Homework #1: Basic Compression [30 points]
Due Date: April 10, 2014

Submission Instructions

Submit via coursework's DropBox. Create **a single compressed file** (`.zip`, `.tar` or `.tar.gz`) containing all your submitted files. Place the compressed file at the top level of your Drop Box. 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 `fgermain_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.

Problem 1. [10 points]

In this lab you will modify a basic compressor and experiment with the level detector. The file `Lab1_<OS>_VST.zip` contains code for a basic compressor—a feed-forward compressor with a peak detector and a gain computer configured as a limiter. The code implements several sliders,

- Input Gain, Output Gain and Threshold,

which are used to control the input and output levels and threshold of compression for the basic compressor, and

- Attack Time and Release Time,

which define the time constants in the peak detector. Two additional sliders are present, but not used in the basic compressor,

- Detector Exponent and Compression Ratio.

You will use these sliders to develop more sophisticated compressors later in this lab and in the next lab.

1(a). [**0 point**] Configure your development environment and digital audio workstation, perhaps with help from Jorge or fellow students. Compile the basic compressor, and listen to its effect on the audio tracks provided, a short drum track, guitar track and a test signal called `tdiff1.wav`. To give you confidence that your plug-in is working as intended, the plug-in `BasicLimiter` is a running version of the basic compressor.

Note: due to licensing restrictions we cannot distribute the VST SDK, so the code won't compile out of the box. You'll need to get the VST SDK 2.4 from Steinberg's website (`http://www.steinberg.net/nc/en/company/developer/sdk_download_portal.html`) and place the needed files inside the project's `vst_sdk` folder (`Readme.txt` files are provided inside that folder with more details).

Solution: There is no solution really. The idea is to make sure the student has a functional development environment.

1(b). [**6 points**] The test signal `tdiff1.wav` contains a 1.0 kHz sine wave with a stepped amplitude, starting out at a level of 0.09 for one second, jumping (at a zero crossing of the sine wave) to a level of 0.9 for one second, and then back to a level of 0.09 for one second.

Set the Input Gain and Threshold of compression (which is defined with respect to the input signal, scaled by the input gain) so that the initial level of the test signal, 0.09, is at the threshold of compression. Set the Attack Time to 10 ms and the Release Time to 200 ms. Capture the output of the basic compressor in response to the test signal, and make two Matlab plots,

- one showing the output signal around the time of the attack, with the attack time labeled, and
- one showing the output signal around the time of the release, with the release time labeled.

Make a rough sketch showing the envelope of the input signal and the detected level as a function of time for the test signal input.

Finally, set the Attack Time to 1 ms and the Release Time to 50 ms, and make another pair of plots showing the attack and release.

Solution: For the given signal, and keeping the input gain at 0 dB the threshold should be set around -20.9 dB. Setting the attack time to 10 ms and the release time to 200 ms we obtain the plot show in Figure 1.

Figure 2 shows the response for shorter attack and release times.

1(c). [**4 points**] Using the drum signal as your input, setting the Release Time to 200 ms, the input gain at +3dB, the output gain at -3dB and the threshold at -12dB. Adjust the Attack Time from 0.1 ms to 100 ms, and listen to the result. At what values of attack time does the compressor make the drums sound

