FOURIER WITHOUT FORMULAS

David Weenink

University of Amsterdam
david.weenink@uva.nl

ABSTRACT

We describe a computer application based on the program Praat[2] that offers teachers and students the possibility to get more insight into Fourier synthesis and -analysis without using formulas. In the application the Fourier components are represented as pure tones. All kinds of relations between sums of pure tones can be synthesized, visualized and listened to. The number of tones as well as their frequencies and phases can be varied at will. The program, among others things, makes it straightforward to demonstrate visually and audibly that sums of harmonically related tones always generate periodic sounds, and sums of non-harmonically related tones do not. As a link to the importance of the amplitude spectrum in speech analysis, the ear's insensitivity to phase can be demonstrated by generating an infinite number of different wave forms that nevertheless all sound the same.

Keywords: interactive phonetic education, periodicity, spectrum, sine function, speech synthesis, speech analysis.

1. INTRODUCTION

Explaining to students of linguistics the elements of spectral analysis and synthesis is not easy. A lot of terminology from physics and mathematics is generally needed which is outside the domain of linguistics. Fourier applications on the internet usually are too technical or do not connect to perception; some examples are [1, 3, 4, 5, 6]. Our application improves on what is available by always having a link to perception. It uses sounds in order to facilitate the insight into spectral analysis and synthesis and it doesn't need the formulas to do so. In the current version the application is implemented as a demo script which runs on top of the program Praat[2]. The application's main function is to facilitate the understanding of synthesis and analysis of complex sounds from simple sinusoidal components by linking them to perception. Before we describe its possibilities, we first have to introduce its interface.

2. THE GENERAL LAYOUT OF THE INTERFACE

Figure 1: Initial view.

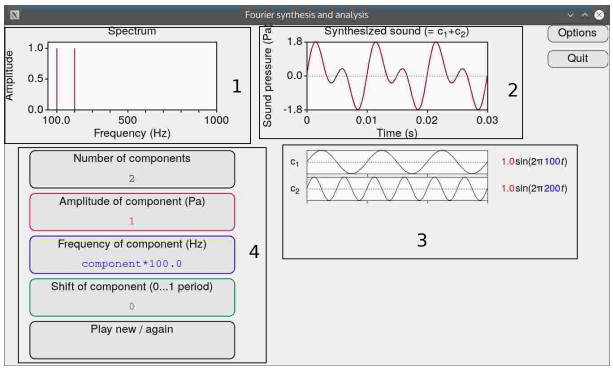


Fig. 1 shows the program's main window, where the important parts of its interface have been outlined with four rectangular boxes numbered 1 to 4.

Box 2, at the top right, is the central part of the interface as it shows the synthesized sound. With default settings the first 0.03 s of a periodic sound of 0.5 s duration are shown. This happens to be exactly three periods. A left mouse click in the sound rectangle plays the total synthesized sound of 0.5 s duration. Playing only 0.03 s of the sound would result in a short plop-like sound whose frequencies cannot be very accurately analyzed by our ears.[1] In the figure it is easy to see that each period lasts 0.01 s and that its fundamental frequency is 100 Hz. The vertical amplitude scale for this particular synthesized sound varies between -1.8 and +1.8.

The sound in box 2 has been synthesized from the two components which are displayed in box 3 below it. These two components are exactly lined up with the synthesized sound and have an amplitude scale which is not shown because it is always fixed between -1.0 and +1.0. At the right of each component we show the sine function of the corresponding component. For example for the first component it reads `1.0sin(2π100t)` and it can be easily verified from the figure that this formula describes a pure tone of 100 Hz because one period lasts 0.01 s. We think these formula's are almost inevitable as they

describe the form of each frequency component as a function of time while their frequency is also displayed (in blue colour because we use colour coding as will be explained further on). As was the case for the synthesized sound, a simple left mouse click in a component's rectangle plays the corresponding tone of 0.5 s duration. In summary, the boxes 2 and 3 together show Fourier synthesis.

