Borja Rojo Lab 1 EAS 103-Bocko

**Lab 1.1**

|  |  |
| --- | --- |
| Sample Rate | Sound quality |
| 44100 | Clean sounding |
| 22050 | Clean Sounding |
| 11025 | Rougher, but still clean |
| 5512 | The sound seems like its old |
| 2756 | The sound sounds almost like it’s video game music |
| 1378 | Hard Crush sound |
| 689 | Pitch was not coherent, Crushed sound |

The task of this code was to create a sound using the values given by the function, x= Acos(2pi f t + phi) where A=1, f=220, phi = pi/3, and a t array from 0 to 1 with an interval that is inversely proportional to sample rate R, which was 44100. Using the *soundsc* command, the values x across the samples of t played a cosine wave whose phase was off-set by pi/3. The *plot* command was also used in order to plot the values of the cosine wave with respect to time with increments equal to that of the rate.

If I turned down the sampling rate, the sound would get harsher and les clear. The lowest sampling rate I could get in order to keep that sound distinguishable was no less then twice the frequency of the pitch.

**Lab 1.2**

The task of this code was to create a sound that played the value of two sound wave together at a slightly off pitch tuning. This time, sine waves were used as opposed to cosine waves. Using the standard model Asin(2pi f t + phi) and adjusting the parameters of the frequency in both waves to be slightly off-set, the sound produced by the *soundsc* command sounded out of tune.

The critical value of Δf for is about 3 cents, or 3% of the way to the next semitone. That is when the beats become easily audible. The critical value of Δf in order for the beats to change into what sounds like two distinct notes is for the second frequency to become one semitone higher then f1, or for the Δf to be equal to 2(1/12) x f1 – f1 . When varying the amplitudes of the notes, the beats were strongest when the signals of both the sounds were equal. If the phase of one of the sounds is changed, there is no audible effect that I could hear.

**Lab 1.3-5**

The task of these codes was to create sawtooth, square, and triangle waves using Fourier Series. This required that there be a sequence of sine waves created and then added together into a series in order to create a desired wave form. The way this was accomplished was by using a for-loop to create functions of x into an array, n, with step 1 from 1 to the desired amount of harmonics, and then added together within the x variable. In order to get the fullest sound, I decided to use as many harmonics possible that fit into the human audio spectrum by denoting n to equal 20000/fundamental. Each waveform has their own respective series what are used in the code.

**Messing with the Fourier Series Parameters**

I made some funky sounding wave forms, though none of them were particularly notable in their shape.