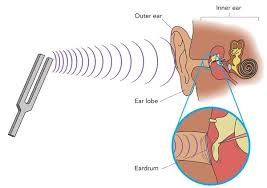
**Fast Fourier Transform**

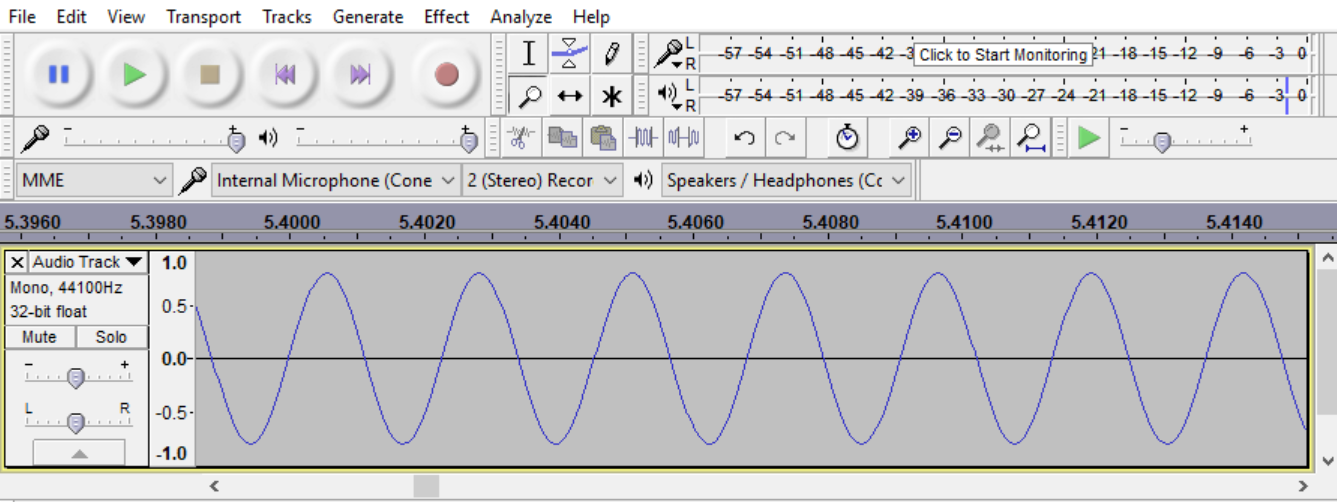
*Part 1: The Basics of Sound*

It turns out that Fourier series are very common in nature, very much different from Taylor series, which only exit on paper as theoretical constructs.

Perhaps the easiest to understand real-existing Fourier series is represented by sound. Any sound exists as variations of pressure in a medium such as air, generating a wave. If I hit a tuning fork, for example, it vibrates, which causes the air surrounding it to vibrate. This vibrating wave spreads through the air and eventually reaches our ear, where it is making our eardrum vibrate as well. These vibrations are received by neurons, converted into electrical pulses and transported to our brain. Our brain then interprets this as sound.

A tuning fork, for example, generates a single sine wave with 440 Hz, which means that the sine wave oscillates 440 times per second. If a wave oscillates faster, the corresponding sound is higher, if it oscillates slower, the sound becomes lower. More complicated sounds can be thought of as a superposition of sine and cosine waves.

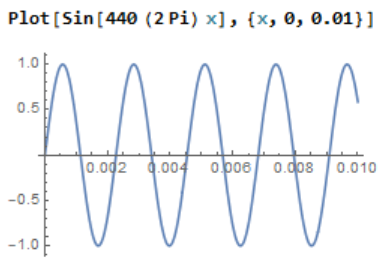
**Example**: Download ***Audacity***, a free and very capable sound editor, from <http://audacity.sourceforge.net/download/> and install it. Start the program, select “Generate | Tone” from the menu, then set the parameters to generate a sine wave of 440 Hz, with an amplitude of 0.8, that lasts 30 seconds (which are all default values, so there is nothing to do for you). Click OK to generate that sound. No sound will play just yet. Zoom in by pressing CTRL-1 multiple times until you have zoomed in sufficiently to see the sine wave. Hit “Play” (or press the spacebar) to play the sound: the computer speaker’s membrane will vibrate 440 times per second, causing the air to vibrate, which spreads the wave to reach your ear drum, which will also start to vibrate, which your brain interprets as an ‘A’ sound.