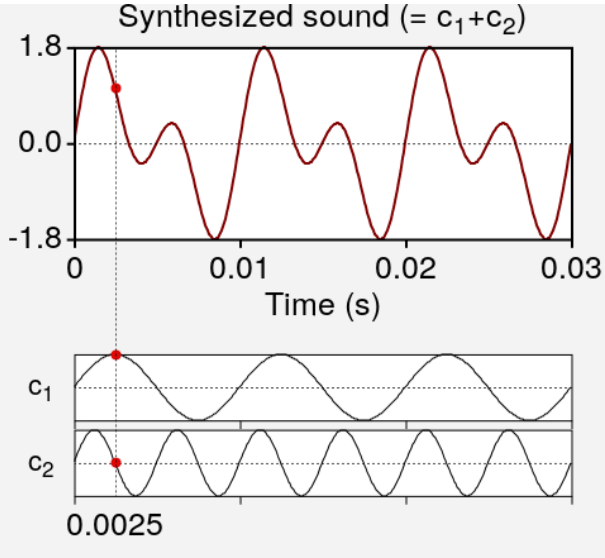
Box 1 shows the amplitude line spectrum of the sound displayed on a linear frequency scale as well as on a linear amplitude scale. We have limited the upper frequency to 1000 Hz for the same reason as we limited the view in box 2 to 0.03 s: the important things should be easy to see. We deliberately show the line spectrum in order not to confuse people since the real spectrum can have different appearances depending on whether the components do or do not "fit" in the duration of the sound. The vertical scale is linear to have a clearly visible relation with the corresponding amplitudes of the components in box 3. Box 1 and 2 together show Fourier analysis and therefore we have Fourier analysis *and* Fourier synthesis represented in the same interface. We will now explain the settings of the controls in box 4 and later extend the information about the parts we have just discussed.

3. THE SETTINGS OF THE CONTROLS

In the settings part of the program, box 4 in Fig. 1, five rounded gray rectangles are present, each one with a label on top. These labels read, from top to bottom, "Number of components", "Amplitude of component (Pa)", "Frequency of component (Hz)", "Shift of component" and "Play new / again". The top four are used to vary the four parameters needed to synthesize a sound from multiple sinusoidal components. They show below their label the current setting of the parameter, each in a different colour. We have used code colouring to relate the important parts in the interface: amplitudes are always shown in red, frequencies are shown in blue and phases are shown in green. Consequently, the outline of the rectangle controlling the frequencies of the components is drawn in blue, as well its current value and the frequency in the sine-formula of each component. Its amplitude is in red. A left mouse click in one of the top four rounded rectangles pops up a form which allows us to change the current value of the parameter. A left mouse click in the rounded rectangle at the bottom synthesizes a new sound according to the settings of the four controls above it and plays this new sound. In the next part we go into more details on the specification of the four synthesis parameters.

3.1. Varying "Number of components"

Figure 2: Interface detail. How two components add up at time 0.0025 s.



As the name suggests, this control offers the possibility to set the number of components used to synthesize a sound. In Fig. 1 the default number of components was set to only 2. In box 3 there is enough vertical space to display maximally eight components. If we choose more than eight components only the first seven components will be displayed separately and the accumulated contribution of the rest of the components will be displayed in the slot of the "eight" component.

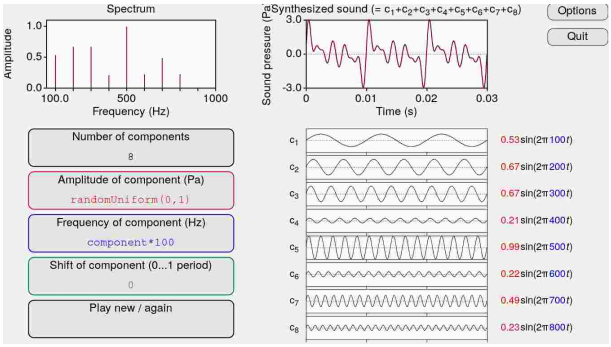
In Fig. 2 we show how synthesis works for a sound synthesized from two components: *synthesis is adding amplitudes of components time-point by time-point*. The text above the sound in box 2 shows that it is the sum of two components: "Synthesized sound (=c_1+c_2)". With a left mouse SHIFT-click in this sound, a vertical cursor hair will be shown that runs from the top of the sound in box 2 vertically down to the bottom of the highest component in box 3. Here the current time at the cross hair is displayed. At the position where the cursor hair crosses a sound's amplitude function, a red dot will be displayed (we don't display the actual amplitude values as this would clutter the interface). In the figure there are three of these dots: the amplitude of the synthesized sound at the cursor hair is the sum of the amplitudes of the components at the corresponding red dot positions. This is easier to see if we only use two components. By SHIFT-clicking at a suit-

able number of different time points in the sound we can show that the amplitudes always add up. In this way we can demonstrate that Fourier synthesis is all about adding up components.

