Lab 2

In this lab we will be exploring the concepts of sampling, quantization and dither.

Watch this!

If you haven't already, I recommend watching this video, it explains everything we're covering in this section in a clear and easy to understand presentation:
http://www.xiph.org/video/vid2.shtml

Frequency Sweeps

To investigate sampling rates, we're going to sweep the frequency of a sine wave. In order to do this we need to talk about phase modulation.
Up until now we've been using the equation: `sin(2*pi*f0*t)` to synthesize our sine wave. If we want to sweep the frequency from 100 Hz to 4000 Hz, the natural assumption would be to use a series of statements like this:

```
Fs = 8000;
Ts = 1/Fs;
t = 0:Ts:4;
f = linspace(100, 4000, length(t));   % (see
explanation of linspace below)
y = sin(2*pi*f .* t);  %  (notice that I use .*
because f and t are both vectors)
```

This, however, is wrong!!! The reason is that our equation assumes that f is a constant. For each value of f, it is calculating the sine function as if we have been using that frequency since time t = 0. Since the frequency changes with time, however, we will get the wrong result.

We can find the correct way to do a frequency sweep by thinking about units. If we look at the terms inside of the sin() function, we see that they are in terms of radians. The sin() function sees a continuously increasing number of radians that results in our phasor going around and around.

The way we need to think about generating our function is by having the number of radians increase by the correct amount for the frequency that we want at that moment in time.

```
    dTheta = 2*pi*f*Ts;
```

This command gives us a vector where each element is the increment of radians for that sample (theta is usually used for the phase angle of a sinusoid). If we now do a running integration, or cumulative sum of this vector, it will give us the correct value of theta at each sample. Fortunately Matlab has just such a function:

```
    theta = cumsum(dTheta);
```

If we now use theta to generate our signal, everything will be correct:

```
    y = sin(theta);
```

Problem 1 below will ask you to generate a swept sine wave to investigate aliasing.

Quantization

In order to investigate the effects of quantization and dither, I have written a function for you to use called **quantBits**. You can use the help command for instructions on how to use it, and are welcome to see the code if you open it in the editor. It's actually not technically correct in terms of how an actual A-D converter behaves, but it's close enough to illustrate the concepts for this lab.

Noise

We will define noise as a random signal. There are a couple of random functions: rand(), randn() and randi(). randi() generates random integers, so it's not very useful to us. randn() generates random values in a normal distribution, which means it has no theoretical maximum value. **rand()**generates random uniformly distributed real numbers from [0 1]. This is probably the way to go, although MATLAB tricks could be done to any of these functions to generate random noise. rand() takes 1 or 2 parameters, and works just like zeros() and ones(), creating a matrix if only 1 parameter is specified.

```
rand(size)
rand(rows,columns)
```
One parameter produces a square matrix, and 2 produces a matrix of size rows × columns.

linspace()

One command that will make your life easier is called linspace(). This command will give you a vector starting with some starting value, ending in another, and gives you a vector of a given length. There is nothing in this command that you can't do with vector tricks. Allow me to demonstrate with an example:

```
beginning = startvalue;
ending = endvalue;
numitems = n;
stepsize = (ending - beginning) / (numitems-1);
vector = beginning:stepsize:ending;
```

Alternatively, linspace can do the same thing in one line:

```
vector = linspace(beginning,ending,numitems);
```

Assignment:

(note please make all wav outputs less than 5 seconds)

1) Sine sweeps and aliasing

a) Generate a swept sine wave from 100 Hz to 5 kHz using a sampling rate of 4 kHz. Export the output as a wav file (YourName-Lab2-1a.wav). Explain the result. What is the last frequency heard?

b) What happens when you increase the sampling rate to 6 kHz?

c) What is the minimum sampling rate needed to correctly represent this signal? Try it and see if you were correct. Export the output as a wav file (YourName-Lab2-1c.wav).

2) Quantization

a) Generate a sine wave and use **quantBits** to quantize it to 8, 4, and 2 bits respectively. Describe what you hear (no need to generate wav files).

b) Plot a few periods of the wavform in a 3 x 1 subplot showing how the waveform changes.