T

Thus, we can *see* *and* *hear* a (simple) Fourier series, consisting of a sine wave at 440 Hz: . Pretty neat, eh?

Of course, we can visualize this function in Mathematica as well, but we can’t *hear* the sound that the wave makes.

**Example**: Generate the graph of this function with Mathematica.

We need to plot the graph of a sine wave with 440 Hz. In other words, the wave oscillates 440 times per second. Thus, the function is . If we plotted the graph from x between 0 and 1, we would have to squeeze 440 oscillations into the space from time zero to time 1 second. That would not look nice, so we will only draw the function for . Since that interval length is 1/100 of the original interval, which would hold 440 oscillations, if I plot it from 0 to 0.01 I would see 4.4 oscillations. See figure above on the right.

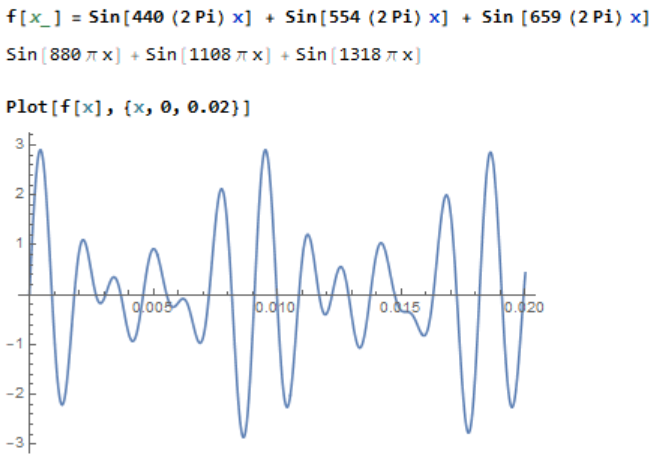
We can now compose more complicated sounds by generating several sine waves and then superpositioning them (superpositioning: adding the functions as math functions, “mixing” them as “tracks” in Audacity terminology). For example, suppose we want to generate a *A – C# - E* chord. We have already learned that the A note is a 440 Hz sin wave, so we need to figure out the frequencies of C# and E. For a usual chromatic scale the frequency increases by 12 steps from one octave to another. Starting at the 440 Hz note A, the frequencies and notes of the chromatic scale are:

|  |  |  |  |
| --- | --- | --- | --- |
| = **A** | = **A#** | = **B** | = **C** |
| = **C#** | = **D** | = **D#** | = **E** |
| = **F** | = **F#** | = **G** | = **G#** |

After that, the next octave starts with **A** again, this time at 880 Hz, but we can for simplicity restrict ourselves to the ‘standard’ octave between 440 Hz and 880 Hz.

**Example:** Use Audacity to compose the chord *A – C# - E*

First, let’s do this mathematically, so that we know what the *A - C# - E* chord looks like. We need to add sine waves with 440 Hz (A), 554 Hz (C#), and 659 Hz (E) and plot the sum from, say, x = 0 to x = 0.02:



Here is how to overlay the three notes *A – C# - E* in Audacity. The idea is simple: generate each note in its own track, then *merge* those tracks into the chord we want.

