

Introduction to Signal Processing Using MATLAB and Simulink

Bruno Korst - bkf@comm.utoronto.ca, David Ding - davidy.ding@mail.utoronto.ca

Abstract

This experiment will introduce you to `Matlab` and `Simulink`, simulation platforms/tools that will help you to accomplish the experimental tasks of ECE216. Keep in mind that the tools presented here are useful across many courses in Electrical and Computer Engineering. In the steps below we will explore the simulation of different signal sources, different displays and signal operations.

Contents

Introduction	1
1 Matlab	2
1.1 Creating and Plotting a Sinusoid	2
1.2 Listening to a Sine Wave	3
1.3 Audio Signal Processing—A Musical Preview	3
2 Simulink	4
2.1 First Simulink Model	5
2.2 The Four Operations (+ − ∗ /)	7
Adding or Subtracting a Constant to Sinusoid • Gain (call it “Multiplication by a Scalar”) • Operating on Two Sinusoids, Making Them Sound via Speaker • Multiplying Two Sinusoids	
3 Conclusion	9
Acknowledgments	9
References	9

IMPORTANT

To submit your assignment for this lab, you must upload a SINGLE ZIP file which includes the completed PDF answer sheet (with associated screenshots and answers you are asked to provide in the instructions), and Matlab /Simulink files you used.

Introduction

As a second year Engineering student, you have been introduced to a variety of tools to handle the theoretical aspects of science (using mathematics, physics, and algebra) and its applied side (using programming languages, circuits and some design). The latter, not surprisingly, is primarily what Engineers do. Yes, Engineers with capital E.

You have been introduced in ECE221 (Electricity and Magnetism) to `Matlab`, a numerical simulation package. At this point you have already observed that `Matlab` looks and feels like a programming language. It comes across as a tool that sits on the border between the theoretical and the applied sciences. You can use it to plot a variety of theoretical curves, or change variables in equations to observe outcomes dynamically. As you will see in the future, you will be able to create “inputs” to equations in the form of curves, and plot “outputs”, as if the behaviour of a system under test were described by your equation.

Your introduction to `Matlab` was in the form of individual commands entered at the prompt, functions written and called from the command line, or full programs run within `Matlab`. In this experiment you will expand your `Matlab` experience

further. First, by making `Matlab` interact with the outside world (i.e., with the file system and the speaker/headphone output on your workstation), and secondly by using `Simulink`.

If you saw `Matlab` as a numerical computation environment, `Simulink` is its visual counterpart, in which you have control of when things start and stop, you can define inputs, observe outputs and determine the behaviour of various systems under test as a simulation model runs. You can think of the “running” as inputs changing along time and producing outputs according to a certain function. This will become clearer as you do the steps in this experiment.

At the end of this experiment, it is expected that you will be very familiar with:

- Making `Matlab` talk to the outside world;
- Building simple simulation models in `Simulink`;
- Generating and displaying simulated signals;
- Making `Simulink` interact with the outside world.

As you work through this lab, there will be exercises for you to answer. Parts of an exercise are numbered (a),(b),(c), etc. For example, Exercise 1.2 (c) is part c of Exercise 1.2 in Section 1.2. Some exercises may only have part a. There is an associated answer sheet where you will fill in the answers to these exercises that needs submitted.

1. Matlab

Start up `Matlab` on your computer. If you use the computers in SF1013 lab, you can start up `Matlab` by clicking on “Applications”→“ECF”→“Matlab”.

NOTE: it is especially important that you set the current working folder of your lab session to a local folder (under your ECF account). We suggest you create a folder called “lab1” and save everything for this lab in this folder. That way, it will be easy to find files and you will not run into the issue of having your code not compile due to missing file errors.

1.1 Creating and Plotting a Sinusoid

In this section you will write a program to generate a 1KHz sinusoid and play it to the outside world (that is, your speakers/headphones). You can write your program on the `Matlab` text editor or can write individual lines at the command prompt. It is better, for editing, if you write in a program. If you do write as a program, don’t forget to use `.m` as the extension when naming your program file.