- smooth,

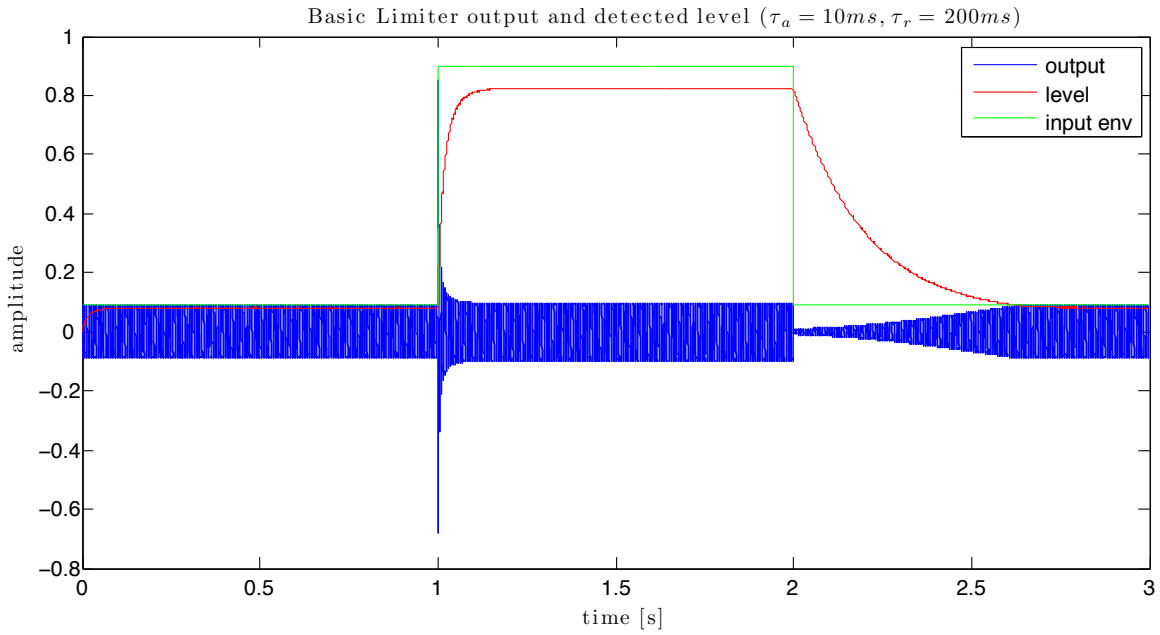

Figure 1: Limiter output, detected level and input envelope for threshold = -20.9 dB, $\tau_a = 10$ ms and $\tau_r = 200$ ms

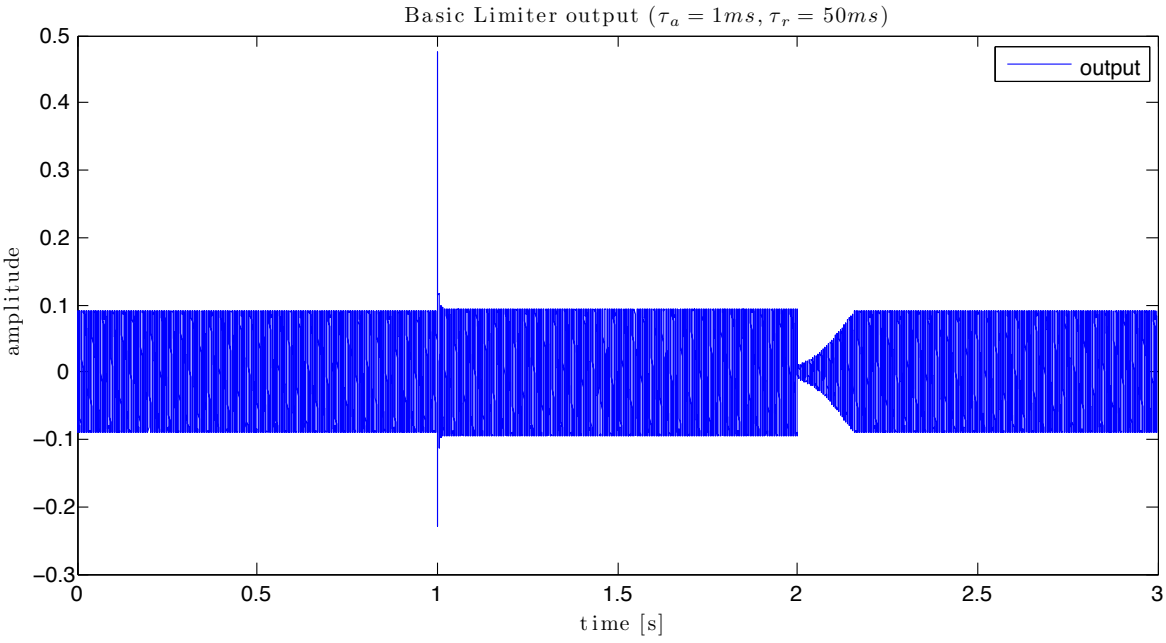

Figure 2: Limiter output for threshold = -20.9 dB, $\tau_a = 1$ ms and $\tau_r = 50$ ms

- tinky,
- thuddy,
- transparent?

