EE-3221 LABORATORY
Quantization Error

Overview
In this lab exercise you will investigate sampled audio signals and then explore the effects of quantization error through MATLAB simulation.

Sampled Audio in MATLAB
A vector of samples in MATLAB can be played as an audio signal using:

```
sound(x, fs, BITS)
```

where x is the vector (or sequence) of samples, fs is the sampling frequency of the sequence, and BITS is the number of bits used to represent the audio signal in the digital-to-analog converter (DAC) of your sound card. If fs and BITS are not specified, they default to 8192 samples/second and 16 bits/sample.

Sinusoids
Run MATLAB and create an index vector that will correspond to 2 seconds of time.

```
fs = 8192;        % fs = 8192 samples/second, default sample rate
Ts = <?>          % sample period Ts = 1/fs seconds/sample & is inverse of fs
t_max = 2;        % max time is 2 seconds
N = t_max/<?>     % determine the number of samples that corresponds to 2
seconds of a sampled sequence (care must be taken that n_max is an integer;
here the values are arranged so it will be, but try help round and doc ceil
for common ways to convert floating point numbers to integer values)
n = 0:1:(N-1); % maximum value is N-1 since we start at 0
% alternatively we can define N = fs*t_max
```

Now we create the sine wave signal. Let's create a 440 Hz (concert pitch "A") tone. We take the desired frequency and multiply it by $2\pi/fs$ or $2\pi \cdot Ts$, which are equivalent. Note that Ts*n is the sampled time in seconds.

```
f_A = 440              % concert pitch A is 440 Hz
x1 = sin(2*pi*f_A*Ts*n);
```

We can listen to the signal through the computer's sound card using **`sound(x, fs)`**. **Always turn down the volume on your computer before playing a sound. The full scale volume is very loud!**

```
sound(x1, fs)
```

```
sound(0.1*x1, fs)
```

The second call attempts to scale the output voltage by 0.1. We would expect this to result in $20\times\log_{10}(0.1)$ = -20 dB power gain, which would clearly give an audible difference. This rests on a few assumptions, the key one being that the sound card driver isn't one that automatically increases the gain for quiet sounds in an attempt to be helpful.

Next create a sine wave signal that represents an 880 Hz tone.

x2 = <?>

1. *What is the expression used to find the sequence vector for x2?*

Listen to x2:

```
sound(x2, fs)
```

2. *Describe the differences between the two sounds. What causes the difference?*

Musical Chords

In western music, a chord is a combination of tones that sound as if (and perhaps are) sounded simultaneously that are related by ratios of the fundamental tone. For example, an A-major chord consists of an A note at 440 Hz, a C# note at about 1.26*440 Hz, and an E note at about 1.5*440 Hz. Those who are musically inclined may know that the exact frequency multipliers for the 3 components are 0, 4, and 7 semitones above the root, which can be computed in MATLAB with 2.^([0 4 7]/12)

Let's build the tones for the notes that make up an A-major chord.

```
x1 = sin(         440*2*pi/fs*n);
x2 = sin(2^(4/12)*440*2*pi/fs*n); % Approx. 1.26 of fundamental
x3 = sin(2^(7/12)*440*2*pi/fs*n); % Approx. 1.5  of fundamental
xs = [x1 x2 x3]; % explain what this does
sound(xs, fs)
xc = mean([x1; x2; x3]);  % explain what this does
sound(xc, fs)
```

3. *Explain what **[x1 x2 x3]** and **mean([x1; x2; x3])** do. Hint: Examine the sizes of the inputs and outputs.*
4. *Suppose you were to examine the frequency spectrum of signals **xs** and **xc** using a spectrum analyzer. Describe the ways in which the spectrum of **xs** and **xc** are similar. Describe how they differ.*

.wav Audio Files

Download the file plumclip.wav from https://faculty-web.msoe.edu/prust/EE3221 .
To find MATLAB's default folder on your system, start MATLAB and enter pwd ("print working directory"). This default folder is a convenient place to save the WAV file. You can change MATLAB's current folder at the top of the command window or with the `cd` ("change directory") command. To import the file into a vector, type:

```
[original_clip, fs] = audioread('plumclip.wav');
```

This .wav file has a sampling rate of 44100 Hz with 16 bits per sample. These are standard industry specifications for CD quality audio. Listen to the clip:

```
sound(original_clip, fs)
```

Low Resolution Samples: The Sound of Quantization Noise and Signal to Noise Ratio (SNR)

The original .wav file used 16 bits to represent the amplitude of each sample. This represents a resolution of the amplitude equal to 2^{16} = 65,536 different levels. To hear the effect of the integer approximations used in quantization, let's reduce the resolution to 8 bits per sample.