The program below generates 1 second of a 1KHz sinusoid using a sampling rate of 48000 samples per second. In other words, the output of this program is a vector with 48000 points, which represent a 1KHz sine wave.

```
T=1;
Fs = 48000;
N = T*Fs;
t = 0 : 1/Fs : T;
Fn = 1000;
y = sin(Fn*2*pi*t);
plot(t,y);
axis([0 48/48000 -1 1])
```

You can think of the output of this program as a sequence of values that follow a sinusoidal pattern. The values are spaced evenly, according to a time interval that you set. You can think of an *analog* as the one where the this time interval is infinitely small in-between samples. In *digital* signals, each of these pieces – or *samples* – of the sinusoid can be identified, as they happen at a specified time interval and are given a fixed value. In other words, they are taken “frequently” at a certain sampling rate (or sampling frequency), and are assigned a value following a sinusoidal function for this particular case.

Exercise 1.1:

- After having understood what the program does, your task is now to modify the program to plot three periods of your 1KHz sine wave. Save a screenshot with your plot and include it with your submission.

1.2 Listening to a Sine Wave

In the program above, you generated a 1KHz sine, but have you ever heard one? This is what you will do now. First, before everything, bring the volume on your workstation down to a low level (close to zero). This is accomplished using the audio controls. Now plug the headphones of your smartphone/mp3 player into the headphone jack of the workstation. When you change volume levels, the computer sounds an audible bell/chime as a reference to you. Make sure you can hear this chime when you change the audio level, and – most importantly – *be sure that the level is comfortable to your hearing*. Keep the level as low as possible, but with the chime still audible. You can keep the transducer (i.e., the earbud) at a short distance from your ear and still hear the signal.

The code below is almost the same as the previous one, but the last line `sound(y, Fs)` sends the sinusoid to the earphones.

```
T=1;
Fs = 48000;
N = T*Fs;
t = 0 : 1/Fs : T;
Fn = 1000;
y = sin(Fn*2*pi*t);
sound(y, Fs);
```

Exercise 1.2:

- Play the program and hear the 1KHz sine wave;
- Change the frequency to 500Hz and play it again;
- Now change it to hear 2KHz and play it again.
- Now change back to 10KHz and explain what you have observed/heard.

Save a screenshot of the waveform for each of the above and include it with your submission.

Before you perform the next exercise, it will be more comfortable if your frequency is set to 1KHz.

- Your task now is to double the volume of your sinusoid modifying one line of your code. Again, set your audio level in the computer to the bare minimum, modify and run your code. As a precaution, keep the transducer (the earbud) at a short distance from your ear, so that you will be able to hear the difference without hurting your hearing.

Your transducer is a coil (an inductor) that causes a membrane to vibrate proportionally to the change in voltage applied to the coil. To the output stage of the audio card in the computer, the transducer (or the coil) is a load. As you double the voltage, the membrane moves farther (*not more frequently!*), giving you the perception of a higher volume.

- As you doubled the voltage, what is the change in dB of the signal as measured at the load? Note that the dB you are calculating here *does not* represent sound pressure; you are only comparing voltages.

1.3 Audio Signal Processing—A Musical Preview

In this section you will get your feet wet by looking at how you can use Matlab to process real-world signals. Real-world signals are considered continuous-time signals, and in order to be processed digitally and be stored on the computer, they must be sampled and quantized. How fast the signal is sampled is called the sampling rate (or sampling frequency), and how many bits per sample is determined by number of bits in the quantization.

- First, download three ".wav" files from Quercus to your current working folder. They are "bass.wav", "drums.wav", and "guitar.wav".
- Next, you need to feed these real-world sounds into Matlab. In order to do so, there is a very useful function called "wavread" that takes in a ".wav" file and outputs the sampled data of the audio file, as follows:


```
[bass, fs, nbits] = wavread('bass.wav');
guitar = wavread('guitar.wav');
drums = wavread('drums.wav');
```

For newer Matlab versions please use "audioread" function instead of "wavread" function.

