CSC 361/661 Digital Media

Worksheet Dr. Jennifer Burg (burg@wfu.edu)

Lab Worksheet – Digital Audio > Non-Linear Companding and Dynamic Range[1]

Objective:

1. To observe the benefit of non-linear encoding by computing the encoded values using a logarithmic function and determining the increased level of detail with which low amplitude signals are quantized.
2. To observe the manner in which compression error is distributed when 16 bits are compressed into 8 bits and decompressed back to 16 bits using the μ-law logarithmic function.

Introduction:

μ-law encoding is an example of a non-linear companding (compression and expansion) method used in telephone communication. Rather than quantizing audio samples with evenly spaced quantization levels, non-linear companding quantizes lower amplitude values in more detail than higher amplitude ones. This method works well for telephone communication because it reduces the signal-to-noise ratio in the area where it matters most – in low amplitude values, which are common in human speech and are particularly subject to noise distortion. The equation for non-linear compression by μ-law encoding is

$$y = \frac{\text{sgn}(x) * \log_2(1 + \mu|x|)}{\log_2(1+\mu)}$$

where $-1 \le x \le 1$, sgn() is the sign function. μ is 255 when samples are being quantized to 8 bits.

Decompression operates by the inverse equation:

$$y = \text{sgn}(x) * \frac{1}{\mu}\left[(\mu+1)^{x} - 1\right]$$

Using MatLab, let's see what effect the compression equation has on audio samples.

Open MATLAB on your computer. (On my computer it's Start\WFU Academic Tools\Scientific Applications\MATLAB 6.5\MATLAB.) In the Command Window, type the following commands, hitting Enter each time. The MATLAB commands below are given in italics.

Create the μ-law function with μ = 255. Call it f.

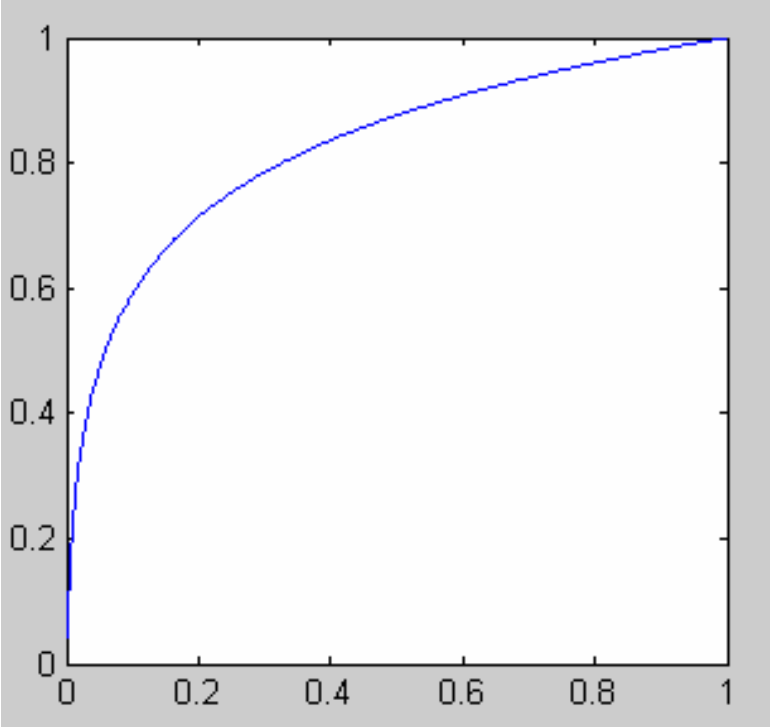
[1] This material is based on work supported by the National Science Foundation under Grant No. DUE-0127280. This worksheet was written by Dr. Jennifer Burg (burg@wfu.edu).

*f = inline('(log2(1+255*abs(x)))/8)', 'x');*

Plot the μ-law function over the domain interval x=[0 1].

fplot(f, [0 1]);

A graph window should pop open and look something like this:



Let's think about what this graph represents and how it relates to the digital encoding of audio samples. Imagine that you are trying to digitize the analog audio wave like the one pictured below.

In non-linear companding, 16-bit digital audio samples are compressed to 8-bit values at the transmission end, and then decompressed back to 16-bit values when they are received. We want to see how much error is introduced by this compression.