BONUS) If the audio signal is represented as integers using the two's complement method in bits, explain how a real A-D converter would differ from the quantBits function.

3) Dither: we will illustrate the two benefits of dither in six steps.

Use a sampling rate of at least 22050 Hz. Make the y-axis go from +/-1.1 for all plots.

a) Generate a 500 Hz sinusoid 3 seconds in length with an amplitude +/- 1.

b) Use quantBits to quantize the signal from part a to only 1 bit. Plot the original and quantized versions in a 2 x 1 subplot (please only plot a few periods of the signal).

c) Generate a 500 Hz sinusoid 3 seconds in length with an amplitude +/- 0.4.

d) Use quantBits to quantize the signal from part c to only 1 bit. Plot the original and quantized versions in a 2 x 1 subplot (please only plot a few periods of the signal).

e) Generate a noise signal using rand() that has an amplitude of +/- 0.5. Add it to the signal from part c.

f) Use quantBits to quantize the signal from part e to only 1 bit. Plot the original and quantized versions in a 2 x 1 subplot (please only plot a few periods of the signal).

g) Listen to all six versions of the signal and describe what your hear for each. For the signal in part f, what two benefits did we achieve by adding noise to the signal? (Note: no need to output wav files, simply describe what you see and hear)

BONUS) Repeat parts e and f using dither that has a triangular probability distribution function (hint: you may need to do a bit of research to find out what the correct amplitude is of this type of dither).

Submission Requirements

Export all plots as .png files and use Word (or similar) to compile them into one document along with a writeup for each section briefly explaining what you did, and what problems (if any) you encountered in the process. All plots and write up should be submitted as a **single PDF file**. Please also submit your code, either in one .m file, or in separate files for each portion. Either way, **make sure you name your files so that I know what they are!**

Lab 2

Nate Paternoster 9/19/13

1) Code:

```
%1a)

fs = 4000;
Ts = 1/fs;
t = 0:Ts:4;
f = linspace(100,5000,length(t));
dtheta = 2*pi*f*Ts;
theta = cumsum(dtheta);
y = 0.5*sin(theta);

sound(y,fs);
wavwrite(y,fs,'NatePaternoster-Lab2-1a.wav');

%1b)

fs = 5000;
Ts = 1/fs;
t = 0:Ts:4;
f = linspace(100,5000,length(t));
dtheta = 2*pi*f*Ts;
theta = cumsum(dtheta);
y = 0.5*sin(theta);

sound(y,fs);
```

```
%1c)

fs = 10000;
Ts = 1/fs;
t = 0:Ts:4;
f = linspace(100,5000,length(t));
dtheta = 2*pi*f*Ts;
theta = cumsum(dtheta);
y = 0.5*sin(theta);

sound(y,fs);
wavwrite(y,fs,'NatePaternoster-Lab2-1c.wav');
```

→ Details

a) The sampling rate could not keep up with the highest frequency. The last frequency you can hear is 2kHz because that is the Nyquist frequency, half the sampling rate. After the frequency sweep passed 2kHz it wrapped around the Nyquist frequency and swept down again.

b) When the sampling rate is increased to 6kHz, the highest frequency heard becomes 3k (the new Nyquist frequency). The first half of the frequency sweep sweeps from 100Hz to 3kHz and the second half sweeps back down from 3kHz to 100Hz (once again the sweep 'wrapped around' the Nyquist frequency).

c) The minimum sampling frequency needed to represent this signal is 10kHz. The Nyquist frequency here would be 5kHz, the highest frequency of this signal.