Now, set the Attack Time to 0.1 ms, and adjust the Release time from 20 ms to 200 ms. At what values of release time does the compressor make the drums sound

- buzzy,
- roomy,
- even (each hit having equal loudness)?

Note that references for these values can be found in the Course Notes, but the idea is for you to evaluate your own compressor. Maybe your perception differs from the Course Notes authors!

Solution: There is no exact solution. The values in the course notes serve as reference. Reported values should be "in the ballpark". If not, double check your code/system to make sure everything works correctly.

Problem 2. [9 points]

Modify the basic compressor by replacing the peak detector with an RMS detector. Use the Release Time slider to set the time constant of the leaky integrator. (The Attack Time slider setting will be unused.) Plot the output of the RMS compressor in response to the `tdiff1.wav` test signal with the Release Time set to 200 ms. How does this output envelope compare to that of the attack/release basic compressor? Very briefly, how does this compressor sound on drums or guitar compared to the basic compressor when the attack time of the basic compressor is set to 0.1 ms, and the release time of the RMS compressor and basic compressor are the same? (Note that you might want to use the `BasicLimiter` plug-in for comparison to the peak detector case.)

2(a). [6 points] Submit your code and a plot of the output of the RMS compressor in response to the `tdiff1.wav` test signal with the Release Time set to 200 ms.

Solution: This is a possible implementation of the `RMSDetector` class:

```
// RMS detector
class RMSDetector {

protected:
    float b0, a1, levelEstimate;

public:
    RMSDetector() {

        // default pass through
        this->a1 = 0;
```

```
        this->b0 = 1;
        reset();
    }

    void setTau(float tau, float fs) {
        a1 = exp( -1 / ( tau * fs ) );
        b0 = 1 - a1;
    }

    void reset() {
        // reset filter state
        levelEstimate=0;
    }

    void process (float input, float& output) {
        levelEstimate += b0 * (input * input - levelEstimate);
        output = sqrtf(levelEstimate);
    }
};
```