The logarithmic function above is the right "shape," but it doesn't have the right units. Let's change the function so that it maps 16-bit samples to 8-bit ones.

With n bits, 2^n different quantization levels ranging from -2^{n-1} to $2^{n-1}-1$ can be represented. If 16 bits are initially used for each audio sample, then we can represent 2^{16}=65,536 different quantization levels ranging from -2^{15} to $2^{15}-1$ (that is, −32,768 to 32,767). After reducing the bit depth to 8 bits, we can represent values between 2^7 and 2^7-1 (that is, -128 to 127).

Accordingly, let's change the function in MATLAB so that it takes values between -32,768 and 32,767 as input and yields integer values between 0 and 128 as output. Here are the MATLAB command.

```
q = inline('round(128*((log2(1+255*abs(x/32768)))/8))', 'x');
fplot(q, [0 32768]);
```

Question 1: Which of the following statements is true?
q(256) – q(1) > q(32767) – q(32512) OR
q(32767) – q(32512) > q(256) – q(1)?

Question 2: Another way of asking question one is this – which segment has more 1 unit quantums in it: the portion of the y-axis between f(1) and f(256) or the portion of the y-axis between f(32512) and f(32767)?

Question 3: Based on your answers to questions 1 and 2, which are samples quantized in a more fine-grained manner – the low amplitude input values, or the high amplitude ones?

Another way to view this is to plot the function on a smaller scale, starting at the low amplitude end. We'll begin with x = 0 to 1000 and go up from there. Try this:

```
fplot(q, [0 1000]);
```

Keep doing this at 1000 amplitude intervals and observe how the function changes.

```
fplot(q, [1001 2000]);

fplot(q, [2001 3000]);
```

etc.

Question 4: How does the graph of the function change as you move from low amplitudes to high amplitudes?

Let's compare this to the graph of a function that effectively maps 16-bit values to 8-bit values by dividing by 256 and throwing away the remainder.

Question 5: Think in terms of the binary representations of 16 bit versus 8 bit samples. What simple bit operation effectively s a 16 bit sample and reduces it to 8 bits in a way that would be equivalent to dividing the corresponding base-10 value by 256?

We'll call the new function d. Type the following commands:

```
d = inline('floor(x/256)', 'x';
```

fplot(d,[0 32768]);

Compare the graph for function d to the one for function q. You might want to look at this graph closer, as you did with the μ-law function, by zooming in on 1000 unit sections, as in

fplot(d, [1001 2000]);

Question 6: What's the difference between the graphs for functions d and q?

Question 7: Assuming that d and q represent two different ways of compressing 16 bit samples to 8 bit samples, explain what these graphs show you about relative precision with which d and q measure samples at different amplitudes?

Question 8: For function q, how many different integer input values from the domain -32,768 to 32,767 map to an output of 1? (Hint: In MATLAB, to evaluate the function f at point 1, use
f(1)
Notice that there is no semicolon after this statement. You may also want to use the inverse of the μ-law function – the decompression function given above – to work "backwards.")

Question 9: In question 6, you identified m input values that map to 1 with the function q. When you decompress these with the inverse function, what do all of these m values map back to (i.e., what is their value in 16-bits after compression)?

Question 10: In a table below, give the percent error for each of these m values (as a ratio of the error to the original value)?

Question 11: For function q, how many different integer input values from the domain -32,768 to 32,767 map to an output of 127?

Question 12: What is the percent error for the smallest of these input values (from question 11)?

Question 13: What is the percent error for the largest of these input values (from question 11)?

Conclusions:

Non-linear companding methods such as μ-law encoding take 16-bit digital audio samples and compress them to 8 bits. 16-bit samples offer a dynamic range of 96 dB. 8-bit samples offer a dynamic range of 48 dB. You might expect that in compressing 16-bit samples to 8-bits, we would be sacrificing half the dynamic range in a digital audio file, but this is not the case. Upon decompression with non-linear companding, the low amplitude samples return to values very close to what they were originally, so we haven't lost the ability to represent low amplitude signals clearly. The result is that μ-law encoding effectively yields a dynamic range of about 72 dB using only 8-bits.