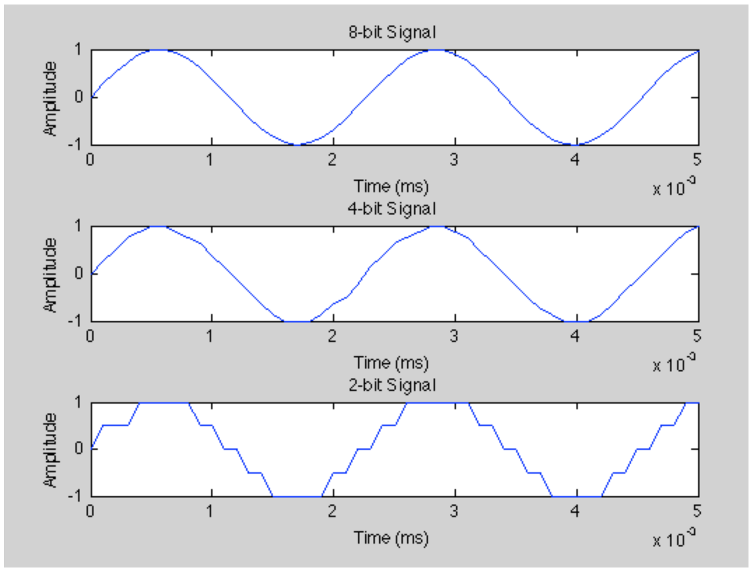
2) Code:

```
%2a)

fs = 10000;
Ts = 1/fs;
t = 0:Ts:4;
y = 0.5*sin(2*pi*440*t);
y_8bit = quantBits(y,8);
y_4bit = quantBits(y,4);
y_2bit = quantBits(y,2);

sound(y,fs);
sound(y_8bit,fs);
sound(y_4bit,fs);
sound(y_2bit,fs);

%2b)

fs = 10000;
Ts = 1/fs;
tplot = 0:Ts:0.005;
y = sin(2*pi*440*tplot);
y_8bit = quantBits(y,8);
y_4bit = quantBits(y,4);
y_2bit = quantBits(y,2);

subplot(3,1,1);
plot(tplot,y_8bit);
title('8-bit Signal');
xlabel('Time (ms)');
ylabel('Amplitude');
```

```
subplot(3,1,2);
plot(tplot,y_4bit);
title('4-bit Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

subplot(3,1,3);
plot(tplot,y_2bit);
title('2-bit Signal');
xlabel('Time (ms)');
ylabel('Amplitude');
```

→ Details

a) All three quantized signals sounded distorted, noisier than the original. In the 8-bit signal you can hear very clearly an upper harmonic of the fundamental pitch. In the 4-bit it is a higher harmonic, and in the 2-bit it is an even higher harmonic.



Quantized signals from 2b

3) Code:

