# ‘Additive Synthesis’

SIGNALS & SYSTEMS IN MUSIC
CREATED BY P. MEASE, 2011

## OBJECTIVES

In this lab, you will construct your very first synthesizer… using only pure sinusoids! This will give you first-hand experience with how the Fourier Series works.

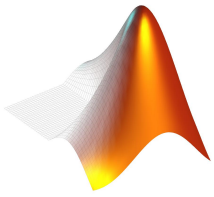
## BACKGROUND

Additive Synthesis, as its name suggests, is a process by which single harmonic components are *added* to create a signal that is harmonically richer than the single components that make it up. Additive synthesis is just one of many methods for synthesizing sound. Other types exist, such as: Subtractive, Granular, Wavetable/FM synthesis; each with specific methodologies for synthesizing sound. We start with additive because it is the most pure form and the most elementary step in understanding the principles of synthesis.

There are many types of additive ‘synthesizers;’ one of the most interesting is not electronic in form: the pipe organ. This massive instrument can be found in many older churches and are typically permanent installations. An example, the “orgel der Severikirche,” can be seen in Figure 1.

Figure 1. orgel der Severikirche

The instrument is comprised of a large array of tuned pipes (some organs have upwards of 20,000 pipes!), each generating a closely sinusoidal waveform. The organist has a control console, which diverts air through a defined set of pipes to change the timbre of the sound. Since we don't want to machine and manually tune a few thousand large pipes to create the harmonic content of the notes we play, we instead use a computer program called Matlab to create and add the sinusoids for us on the computer. Today, most synthesis is performed on computers or dedicated DSPs.



Matlab is a high-level programming language, meaning we are far removed from the machine code that the computer will actually see. This is good, since humans cannot directly code machine code. We are also using Matlab because it is even *higher*-level than more common languages like Java, C/C++,etc… The learning curve will be less. Matlab is pretty easy once you know its basics.

Matlab has two main windows you will work with: the Command Window and the Script Window (formerly called an 'm-file'). These two windows both take the same code, the only difference is that the command window cannot save your code… it is primarily used for

running/testing a small snippet of code and for viewing code outputs. The Matlab script is a file that you can write code in, but now you can save the file so that you can run your code later. This is the preferred place to write your code.

Before we can continue, we should mention a process called 'sampling.' Sampling a signal is when we take only certain amplitude values from it and store them. A continuous 'analog' signal is sampled and becomes *discrete*. It is no longer continuous and *only* exists exact at the moments we've taken that sample. Matlab (or any computer for that matter) ONLY deals with discrete signals. The way we can make them *seem* continuous is to sample them fast enough where the adjacent points are very close to one another. The reason we sample, or discretize, a signal is simple: we have to! Continuous waveforms take up infinite space in computer memory. Why? Because continuous signals have an infinite amount of points (remember, there are NO discontinuities). If we want to manipulate or build signals using a computer, we have to make sure it fits in the memory. Furthermore, we also don't want to use all of the computer's memory for a simple waveform. We want to have just enough of the waveform represented (as samples) so that it is a *good enough* representation of the original continuous waveform. Which brings us to a warning:

If we sample too slow (the samples are far apart), we can start to loose the original signal to the point where it is not longer a good representation of the original signal. Figures 2, 3, and 4 display a 1kHz sinusoid sampled at 10kHz, 40kHz, and 5MHz, respectively. These are plotted using the >> stem(.) command so you can see the individual sample 'dots.' Note how the sinusoid gets smoother the higher fs approaches.

A quick 'n crude rule is to make sure you sample at a rate faster than about 5-10 times the HIGHEST frequency in the signal (the actual bare minimum is defined by what is called the Nyquist frequency… but you can read about that on your own (HINT: look at the sample rate of CD-quality audio)). We will be going over all of this again in-class so if it's not 100% clear, it's OK.