Figure 3 shows the response for the RMS limiter

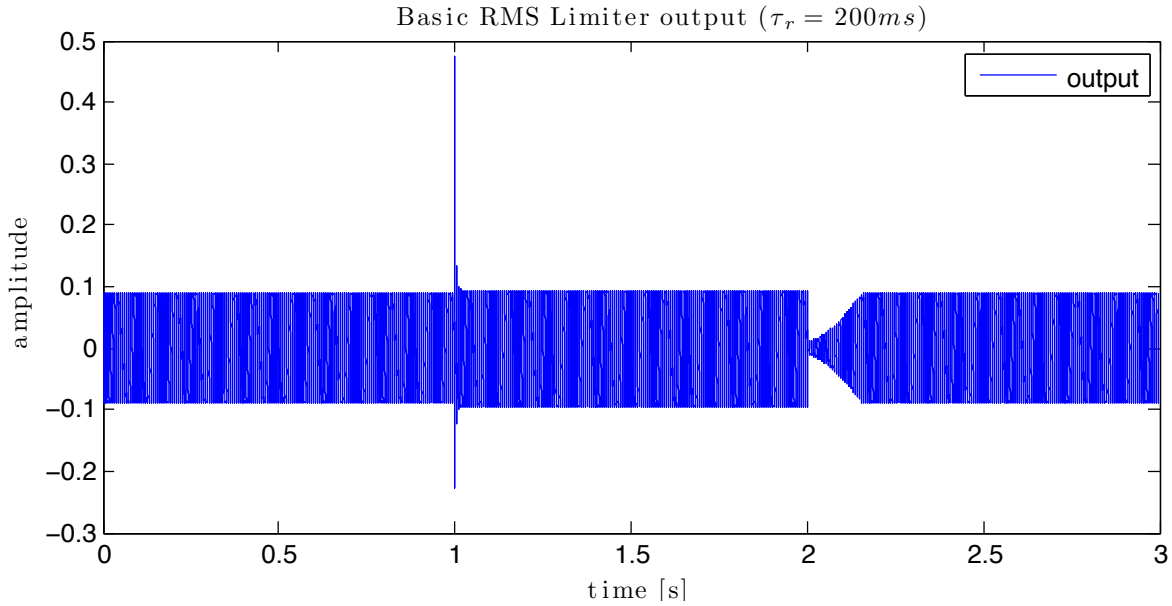

Figure 3: Limiter output for threshold = -20.9 dB and τ_r = 200 ms

2(b). [3 points] Answer the following questions:

- How does this output envelope compare to that of the attack/release basic compressor?

- Very briefly, how does this compressor sound on drums of guitar compared to the basic compressor when the attack time of the basic compressor is set to 0.1 ms, and

the release time of the RMS compressor and basic compressor are the same? (Note that you might want to use the `basiclimiter` plug-in for comparison. to the peak detector case.)

Solution:

If you set the basic compressor so that the attack and release time are identical, and set the RMS leaky integrator time at that same value as well, you should observe that the RMS detector has (1) a faster attack and (2) a slower release than the basic compressor. You should also observe that gain computer is higher for the RMS detector compared to the basic compressor, meaning that after the attack the level of the output signal will be higher for the RMS compressor. This makes sense as we know the peak level for a sinusoid **in steady state** will be higher that its RMS level.

For the second question, many answers are acceptable. One thing that is interesting to observe is what we mentioned in class regarding the smoother gain computer during release for the RMS detector. It should result in a decrease of the audible distortion introduced by the compressor.

Problem 3. [11 points]

Now, modify the RMS detector of the previous problem to implement a running p-norm of the input. In other words, replace your RMS detector with a detector that forms the pth root of the smoothed pth power of the absolute value of the input signal. Here, the smoothing is a leaky integrator described in the notes, having transfer function

$$H(z) = \frac{1-a}{1-az^{-1}}, \tag{1}$$

where $a \in [0, 1)$ is determined by the Release Time slider value. Use the Exponent slider to determine the value for $p \in [1, 10]$.

3(a). [5 points] Submit your modified code, and plots of the response of your modified compressor to the test signal for three cases, $p = [2, 5, 10]$, for a Release Time of 100 ms.

Solution: This is a possible implementation of the `RMpDetector` class:

```
// RMp detector
class RMpDetector {

protected:
    float b0, a1, levelEstimate, p;

public:
    RMpDetector() {

        // default pass through
        this->a1 = 0;
        this->b0 = 1;
        reset();
    }
```

```
    void setTau(float tau, float fs) {
        a1 = exp( -1 / ( tau * fs ) );
        b0 = 1 - a1;
    }

    void setP(float p) {
        this->p = p;
    }

    void reset() {
        // reset filter state
        levelEstimate=0;
    }

    void process (float input, float& output) {
        levelEstimate += b0 * (powf(input,p) - levelEstimate);
        output = powf(levelEstimate, 1.0/p);
    }
};
```

Figure 4 show the response for $p = [2, 5, 10]$.