Each of the "bass", "guitar", and "drums" variables store a vector of sampled values of the corresponding continuous-time signals. If you look at the lengths of each of these sampled vectors (which indicates the number of sampled values), the numbers are huge! DO NOT display these sample values in the command window as it will take a long time to display all of them and therefore will likely freeze the screen. If you accidentally did this, press CTRL-C to stop the display (or wait for it to finish). For other audio formats, you will need to use the command "audioread".

Exercise 1.3:

- (a) Examine the length of the guitar signal (via: `length(guitar)`). The sampling rate is stored in the variable `fs`, and is in units of "samples per second". The value of "fs" is the same for all three sound signals. Using these facts, determine, in seconds, the duration of the guitar signal.

3. Copy and paste the following code into the editor or command window to create three sounds lasting 5 seconds each.

```
duration = 5;
b = bass(1:fs*duration);
g = guitar(1:fs*duration);
d = drums(1:fs*duration);
```

4. The Matlab function "sound" plays back the sampled audio signal. To use "sound", you need to feed it the audio data as well as specify a sampling rate, in this case, "fs".

```
sound(g, fs);
```

Now try to hear the "b" and "d" sound signals in a similar way.

- (b) Here comes the punchline: you can synthesize sounds by combining sound data digitally, as follows:

```
comp = b + g + d;
sound(comp, fs);
```

Play it. Save a screenshot of the waveform and the code and include it with your submission.

- (c) Now write a function or script that takes in "b", "g", and "d" audio data and outputs a five-second audio clip that includes all three of these instruments, but this time gradually increase the volume of the guitar sound. Show the audio clip and completed code. Save a screenshot of the waveform and the code and include it with your submission. Caution: signal amplitude gain should not exceed 5, or you will hurt your ears!

2. Simulink

You have now seen enough of Matlab to be able to use it as a tool in your studies. Now you will concentrate on Simulink, which is the visual simulation package that complements Matlab. Matlab and Simulink are complementary tools, and you can apply the same understanding and terminology you have from Matlab to Simulink.

When making Matlab interact with Simulink, the command window is still the window in which commands are written and parameters are set. If you set a parameter in a Simulink block, it will be visible to Matlab. When you change a parameter in Matlab and it is being used by a block in Simulink, it will be changed globally, and affect your block in Simulink. Keep this in mind.

The Simulink Library Browser icon is presented in Figure 1, and is found among the icons at the top of the main Matlab window you have opened. The icon is almost in the middle of the menu. Double-click on it now.



Figure 1. The Simulink Library Browser Icon

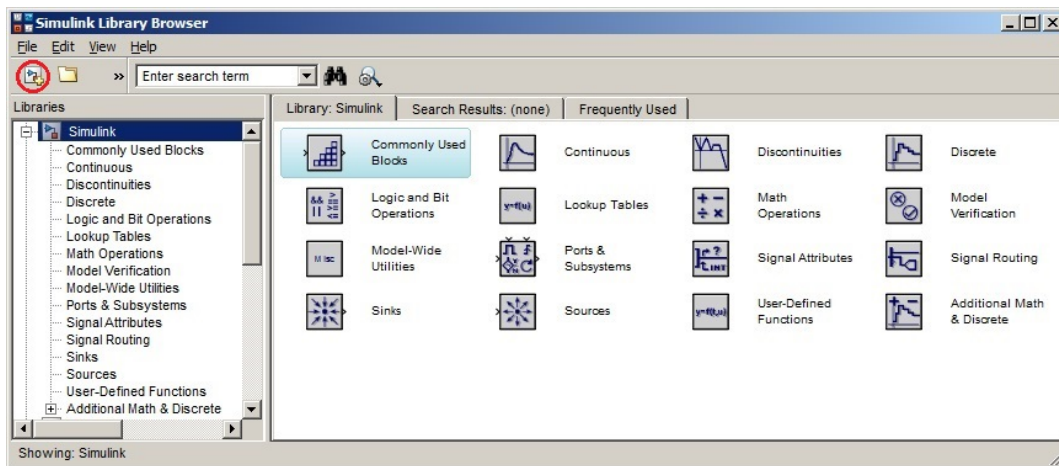


Figure 2. The Simulink Library Browser Window - New Model Icon Highlighted

This will open the Simulink library browser window, as presented in Figure 2. This is where you will open new and existing simulation models and where you will find all the blocks you will need throughout this experiment.