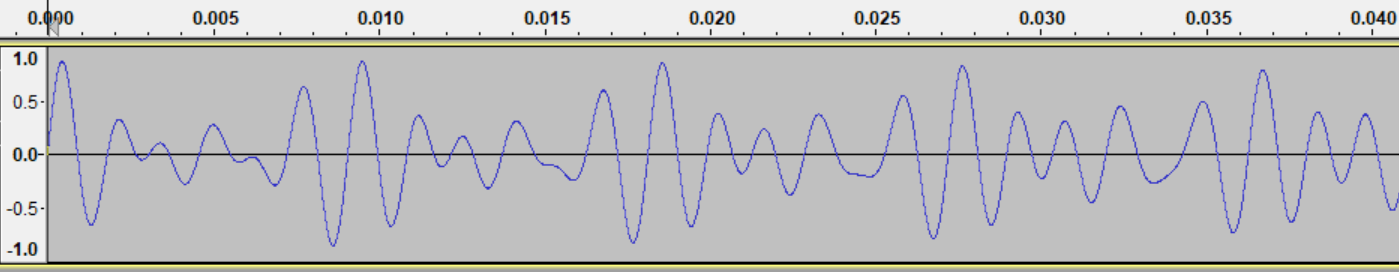
1. Start Audacity and begin with a clean slate
2. Select “Tracks | Add New | Mono Track”
3. Select “”Generate | Tone” and generate an *A*, i.e. create a 440 Hz tone at 0.3 amplitude.
4. Again, select “Tracks | Add New | Mono Track”
5. Select “Generate | Tone” and generate a *C#*, i.e. create a 550 Hz tone, again with 0.3 amplitude
6. Once more, select “Tracks | Add New | Mono Track”
7. Select “Generate | Tone” and generate an *E*, i.e. create a 659 Hz tone, again with 0.3 amplitude

Note that the amplitude of a wave determines how loud the corresponding sound is: large amplitude means loud sound, small amplitude gives a soft sound. Since Audacity normalizes the amplitude to be between +1 and -1, we pick 0.3 for each of our three waves so that the combined wave will have an amplitude of 0.9 or less. If you try to play a wave with amplitude bigger than 1, it will sound distorted.

Finally, we want to superposition the three notes into a single chord: hit CTRL-A to select all three tracks, then pick “Tracks | Mix and Render to New Track”. This will mix the three tracks together and show the result in a fourth track.

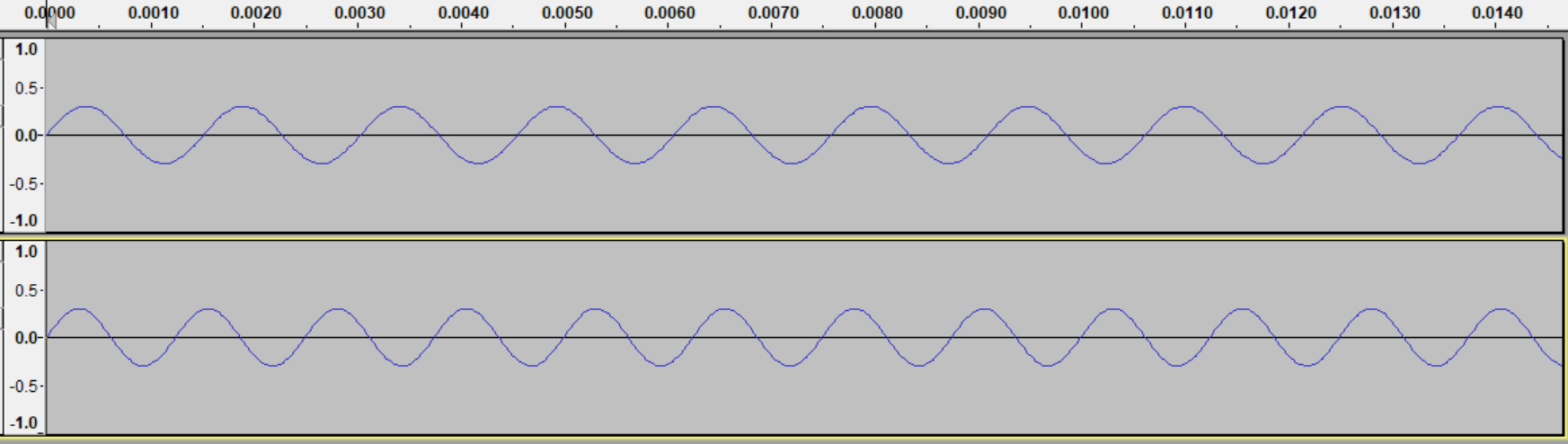
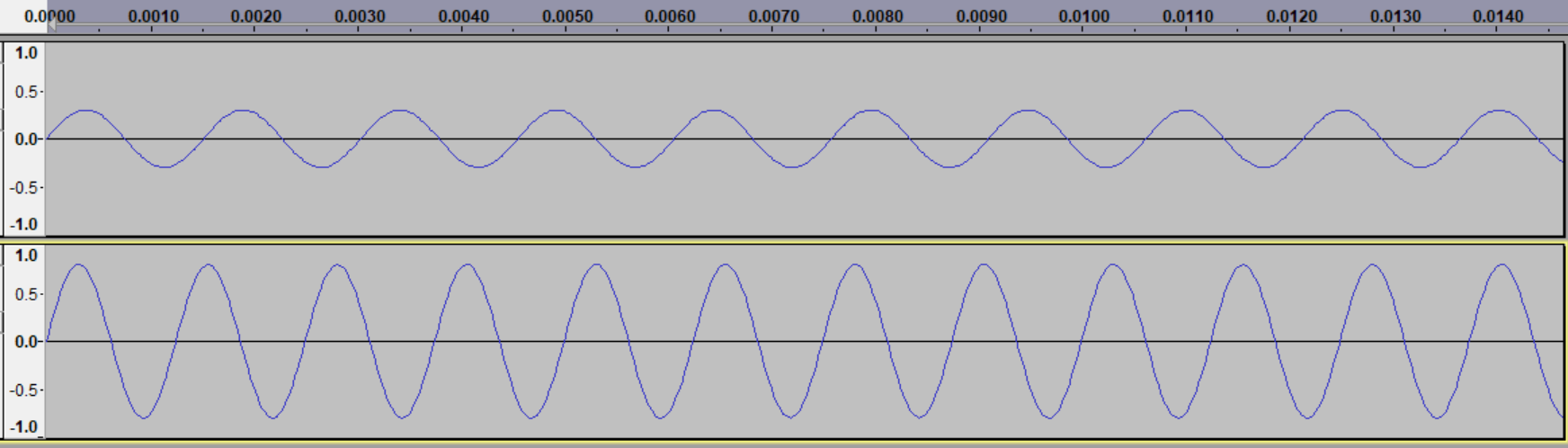
Now let’s verify that we generated the correct chord and then let’s listen to it: select the fourth track and zoom in by pressing CTRL-1 repeatedly (or CTRL-3 to zoom out) until you see the wave as a single curve. Press the HOME key to move to the beginning of the wave and compare it with the graph we generated with Mathematica for time to . Finally, listen to the combined wave to hear a *A – C# - E* chord: select the fourth track, click the ‘solo’ button at the beginning of the track, and hit “Play” (or spacebar)