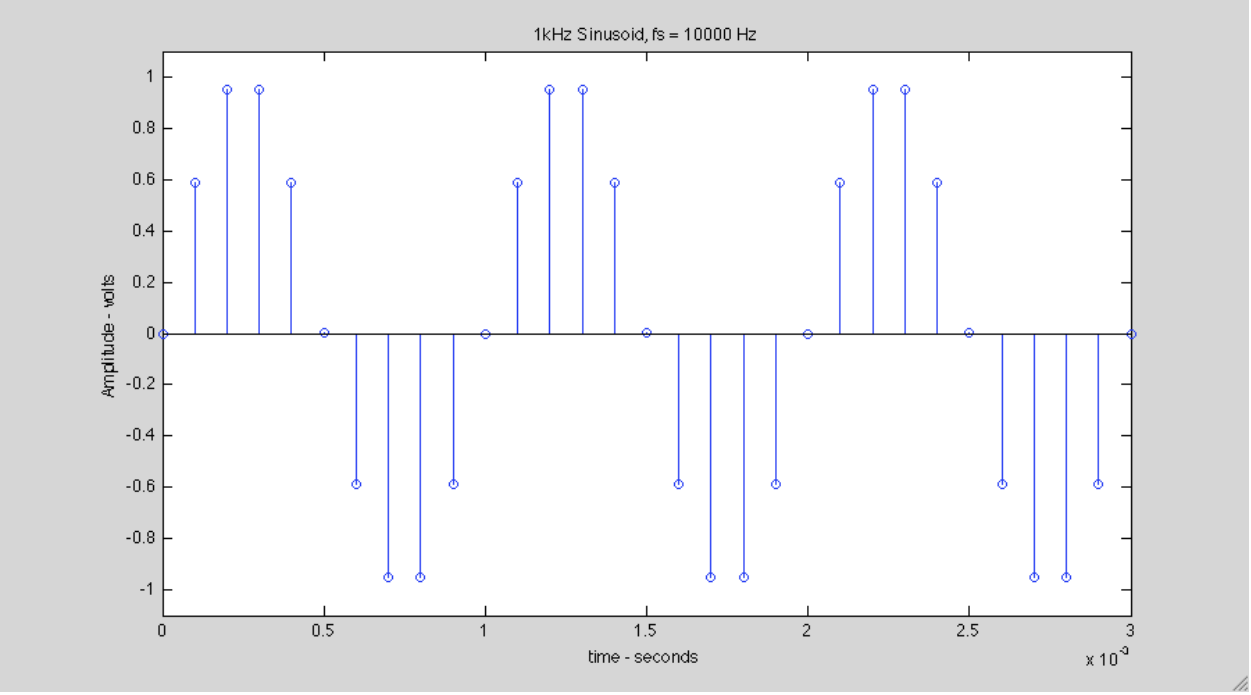



Figure 2. fs = 10kHz.