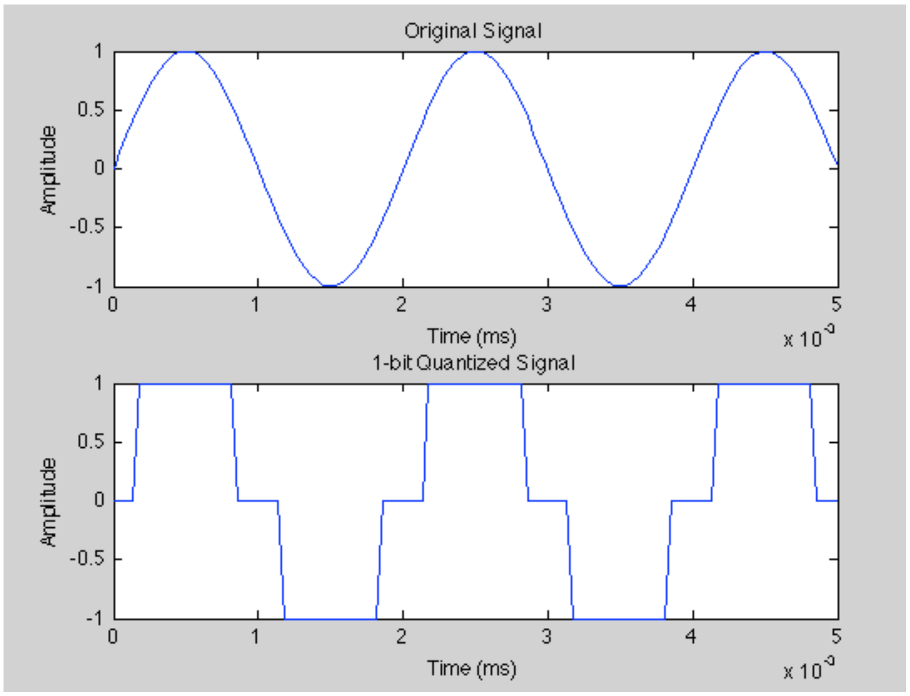
```
%For Part 3 I used two different functions - one to listen to the
waveform, and the other to plot the waveform. The functions I
used to plot were y1, y2, and y3, as well as y1_1bit, y2_1bit, and
y3_1bit. I used ya, yb, yc, ya_1bit, yb_1bit, and yc_1bit to listen
to.

%3a+b)

fs = 22050;
Ts = 1/fs;
t = 0:Ts:3;
tplot = 0:Ts:0.005;
ya = 1*sin(2*pi*500*t);
ya_1bit = quantBits(y_a,1);

y1 = 1*sin(2*pi*500*tplot);
y1_1bit = quantBits(y1,1);

subplot(2,1,1);
ylim([-1.1 1.1]);
plot(tplot,y1);
title('Original Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

subplot(2,1,2);
ylim([-1.1 1.1]);
plot(tplot,y1_1bit);
title('1-bit Quantized Signal');
xlabel('Time (ms)');
ylabel('Amplitude');
```

```
%3c+d)

fs = 22050;
Ts = 1/fs;
t = 0:Ts:3;
tplot = 0:Ts:0.005;
yb = 0.4*sin(2*pi*500*t);
yb_1bit = quantBits(y_b,1);

y2 = 0.4*sin(2*pi*500*tplot);
y2_1bit = quantBits(y2,1);

subplot(2,1,1);
ylim([-1.1 1.1]);
plot(tplot,y2);
title('Original Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

subplot(2,1,2);
ylim([-1.1 1.1]);
plot(tplot,y2_1bit);
title('1-bit Quantized Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

%3e+f)

y_rand = 0.5*rand(1,111);
y_rand2 = 0.5*rand(1,66151);

yc = yb + y_rand2;
yc_1bit = quantBits(yc,1);
```

```
y3 = y2 + y_rand;
y3_1bit = quantBits(y3,1);


subplot(2,1,1);
ylim([-1.1 1.1]);
plot(tplot,y3);
title('Original Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

subplot(2,1,2);
ylim([-1.1 1.1]);
plot(tplot,y3_1bit);
title('1-bit Quantized Signal');
xlabel('Time (ms)');
ylabel('Amplitude');

%3g)

sound(ya,fs);
sound(ya_1bit,fs);

sound(yb,fs);
sound(yb_1bit,fs);

sound(yc,fs);
sound(yc_1bit,fs);
```