Upon inspecting this window, you will find not only the basic Simulink blocks, but also all the toolboxes installed in your workstation. For now, take some time and explore the subsets within the main Simulink block set. You can do that by clicking on their name and inspecting the blocks. You can double-click on the blocks to see the parameters, but since the block does not yet belong to a model, you will not be able to edit the parameters, just view them. Since you are in your second year of studies, you should be somewhat familiar with some of the blocks in the blocksets labeled Continuous, Logic and Bit Operations, Math Operations, Signal Attributes, Signal Routing, Sinks and Sources. This is to say that you have heard of many of these *names*, but perhaps never combined various of these parts into a large, more complex, system.

You may have also noticed that in Figure 2 there is one item circled. This is the icon you select to open a new model in Simulink. This is what you will start next.

2.1 First Simulink Model

From the Simulink library browser window, open a new model. You can do this in different ways: 1) by typing `Ctrl+N`; 2) by going under `File -- New -- Model`; or 3) by clicking on the icon highlighted in Figure 2. This will open an empty model. You will now click on desired blocks from the library browser you opened before and drag them to your empty model.

If you want to simulate a sine wave, you bring two blocks into your new model: the *source* called Sine Wave, and the *sink* called Scope. When the two of them are in the model, you connect the output of the Sine Wave to the input of the Scope. That is, you click on the output port and drag the arrow to the input port of the next block and let go. The output of one block goes into the input of the next block, just like in the lab. Now save the model and name it as you wish. When you are done, it should look like Figure 3 below.

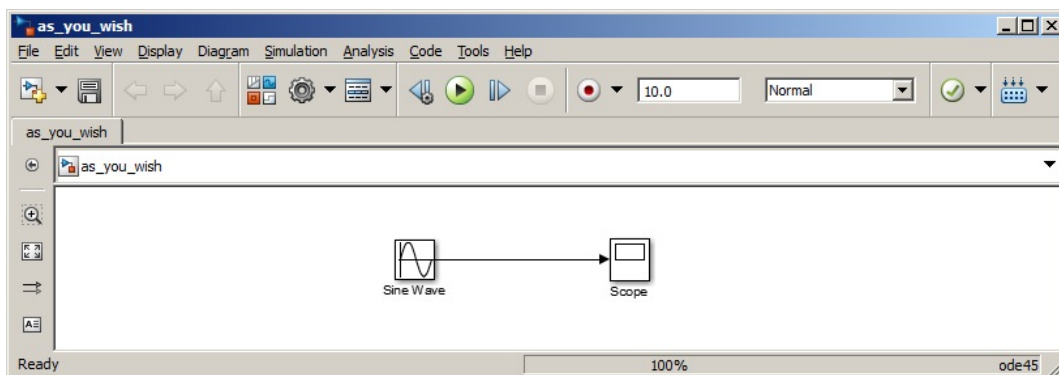


Figure 3. First Simulink Model: Sine Wave and Scope

We now want to simulate the system we have built. When we *simulate*, we are making things run along a time base. This time base is dictated by Simulink, but it is meant to represent “real” time, or seconds. The top menu of your model presents a space where you can determine your desired simulation time in seconds, and when you run the model it displays the simulated

time in seconds as well. If you time it on your watch, they may not match, but as far as your signals are concerned, all the time labels are in seconds (just like on an oscilloscope).

In 3 above, if you double-click on the sine wave block, you can define the frequency of your sine wave, in radians per second. So if you set your frequency to be $2 * \pi * 2$, and your simulation time to 1 sec, you should see two periods of a sine wave displayed on the oscilloscope.

Another way to set the parameters for the sine wave is to use the *sample base* type of simulation. When you use this type, you must set your frequency in samples per period, and you must define a sampling time. The latter is the interval between samples of the sine wave. For instance, mp3 songs ripped from CDs typically use a sampling rate of 44100 samples per second, which translates to a sampling *frequency* of 44.1KHz. The mp3 format allows for a maximum of 48000 samples per second.