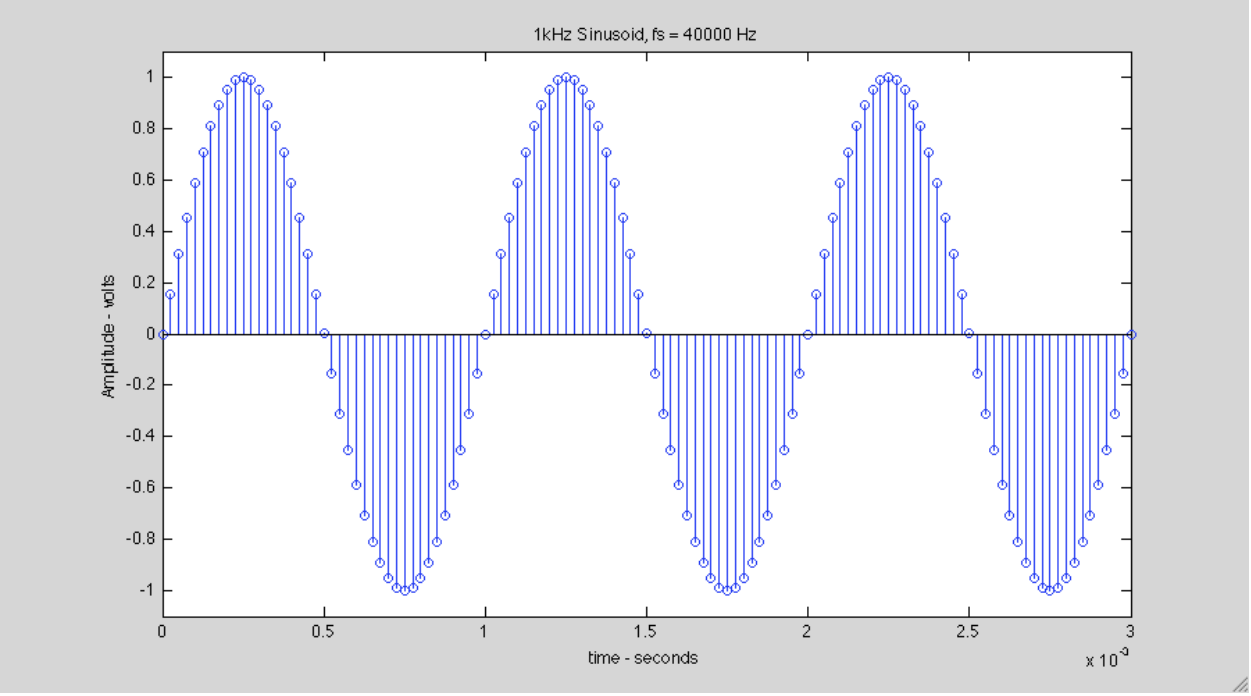



Figure 3. fs = 40kHz.

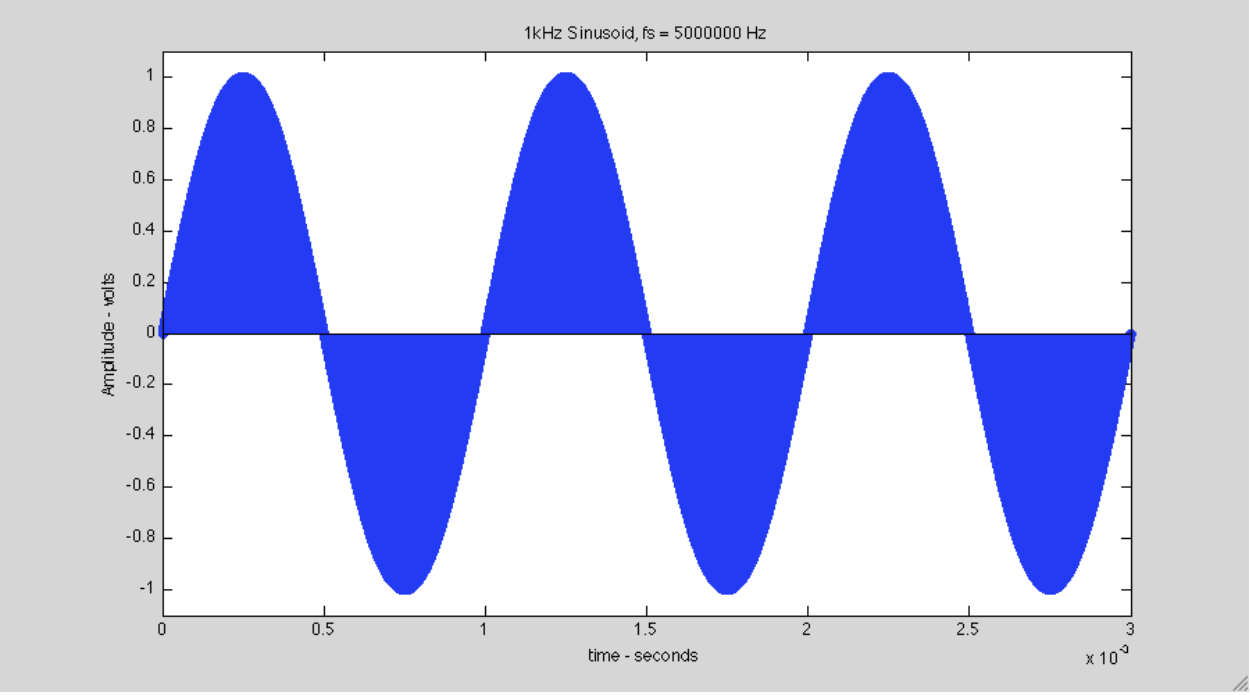



Figure 3. fs = 5MHz.

Now onto some important things to note when programming in Matlab:

Arrays are just single-row (or column) 'lists' of data. Arrays are matrices with one dimension being 1. Arrays can also be called vectors. You can create your own single-row array in Matlab by typing in:

```
>> X = [0 1 4819 0 -21.2 9 12.209418]
```

NOTE: for all code, the >> only signifies that it is actual code… you wouldn't actually type the '>>'.