5. *How many possible amplitude levels are possible when 8 bits are used to quantize the signal?*

To do this, we will make use of MATLAB's round function, which rounds to the nearest integer.

To obtain the integer values present in the original .wav file, we must rescale the sound vector original_clip, which currently has values from -1 to 1. We want to represent the sound vector the way it was stored in the .wav file, with each sample represented by a 16-bit unsigned integer. The following will offset the signal so it ranges from 0 to 2 instead of -1 to 1, and then scale the samples so they range from 0 to 2^{16}.

```
integer_original_clip = (original_clip+1) * pow2(15);
```

Now, reduce the resolution to 8 bits by dividing by 2^(16-8) and using round() to round the results of the sequence values. Remember, our original clip has 16 bits of resolution. Dividing a binary number by 2 discards the least significant bit and dividing by 2 eight times will discard the lowest 8 bits. This leaves only the 8 most significant bits out of the original 16 bits.

```
integer_lowres_clip = round(integer_original_clip/pow2(8));
```

Now, each sample ranges from 0 to 255 since we are down to 8 bits of resolution. Note that MATLAB uses doubles to store the integer results in this case; MATLAB also supports true integer data types, but they generally aren't used unless utmost efficiency is required. Scale and offset the 8-bit integer vector back to the range of -1 to 1 for playing in MATLAB and listen to the result:

```
lowres_clip = integer_lowres_clip/pow2(7) - 1;
sound(lowres_clip, fs); % make sure you have the correct fs here.
```

6. *Describe what you hear. What might be causing this?*

Let's create a function that will let us hear what the clip will sound like with different bit resolutions ranging from 2 to 16 bits. Here's the function:

```
function new_audio_clip = ChangeBitRes(original_audio_clip, resolution_bits)
% Convert the amplitude of the original 16-bit audio clip to an unsigned
% integer range from 0 to 2^16.
integer_original_clip = (original_audio_clip+1)*pow2(16-1);
% Discard 16-resolution_bits to keep a vector that is in the integer range
% of 0 to 2^(resolution_bits)
integer_lowres_clip = round(integer_original_clip/(pow2(16-resolution_bits)));
% Scale the amplitudes that range from 0 to 2^resolution_bits back to the
% range of -1 to +1.
new_audio_clip = integer_lowres_clip/(pow2(resolution_bits-1))-1;
end
```

Save the function and experiment with the song using different resolutions. For example, try:

```
sound(ChangeBitRes(original_clip,12), fs) % 12 bits used to quantize amplitude
sound(ChangeBitRes(original_clip, 6), fs)
sound(ChangeBitRes(original_clip, 4), fs)
```

Rounding and quantization adds error to each sample. The amount of error in each sample does not follow any particular pattern, so it sounds like a shhhh-ing sound which we call white noise. We will discuss this effect later in the course as well.

Play the low resolution clip into the oscilloscope and observe the spectrum (FFT function on lab scopes).

7. *What do you see that is different in the spectrum compared to the original clip? Go to even lower resolution if needed so that you can see a difference.*
8. *Describe, qualitatively, what happens when fewer bits are used to represent the samples.*

Quantization Error and SQNR

As observed in the previous exercise, quantization of a sampled signal results in errors between the original (unquantized) amplitudes and the quantized amplitudes. This difference is referred to as *quantization error* and is often modelled as a noise that is added to the original signal. The ratio of the signal power to the quantization noise power is known as the *signal-to-quantization-noise ratio*, or *SQNR*.

$$SQNR = 10\log_{10}\frac{P_x}{P_n}$$

P_x = signal power

P_n = quantization noise power

In this portion of the exercise, we will carefully simulate the quantization process, examine characteristics of the quantization noise, and compare measured SQNR to theoretical predictions.