.



Note that the chord sounds pretty “computerized”. That is because each tone consisted of a pure sine wave at the appropriate Hz. Real sounds, such as an A on a piano or guitar also consists of wave at 440 Hz but they include many other frequencies that give the tone its characteristic ‘piano’ or ‘guitar’ sound (see exercises)

**Exercises:**

1. Use Audacity and follow the above examples to generate a *C–E–G* chord. Confirm the shape of your sound wave using Mathematica.
2. Suppose we have two sounds as shown. Which one will be higher? Estimate the frequency of each sound.  
   
3. Suppose we have two sounds as shown. Which one will be louder?   
   
4. How could you make a sound louder? How could you make an arbitrary sound “fade in” and/or “fade out”? Apply the “Fade In” and/or the “Fade Out” effects to a chord to verify your suggestion.
5. Suppose you have a wave representing a C–E–G chord. How can you transpose the chord into a A-C#-E chord? Audacity has a tool to change the pitch under “Effects | Change Pitch …”. Use it to change the C-E-G chord you created earlier into an A-C#-E chord. Play both sounds to compare them, and compare the shape of the wave.
6. The “Change Pitch” effect you applied in problem (4) has a side effect that changes the beginning of the converted sound. Describe that side effect.
7. If you want to change a C to an E, say, you need to change the frequency by 3 half-notes, or multiply it by . Why does it not work to simply replace x by to change the 3-note chord C-E-G to E-G#-B, or to transpose an entire song by 3 half-notes that way. How, then, could the “change-pitch” effect introduced in (4) work?
8. We have seen how to merge multiple tracks into a new track by adding the wave functions. Do you think it is possible to decompose a complicated song into its Fourier components? For example, say I give you the wave form for a 3 note-chord, could you determine which three notes the sound consists of?