Let's try it now. The objective is to display a 1KHz sine wave at a sampling rate for high audio quality. Set your sampling time to $1/48000$ to start (that is the "high audio quality"). Now you want to display a 1KHz sine, so how many samples are there per period? As you made these changes, note that the icon for your Sine Wave block changed automatically to show that you are now using samples. After you have set everything up, run the simulation for 1 second. Everything looks perfect... until you double click on the scope!

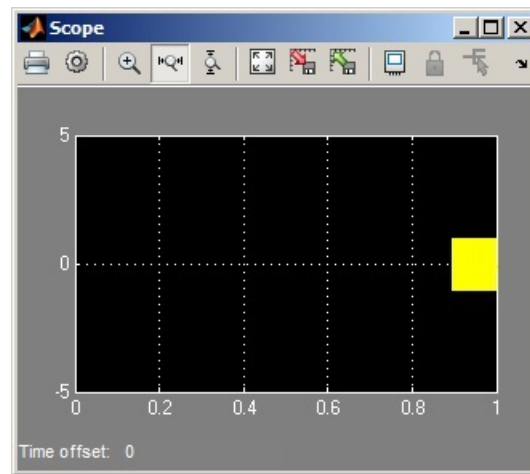


Figure 4. The Dreadful Yellow Blob

What you likely see when you click on the scope is the Dreadful Yellow Blob. How can you tell that this is, in fact, your 1KHz sine wave? You need to use the zooming function of the oscilloscope to achieve a meaningful display. The default setting is more confusing than helpful. Hover the mouse over the icons at the top of the scope display. You will find four tempting options: *zoom*, *zoom-X*, *zoom-Y* and *autoscale*. Resist your temptation to go for the *zoom* and *autoscale*. Instead, use *zoom-X* and *zoom-Y* to display your sine wave so that it does look like a sine wave, and then press on **save current axes settings**. The last step is key; if you do not save your axis settings, the display will return to the Dreadful Yellow Blob the next time you run your simulation. So now that you have saved your axes settings, run your simulation and see what happens.

Exercise 2.1:

- (a) When you are happy with your simulation, show your 1 KHz sine wave with $\frac{1}{48000}$ sampling. Save a screenshot of the waveform and include it with your submission.

This *sample base* type of signal is the one you will use to interface with the "outside world" during your simulations. This means collecting a signal from a file or from a microphone, or sending a signal to headphones. You have already done something similar from the command line using *Matlab* and you will do some more of that in this experiment.

It is worth keeping in mind that the amplitude of the signals presented in the simulation is meant to represent voltages, or differences in electrical potential relative to ground. For the simulation, it is reasonable to assume that ground (or the reference) is the origin, or 0 Volts. *Simulink* does not say it explicitly, but if your Y axis ranges from -1 to +1, you can think of the unit as Volts, as if it were "the real thing". If the simulation were not the same as "the real thing", there would not be a point to running the simulation. When you did the *Matlab* part of the lab, we discussed assigning a 16 bit number to every sample ¹.

Now that you know almost everything about generating a signal, let us do some operations on signals.

¹In the process of extracting samples from a signal, this step is called *quantization*, or "assigning a number value to the amplitude".

2.2 The Four Operations (+ - * /)

If you think of your sine wave as a sequence of individual points equally spaced along time (that would be your X axis) and with values representing Volts (that would be your Y axis), it should be easy to think of operating mathematically on this sequence. When making reference to sinusoids, Electrical Engineers refer to *frequency* and *amplitude*. Frequency is specified in Hertz (Hz), and amplitude in Volts (V), either “Volts-peak” (V_p) or “Volts-peak-to-peak” (V_{pp}). The latter refers to the difference between two crests of the signal, whereas the former refers to the difference between the ground – which is your reference – and the (highest) positive crest.