A well-known result in the DSP community is that SQNR increases by 6.02 dB for each additional bit used in the quantizer. The exact SQNR for a given signal depends on various factors, including the statistics of the signal itself. For example, the SQNR of a sinusoidal signal can be shown (under a set of assumptions) to be

$$SQNR = 6.02b + 1.76 \text{ dB}$$

where b is the number of bits used in the quantizer. For example, the SQNR of a sinusoid sampled using a 14-bit quantizer is, in theory, 6.02(14)+1.76 dB = 86.04 dB. Since 86.04 dB corresponds to 4.02E8, the signal power in the quantized signal is *400 million* times larger than the quantization noise power!

Let's now simulate a quantizer in MATLAB. We begin by generating samples of a sinusoid:

```
fs = 1000;
Ts = 1/fs;
t = 0:Ts:1;
x = sin(2*pi*4*t); % 4 Hz sinusoid
```

By default, MATLAB uses "double" precision (according to IEEE Standard 754) in calculating these sample points, which utilizes a 64-bit floating-point representation. Suffice to say, the representation is extremely accurate.

We now create a 3-bit quantized version of the sinusoid, paying careful attention to placement of the quantization levels.

```
Nbits = 3; % number of bits
L = 2^Nbits; % number of levels
xq_int = floor((x+1) * (L/2)); % quantization level, integer on [0,L]
xq = (xq_int - L/2 + 1/2) / (L/2); % quantized value (rounded)
```

(Note: The above formula "cheats" when x is precisely at its maximum of 1. The quantized integer should be on [0,L) (including 0, not including L), but is L in this case. Here, this edge case not very important, but care must be taken in practice since assigning an integer 1 beyond the maximum will wrap around to 0 for integer data types in many systems. Many DSPs support "saturating arithmetic" that prevents this, replacing overflows with the most appropriate extreme value.)

We can plot the original signal x and the quantized signal xq with the following:

```
figure
plot(t,x,t,xq)
xlabel('t')
legend('x','xq')
```

Carefully inspect this plot, paying special attention to the placement of the quantization levels. Note that values of x are "rounded" to the nearest quantization level. This particular quantizer is known as a "midrise" quantizer.

The quantization error can be calculated as

```
e = x-xq;
```

9. *Create a plot showing x, xq, and e on the same time axis (i.e., one figure showing all three signals). Label the time axis and include a legend. Include this plot in your submittal. What is the range of e, and how does this range correspond to the step size of the quantizer?*

We now calculate the simulated SQNR as

```
S = mean(xq.^2); % signal power is mean square value
Q = mean(e.^2); % noise power is mean square value
SQNR = 10*log10(S/Q); % in dB
```

You should find the SQNR to be 18.7 dB. The theoretical SQNR for a 3-bit quantizer is 19.8 dB.

10. *Repeat this simulation for each of the quantizers listed in the table below. Complete the table and include in your submittal.*

Number of bits	Simulated SQNR (dB)	Theoretical SQNR (dB)
2		
4		
8		
12		
16		

A modified version of the quantizer can be simulated by replacing the previous quantization code with

```
Nbits = 3; % number of bits
L = 2^Nbits; % number of levels
xq_int = floor((x+1) * (L/2)); % quantization level
xq = (xq_int - L/2) / (L/2); % quantized value (truncated)
```

Notice the line that computes xq differs from the previous quantizer. One interpretation of this new quantizer is that it truncates to an integer bit rather than rounding to the nearest bit.

11. *Create a plot showing x, xq, and e on the same time axis (i.e., one figure showing all three signals) for this 3-bit "truncating" quantizer. Label the time axis and include a legend. Include this plot in your submittal. What is the range of e, and how does this range correspond to the step size of the quantizer?*
12. *Simulate this "truncating" quantizer using 8 bits. What is the resulting SQNR? How does this SQNR compare to the SQNR for the 8 bit "rounding" quantizer that was previously simulated? Explain the results.*