3.2. Varying "Frequency of component"

The default setting for "Frequency of component (Hz)" is `component*100.0`. This setting has the effect that the frequency of each component is calculated as a multiplication of its component number and 100. The first component which has component number 1 will therefore have a frequency of 100 Hz (= 1 * 100), the second component will have a frequency of 200 Hz (= 2 * 100), and so on. The resulting synthesized sound will always be periodic with a period of 0.01 s. As an example of this examen Fig. 1 where two components were synthesized. Now, if we want to demonstrate that a periodic sound can only be synthesized from harmonically related components, we have to show the following two things:

Figure 3: The sum of harmonically related components is always a periodic sound.



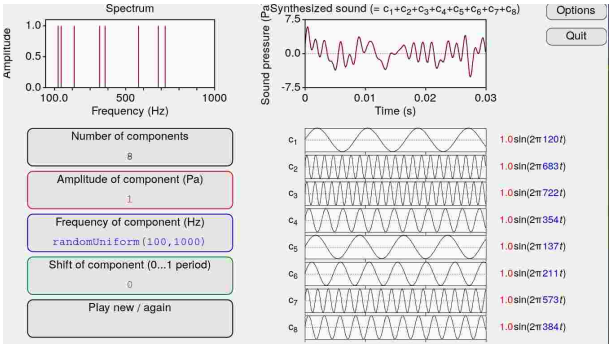
1. adding harmonically related tones always results in a periodic sound, and,
2. adding non-harmonically related components does not result in a periodic sound.

To start with the first point, the default setting for the "Frequency of component Hz)" as `component*100` suffices. Eight components of frequencies from 100 to 800 Hz will synthesize a periodic sound with a fundamental frequency of 100 Hz. If we change the setting to `component*150`, the synthesized sound will have a fundamental frequency of 150 Hz while the components will have frequencies which are multiples of 150 Hz (i.e. 150, 300, ..., 1200). Whatever we change the number after `component` to, the synthesized sound will always be periodic. One could argue that this happens because the amplitudes of the components are always equal to one. We can parry this objection by generating components whose amplitudes vary randomly as is demonstrated in Fig. 3. Here we have chosen `randomUniform(0,1)` as the setting for "Amplitude of component (Pa)" with the frequencies of the 8 components still being multiples of 100 Hz. The effect of this amplitude setting is that for each component the amplitude will be a random number from the interval [0,1). Each value in the [0,1) interval has equal probability of being drawn. For example, the number drawn for the first component, rounded down to two significant digits, happened to be 0.53 while the amplitude for the 8^{th} component happened to be 0.23. These amplitudes are also clearly shown in the spectrum. The nice thing of this setting is that every time you now click the "Play new / Again" button a *different* sound will be synthesized because of this `randomUniform` argument. It is therefore possible to generate an infinite number of sounds with the same fundamental frequency, i.e. the same pitch, that sound and look different. This makes it very easy to demonstrate that the timbre of a sound depends on the amplitude relations of its components.

Changing the frequency setting to `(component+1)*100` makes it easy to demonstrate that the first harmonic is not even necessary for pitch perception. Because of this setting, the first component will now have a frequency of 200 Hz ((1+1)*100).

The second point above, adding random frequency components, can be demonstrated by choosing for example `randomUniform(100,1000)` for the frequency setting. For each component's frequency a random value between 100 and 1000 is picked with equal probability. An example of this is shown in Fig. 4. Amplitudes were all 1.0 as is shown by the spectrum as well as by the numbers in front of the sine function on the right of each component. The components in box 3 are shown in order of component number. The first component happened to have a frequency of 120 Hz (rounded down to three significant digits). The highest frequency, 722 Hz, happened for component 3. In the synthesized sound it is hard to detect any periodicity. Each time we click the "Play new/ again" button a new synthesized sound is played with non-harmonically related frequencies and no detectable periodicity. It will not be too difficult to convince an audience that there will be no periodicity in the synthesized sound however often you click. A proof, of course, can only be given using the proper mathematics.