2.2.1 Adding or Subtracting a Constant to Sinusoid

Now that you have already built a model with two blocks, you can easily bring in some more blocks to make your model more complex. Drag in two more blocks: *Constant* and *Sum*. Connect them as shown in Figure 5. You may have to delete the existing connection before placing and connecting all blocks (there are different ways of doing it).

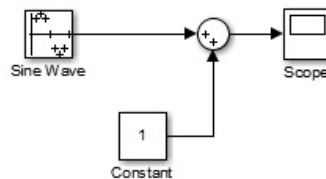
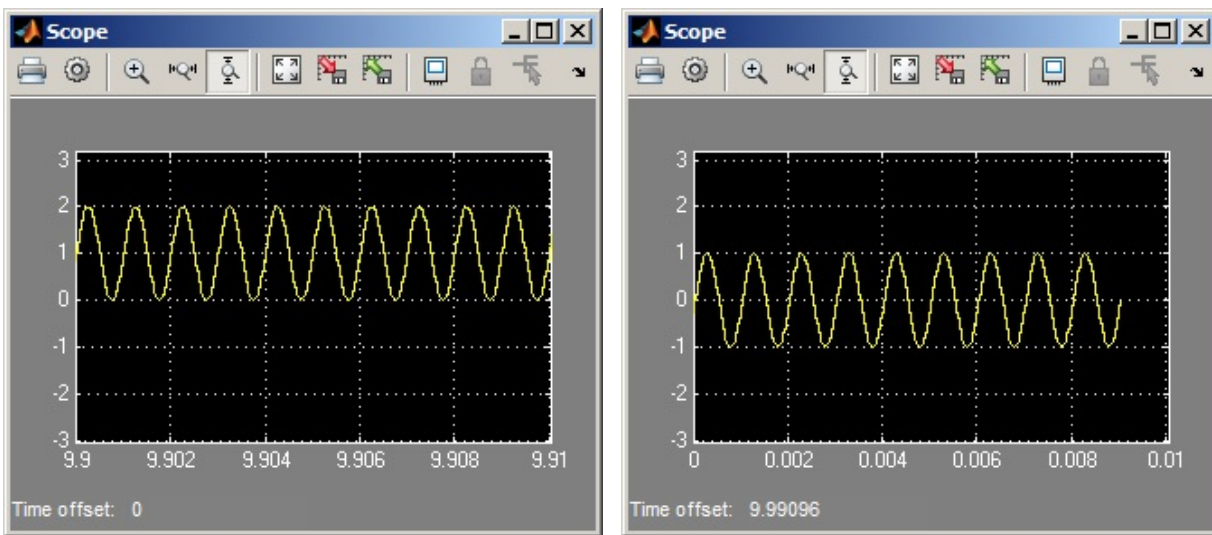


Figure 5. DC Shift

Now run the model and view the output on the scope. Figure 6 shows a side-by-side comparison of the output before, and after, adding a constant to your sine. In Electrical Engineering, this is called a *DC shift*. That is to say, you have a signal of a certain level (voltage) varying in time and you add a constant voltage (like a battery) to it. This happens, for instance, when you have a microphone that needs *biasing* to work. A DC voltage provides the microphone with the potential it needs to work, and the voice it picks up (which will be converted to voltage that varies in time) is added to this DC voltage prior to going into the amplifier. The amplifier will have a capacitor right at its input to block the DC and let the voice pass on to be amplified.



(a) DC Shift

(b) No DC Shift

Figure 6. Two Sinusoids: Without DC Offset and With DC Offset

Exercise 2.2.1:

- (a) Modify the constant to show a DC-shift of 2 for the sinusoid. Save a screenshot and include it with your submission.

2.2.2 Gain (call it “Multiplication by a Scalar”)

A gain is a multiplication by a factor, say β . If β is less than one and greater than zero, you have *attenuated* the input signal (really a division!). If β is negative, then the signal is also “flipped” across the Y axis, which really means a phase shift by 180 degrees.



Figure 7. 6dB Gain

After you add the gain block to your model, try different factors and observe the result. Why is Figure 7 labeled a “6dB Gain”?

Exercise 2.2.2:

- (a) Apply a 10dB gain to your sine wave and show it. Save a screenshot and include it with your submission.