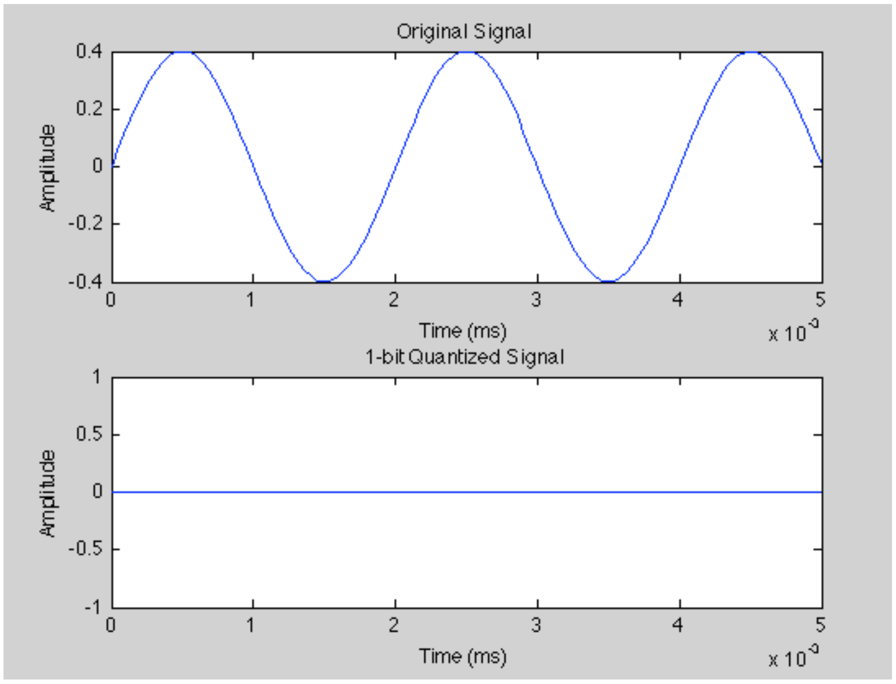
→ Details

a) + b)



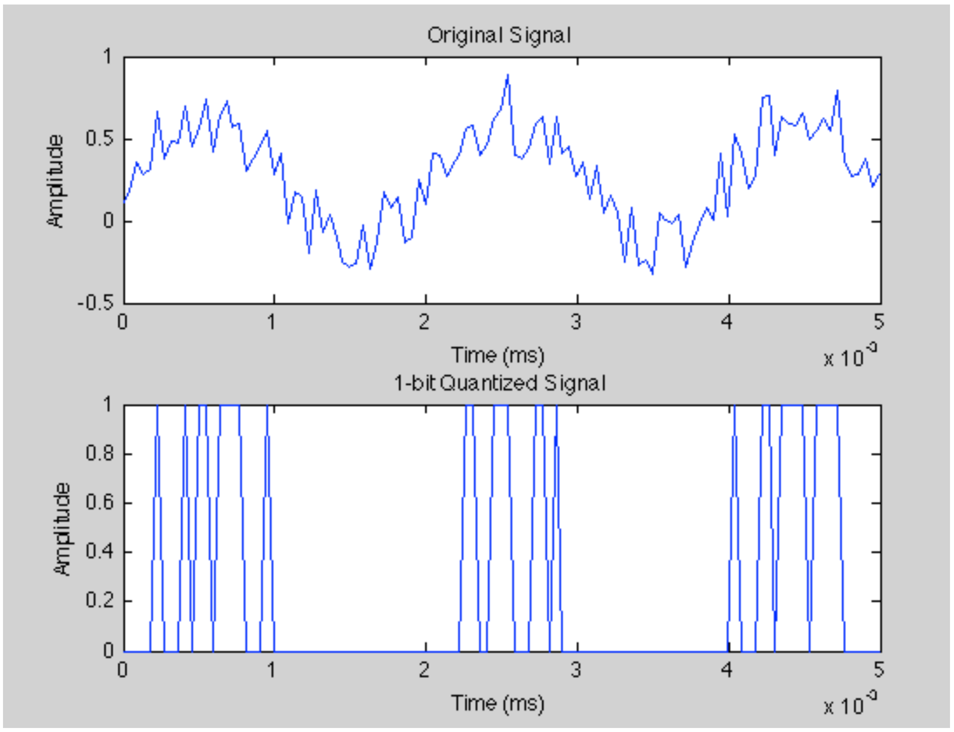
A 500Hz signal with amplitude of 1 and the same signal quantized to 1 bit

c) + d)



A 500Hz signal with amplitude of 0.4 and the same signal quantized to 1 bit

e) + f)



The 500Hz signal with amplitude of 0.4 with noise added and the same signal quantized to 1 bit

g) The first two waveforms (1 and 2) were for a 500Hz sinusoid and that same sinusoid quantized to 1-bit. The quantized version sounded a lot crunchier and the first harmonic was actually more prominent than the fundamental frequency.

The second two waveforms (3 and 4) were for a quieter 500Hz sinusoid and it's 1-bit quantized version. Here the quantized signal was quantized to all zero values. The signal was lost and on playback you just hear silence.

The last two waveforms (5 and 6) were the same as 3 and 4 except with some noise added. Because the noise was set louder than the sinusoid, the original signal sounded muffled by the noise. However, unlike 4, the quantized version 6 was clearly audible and you could distinguish the original 500Hz pitch fairly clearly. Also, unlike 2, 6 sounded the fundamental pitch clearly instead of the first harmonic.