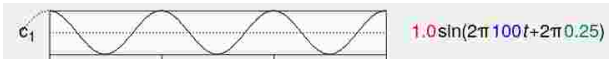
Figure 4: The sum of non-harmonically related components is not a periodic sound.



3.3. Varying "Shift of component"

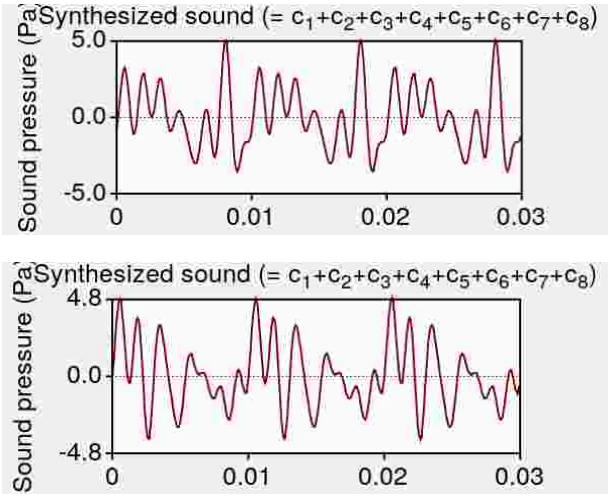
Figures 1 to 4 show that the amplitudes of the displayed synthesized sounds as well as their components all start at zero. This was a necessity because we synthesized with pure sines and they start with zero amplitude at time zero. It does, however, limit our synthesis possibilities. If we want to synthesize sounds that do not start at zero amplitude, we need sines that do not start at zero at time zero. By shifting one or more of the sine components to the left (or to the right) we can create sounds that do not start at zero amplitude. Because the components are periodic, we can attain all possible amplitudes at the start by a shift over maximally one period. Fig. 5 shows the idea for the first component. The setting is a shift of $\frac{1}{4}$ (of a period). In the formula at the right side this shift is multiplied by 2π and this results in the value for the *phase* of the component (displayed in green colour). Normally only the component's part within the rectangle is displayed. However, by clicking on the green number in the formula on its right, the shift is explicitly shown by the dotted part of the sine curve at the left of the starting time (a quarter period in the example). It is easy to demonstrate that setting a

Figure 5: Interface detail. The shift of a component by $\frac{1}{4}$ of a period.



constant shift for all components results in another synthesized sound, although the spectrum doesn't change and also the sound stays the same. This is already a hint that the ear is not sensitive to shifts of the individual components of a sound. However we can demonstrate this much better, faster and more convincingly by using the following settings for the phase: `randomUniform(0,1)`. Together with, say 8 components, amplitudes `1` and a frequency setting of `component*100` one can generate with each click of the "Play new / again" button a synthesized periodic sound which looks different from the previous ones but nevertheless sounds the same. Fig. 3.3 shows two periodic sounds that despite their very different form sound the same. Because the amplitude spectra of these sounds are all the same, it shows that the spectrum covers the essential aspects of human speech perception.

Figure 6: Interface details. Two synthesized sounds that sound the same.



3.4. Varying "Amplitude of component"

In the previous section we already introduced random uniform amplitude variations. Amplitudes can also be varied depending on component number. All kinds of Fourier sums can be synthesized, for example, a sawtooth by using `(-1)^(component+1)/component` for the amplitudes and `component*100` for the frequencies, where the ^ stands for exponentiation.

4. DISCUSSION

We have presented an application that makes it relatively easy to show both Fourier analysis and synthesis as well as their links to perception.

5. REFERENCES

[1] Adams, W. K. 2011. Fourier: Making waves - An interactive simulation for visualising Fourier analysis. *The Journal of the Acoustical Society of America* 2396.

[2] Boersma, P., Weenink, D. Praat: doing phonetics by computer [Computer program]. Version 6.0.48, re-

trieved 17 february 2019 from http://www.praat.org/.
[3] Efstatiou, C. E. http://195.134.76.37/applets/AppletFourier/Appl_Fourier2.html. [Online; accessed 23 February 2019].
[4] Falstad, P. http://www.falstad.com/fourier/. [Online; accessed 23 February 2019].
[5] Wikipedia contributors. Fourier analysis. https://en.wikipedia.org/wiki/Fourier_analysis. [Online; accessed 23 February 2019].
[6] Wolfram contributors. http://mathworld.wolfram.com/FourierSeries.html. [Online; accessed 23 February 2019].