2.2.3 Operating on Two Sinusoids, Making Them Sound via Speaker

Build the model to have two sine wave inputs: one of them at 1KHz and the other at 2KHz. Now apply a gain of 2 to the 1KHz sinusoidal tone, add them and plot. Figure 8 is what your model should look like. Run your model and observe the output.

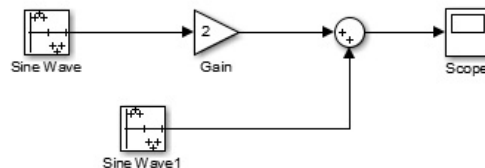


Figure 8. Two Sinusoids, one with a gain

Now you will listen to the sounds that your simulation generates, and will change parameters while the simulation is running. In order for you to do this, first you will need to send your signal to the standard audio device of your workstation. Go to the DSP System Toolbox -- Sinks, and drag into your model the To Audio Device block. Link it to the same signal path as your Scope. You can give this new model a run as it is, but it can be improved with a variable gain, as you will see below. In case you make it run now, be sure to put the headphones/earphones at a distance from your ear, until you find the right level to use.

What you need to do now, is to substitute the Gain block associated with the gain of 2 by the Slider Gain block. Also, add a Slider Gain to the other path as well. This block is found in the library Simulink -- Math Operations. You need to define the range of gains to be applied to the signal on each path. Suggestions are from 0 to 2 on the fundamental signal (your 1KHz sine) and from 0 to 1 on your 2nd harmonic (your 2KHz sine). Your final model should look like Figure 9 below.

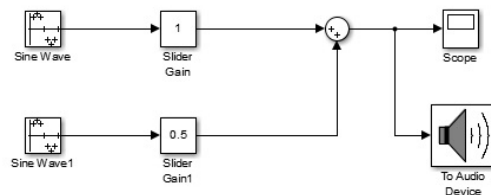


Figure 9. Two Sinusoids, Two Variable Gains, Sounding via Audio Device

With this model, let us simulate harmonic distortion. Musicians often refer to the addition of harmonics onto the original signal as “coloration”. Harmonics can be added in a variety of ways. Some are *wanted*, like sound deliberately put through overdriven tube amplifiers, and some are unwanted, like the 2nd harmonic “hum” added by a transformer in the rectifier/power supply stage of an amplifier. Now run your model and change slowly the gain on the 2nd harmonic and see what audible difference it makes to the signal. Is it “more colourful”? When you run your model, make sure to have the Scope display the waveform, so that you can correlate what you hear with what you see.

Exercise 2.2.3:

- (a) On your Simulink Scope, Show the result of adding a 1Vp, 1KHz sinusoid and a $\frac{1}{2}$ Vp, 2KHz sinusoid. Use the slider gain blocks to assign the magnitudes. Save a screenshot and include it with your submission.

2.2.4 Multiplying Two Sinusoids

Finally, another interesting operation is the multiplication of two signals. To make things resemble your old high-school days, let us multiply two sine waves. As you know, when you multiply two sine waves of different values, you end up with two sine waves again, but with a different amplitude and different frequencies (recall the $\sin(a) * \sin(b)$ formula). The resulting sine waves have frequencies that are the sum and the difference of the original frequencies. In Electrical Engineering, a multiplication by a sine (or cosine) is referred to as *modulation*. Examples of modulation are Amplitude Modulation (AM), Frequency Modulation (FM), etc.

Now build a model to try this out, as shown in figure 10.