This array is stored in a variable we choose called "X" and it has a size of 1x7 meaning it has one row and 7 columns. Note that different row elements are separated by a space or, if you want, you could also use commas (they are the same). If you now wanted a 7x1 array, you would type:

```
>> X = [1; 3; 538.2; -1; 109; pi; 0];
```

Note that the semicolons denote new columns. Also note that if we ran these two arrays, the second 7x1 array would OVERWRITE the first X. This is because we named them both "X." If you were looking carefully, you also see that I put a ; at the end of the line. This will simply suppress this array from being seen in the command window – nothing else! So if you don't

want to see your array outputted visually to the command window, be sure to end the line with a semicolon. The semicolon does not prevent the array from being written to memory.

OK, so for our experiments we will have to create several sinusoids. The equation for a sinusoid is:

```
>> S = A*sin(2*pi*f*t – phi);
```

We can ignore phi (the phase term) for now so it is just:

```
>> S = A*sin(2*pi*f*t);
```

In the above, pi is a reserved variable name for pi (which is the 3.1415… number we all know of). This is a constant, f is the frequency you want the sinusoid to oscillate at (in Hz), and t is the time vector (or array) to have S evaluated over. It is time. A is the amplitude you want the sinusoid to be.

So, we can easily pick A and f (since we know what frequency and amplitude we want the sinusoid to be) but what about t?! Without a value of t, we cannot get a value for S, right?! RIGHT! So what value(s) do we use for t? Well, t is in seconds and is the variable that we must 'plug in numbers' in order to get results from our sinusoid equation. Now back to sampling. If we wanted a perfectly continuous sinusoid, we would have to solve S for ALL possible values of t between the time range we want the sinusoid to play. Let's say we want it to play for 5 seconds… we'd need an infinite amount of 't's' to fill it. Even a 0.001 second long sinusoid would take infinite amount of 't's' to fill. So. We obviously cannot create or store an infinite amount of points on a computer as it would run out of memory and probably crash. If we say: 'every so often create a t value for S to be evaluated' then we can control how many points (or samples) make up S. So let's think about this. We know f of our sinusoid and we know that we need to sample at a rate waaaay higher than this (again 2-10+ times should work)… so it would be logical to pick a sample rate that was, let's say 10 times our f. If we wanted a 1kHz sinusoid, we'd have to sample at 10*1kHz, which is 10kHz. Since Ts = 1/fs, we can find the period of our sampling frequency, which is just Ts = 1/10,000 or 0.0001s. Ts is the space (in time) between two adjacent samples and we can call this the sampling interval. The smaller the interval, the more continuous our signal will look but the more samples are required to store in memory. Please note that fs (our sampling frequency) is NOT f, our sinusoid frequency. Read that a few more times and things will settle a little better.

That was long. OK, so we figured out that we need to plug in many t's to get many data points for the sinusoid for however long we want to play it for. Matlab has a very easy way to make a vector (or array) of evenly-spaced values. We can call this the 'time-sweep' vector and we can create one by:

t = startValue:increment:stopValue

where startValue is the time (in seconds) we want to start the sinusoid. For most applications, you can just pick 0s. increment is the space between two adjacent values and is exactly equivalent to Ts (see above), our sampling period. stopValue is where we want the sinusoid to stop playing. Again, this is in seconds and for us, we want at least a few seconds to play so we can hear it. Let's pick 5 seconds.

```
>> t = [0:1/10000:5]
```

Try typing this in just the command window and look at the values t returns (once you're satisfied and don't need to see the output in the command window, be sure to add the semicolon at the end of the line). Soooo… to make a sinusoid with frequency 1kHz, amplitude 1V, lasts for 5 seconds, and is sampled at 10kHz we would type the following two lines:

```
>> t = [0:1/10000:5];

>> S = A*sin(2*pi*f*t);
```

The 't' statement has to be placed BEFORE the 'S' statement since S NEEDS its 't' to be some values in order to evaluate. If it were the opposite order, t would be undefined and throw an error. In the above statements, t is a huge array of values. Since Matlab is awesome, it will automatically create a value of S for EVERY value of t and put all of these results into S. So by making t a vector of values, S is now a vector of values representing the amplitudes of the sinusoid. Now that we have found both t (time) and S (amplitude), we have all we need. To see this waveform, we can plot amplitude vs time like:

```
>> plot(t,S)
```

It may be hard to see this waveform since there are so many cycles in 5 seconds worth, so try out the zooming tools in the plot window. You may also just reduce the stopTime to a few periods for viewing purposes. Your call. Also note that there are no axis labels. This is not acceptable. To add axis labels in Matlab just write (after the plot command):

```
>> xlabel('time – seconds');
>> ylabel('amplitude – volts');
>> title('1kHz sinusoid, fs = 10kHz');
```

Excellent. We can now make any sinusoid at any frequency and at any amplitude. This is a good thing.