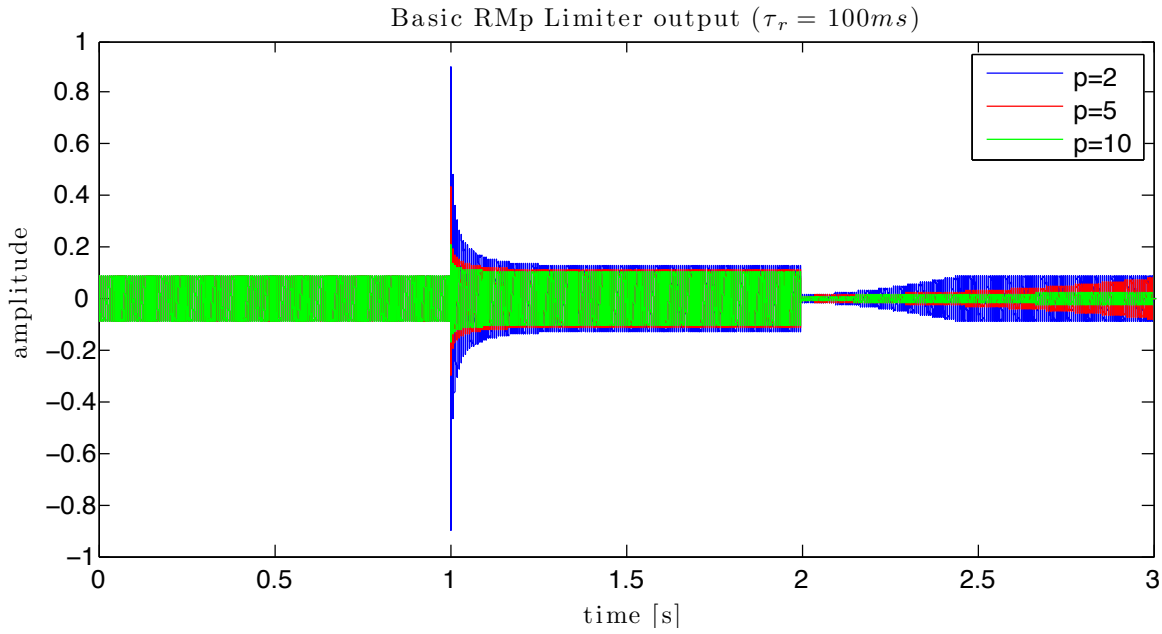

Figure 4: Limiter output with RMp detector, for threshold = -20.9 dB, $\tau_r = 200$ ms and $p = [2, 5, 10]$

3(b). [6 points] Note the similarity between this "RMp" compressor and a compressor with a peak detector having separate attack time and release time controls. With larger values of p, the attack becomes quicker and the release becomes slower. In this way, a compressor with separate attack and release time controls can be made using a linear

time-invariant filter and memoryless nonlinear elements. The advantage is that the inner loop of the process doesn't have any conditional statements, slowing the computation. The disadvantage is that outside a limited range of exponents p (which controls the ratio of release time to attack time), there are numerical difficulties.

Let's define the time constant of the leaky integrator, τ, as the time taken for the leaky integrator to reach $1 - 1/e$ of its target value. More precisely, if the input to the leaky integrator was, for a long time, at a level μ so that the state of the leaky integrator $\lambda(n)$ was also μ, and at a given point in time, the level of the leaky integrator input became ν, then the time constant of the leaky integrator is the time taken for the leaky integrator output $\lambda(n)$, starting at μ to reach

$$\lambda(n) = \mu + (1 - 1/e)(\nu - \mu). \tag{2}$$

For a given exponent p and leaky integrator time constant τ, derive expressions for the attack and release time constants of the RMp detector.

Solution:

The idea here was to transform the "RMp" detector into an equivalent Peak detector and then find the attack and release time constants (τ_a and τ_r). The detected level can be expressed as follows:

$$\lambda[n]^p = y[n] = (1 - a)|x[n]|^p + ay[n - 1] \tag{3}$$

Note that $a = e^{-1/(f_s\tau)}$, where τ is the time constant of the one pole filter and f_s is the sampling rate. To compute the time constants, we need to analyze the two cases (attack and release) separately, using a step input.

Attack:

Let

$$x[n] = u[n] = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$$

Then

$$y[n] = \begin{cases} 0 & n < 0 \\ (1 - a) + ay[n - 1] & n \geq 0 \end{cases}$$
$$= \begin{cases} 0 & n < 0 \\ 1 - a^{n+1} & n \geq 0 \end{cases}$$

Thus, for $n \geq 0$ we have:

$$\lambda[n] = y[n]^{1/p} = \left(1 - a^{n+1}\right)^{1/p}$$

To find the attack time constant (τ_a) we need to solve:

$$\begin{aligned}\lambda[\tau_a f_s] = \big(1 - e^{-1}\big) &= \big(1 - a^{\tau_a f_s + 1}\big)^{1/p} \\ &= \Big(1 - e^{\frac{-1}{\tau}(\tau_a + \frac{1}{f_s})}\Big)^{1/p}\end{aligned}$$

which leads to

$$\tau_a = -\tau \ln\big(1 - (1 - e^{-1})^p\big) - \frac{1}{f_s} \tag{4}$$

Release:

Similarly, for the release case we let

$$x[n] = 1 - u[n] = \begin{cases} 1 & n < 0 \\ 0 & n \geq 0 \end{cases}$$

Then

$$\begin{aligned} y[n] &= \begin{cases} 1 & n < 0 \\ ay[n-1] & n \geq 0 \end{cases} \\ &= \begin{cases} 1 & n < 0 \\ a^{n+1} & n \geq 0 \end{cases}\end{aligned}$$

Thus, for $n \geq 0$ we have:

$$\lambda[n] = y[n]^{1/p} = a^{\frac{n+1}{p}} = e^{\frac{-(n+1)}{p f_s \tau}}$$

To find the release time constant (τ_r) we need to solve:

$$\begin{aligned}\lambda[\tau_r f_s] = e^{-1} &= a^{\frac{\tau_r f_s + 1}{p}} \\ &= e^{\frac{-(\tau_r f_s + 1)}{p f_s \tau}}\end{aligned}$$

which leads to

$$\tau_r = p\tau - \frac{1}{f_s} \tag{5}$$

Note that as p increases, τ_a decreases and τ_r increases. Figure 5 shows how these time constants change as a function of p.

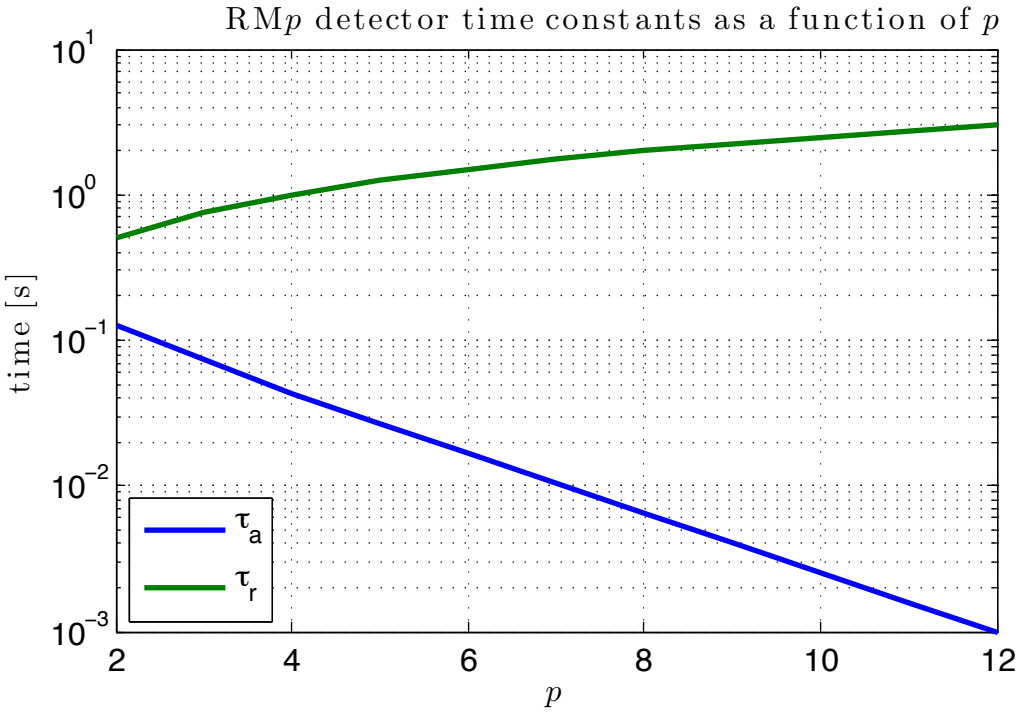

Figure 5: τ_a, τ_r as a function of p, for $\tau = 0.25$ [s] and $f_s = 44100$