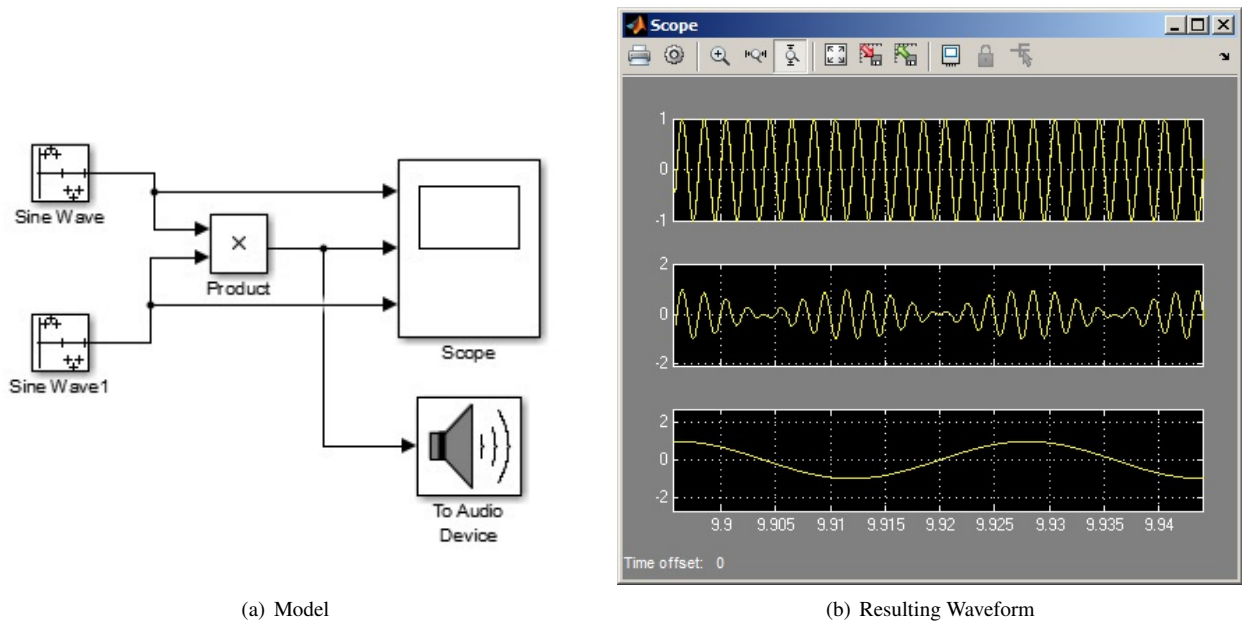


Figure 10. Modulation

The frequencies you will use are 31.25Hz and 500Hz. The amplitude is 1 for both. Now it would be nice to see all signals on the scope while listening to them. The way to make the multiple-input scope shown on Figure 10(b) is to click on the scope parameters (the gear-like symbol at the top left of the scope menu), and select the number of inputs you want in the Number of axes field. The case shown is for 3 inputs.

Exercise 2.2.4:

- (a) Put the model together, make it run. Show it. Save a screenshot and include it with your submission.

You may want to try different frequency values. What you hear is actually an effect used in audio called a *Ring Modulator*.

3. Conclusion

In this experiment you became familiar with some extra capabilities of Matlab and were introduced to Simulink. You have tried a basic set of blocks found in Simulink and made your models interact with Matlab and with the outside world. Perhaps you can now make your own musical instrument with Simulink. Do this as a side project, though, because there is plenty to come in ECE216!

Acknowledgments

Thanks for all the students who have provided input on the previous versions of this experiment. Also for external sources [3], [1], [2], [4], [5], [7], [6], [9], and [8].

References

- [1] S. Haykin. *Communication Systems, 4th Ed.* 2001.
- [2] S. Haykin and M. Moher. *Introduction to Analog and Digital Communications, 2nd. Ed.* 2007.
- [3] P. Kumar. *Digital Signal Processing Laboratory.* 2005.
- [4] B.P. Lathi. *Modern Digital and Analog Communication Systems, 3rd Ed.* 1998.
- [5] Mathworks. <http://www.mathworks.com/hardware-support/audio-dst.html>.
- [6] Yoder M.A. McClellan J.H., Schafer R. W. *Signal Processing First.* 2003.
- [7] Schafer R. W. Oppenheim A. V. *Discrete-Time Signal Processing, 3rd. Ed.* 2009.
- [8] Tektronix. *AFG3000 Series Arbitrary/Function Generators - Quick Start User Manual.*
- [9] Tektronix. *TDS1000 and TDS2000 Series Digital Storage Scope - User Manual.*