Let's talk about sampling rate once more… I mentioned previously that you need to sample your signal at a rate of at least twice the highest frequency. Since we're going to be adding up

sinusoidal harmonics all the way up to the upper limit of hearing (which is 20kHz), let's FIX our sampling frequency, fs, to one single value. Let's pick CD quality, which is 44.1kHz (sound familiar?!). The reason we will use the same fs for ALL sinusoids is because they all have to be the same length to add 'em up. Again, to add them, they ALL have to be exactly the same length!

Now let's talk about something called normalization and, I promise, this will be all the background you'll need to make your synthesizer.

Normalization is the process of scaling data such that it fits into a certain range. Simple normalization is typically done by multiplying or dividing by a constant, which means the signal really doesn't change, it just gets smaller or larger. The ratios that exist between the data remain the same.

We will need to use normalization for two purposes in this lab:

1) Normalizing harmonic amplitudes
2) Normalizing summed waveform to fit within -1 to 1V amplitude to avoid clipping.

We will talk about this process in class (take notes!).

A few other Matlab tips:

To find how to use ANY function in Matlab just type in the command window:

>> help *functionnamehere*

and hit enter.

## EQUIPMENT & SOFTWARE

Headphones/Speakers
Laptop/PC
Data from Part 2 of In the Harmonics
Matlab
Breakout Card
3.5mm M-M stereo cable

## PROCEDURE

- Open Matlab, then open a new script.
- Create your time-vector. It should range from 0s to 5s.

- Now create the sinusoids for each harmonic you found in the previous lab. You will need both the amplitude and frequency data for each.
- Now add em up!
    - Plot this guy
- Now normalize the resulting signal so it fits in the amplitude range -1 to 1 to avoid clipping. This kills two birds with one stone since it ends up normalizing the harmonic amplitudes and the final signal.
    - Plot it to make sure your normalization worked.
- Plot and play the resulting waveform using the sound(.) command in Matlab.
    - Type >> help sound to find out how to use it.
- Play the signal through the oscilloscope and capture its spectrum.
    - To save your audio in Matlab look up: wavwrite
- Be sure to save your work.

## DELIVERABLES

- Formal Lab Report (see lab report format)
- Matlab code used to create your synth (place in Appendix of your report).
- Matlab figure (plot) of your signal (time-domain).
- Purposely undersample a 1V, 1kHz sinusoid and plot: What happens if you sample exactly at 1kHz? What happens if you sample at 2kHz? What happens when you sample at 4kHz? What happens when you sample at 10kHz? (Be sure you look at the value of the amplitude axis to compare equally!) Use plots to justify and explain why sampling is critical. TIP: using the command >> stem instead of >> plot in Matlab will keep Matlab from 'connecting the dots' between points. Stem plots will allow you to precisely see where the samples are located.
- If you know a signal is periodic, how many periods of the signal do you need to synthesize to reproduce it exactly?
- Spectrum (screenshot) of your final signal.
    - Is the spectrum close to the appearance of the real signal? Explain.
- Explanation of the differences (if any) of how the real and the synthesized signals sound. Why are they not exact? Use your spectrum screenshots to compare and justify your statements. Relate what you see in the spectrum domain to what you hear in the real signal and the synthesized real signal.
- Brief summary of contributions by each team member and out-of-class meeting time(s).

- EXTRA CREDIT: Modify the sinusoids in Matlab to make the signal more realistic sounding. HINT: you may want to remove measurement errors. Also, many harmonics end up being small integer ratios off the fundamental. You also know EXACTLY what the fundamental frequency is (exploit this). Again, you can use this to 'fix' your sinusoids before you add them, but hey, that's getting fancy.

Be sure to spend ample time discussing both results and the solutions with your entire team. Be thorough and precise in your statements. If you have any questions, please ask before submitting. DO NOT MISS ANY ITEMS IN THE DELIVERABLES SECTION.

## SAFETY & LAB PROTOCOL

- Be sure to turn down any headphone volumes BEFORE putting them on your head!
- Wear earplugs when dealing with high SPLs
- Heed all warnings above
- Return all cabling neatly to the racks
- Clean your workspace when finished your experiment
- No food or drink allowed in the lab
- Use safety glasses when required