blocking filter removes the DC component for $\gamma = 1$. Note that you will need to apply the input for a sufficient time so that the DC blocking filter can settle before beginning the DFT window.

Turn in a separate project (only the `.h` and `.cpp` files) for Problem 2.