Module 3: Listen to Audio Signals with MATLAB

The main goal of this module is to use MATLAB to play audio signals through the loudspeakers. You will first play synthetic waveforms, such as sine waves at different frequencies. Then, you will play and visualize arbitrary waveforms (such as music, speech, and test signals) and manipulate these signals to create interesting audio effects. ***Remember: Whenever you are stuck, have questions, or are interested in learning more details about a specific aspect, please feel free to ask us—we are here to help!!***

---

4 Setting up the Loudspeakers

MATLAB is not only useful to perform calculations and plot functions, it can also be used for *signal processing*. Signal processing is an important subfield of electrical engineering, and is concerned with the analysis, synthesis, and modification of signals. Broadly speaking, signals are functions (such as a the sine or cosine function we generated in Part 1 of this lab) that convey "information about the behavior or attributes of some phenomenon," including sound, images, or physical measurements. In this module, you will learn how to synthesize, analyze, and modify signals using MATLAB through the pair of loudspeakers we provide. More concretely, you will learn how to play a variety of audio signals. We will also analyze their properties and modify the signals to create interesting audio effects. Besides the fact that it is fun to play around with audio signals, you will see later in this project that signal processing is one of the key components in the design of communication systems.

4.1 First Steps

The AVRICKA Peals are USB-powered loudspeakers. Later, we will use these loudspeakers as transmitting "antennas" in this project. Note that USB is only needed to provide power; the audio signal itself is transmitted over the *3.5mm audio jack* cable. First, connect the USB connector to the computer (on the left side of the monitor) and then, connect the audio jack to the computer (also on the left side of the monitor). You can now turn the speakers on by using the switch on the back of one of the right (indicated with "R" just below the two three cables) speaker from "OFF" to "LED." If the speakers is connected to USB and receives power, then you should see a glowing blue light. Then, use the volume knob and turn it no more than 1/3 between "-" and "+". Also check whether the computer audio is turned on; click on the loudspeaker icon lower right of the screen and ensure that the level is about half-way to the maximum. ***Remember: Please be cautious with the audio volume! You do not want to bother the other teams with signals that are too loud or annoying. Also note that we provide free earplugs in case you feel that it is getting too loud in the labs.***

4.2 Transferring Files from X to Y

In order to simplify access to the loudspeakers from MATLAB, for example to play a signal through the loudspeakers, we provide a number of MATLAB functions. These functions hide some of the details that are necessary to play or record signals through external computer interfaces. [*cs*: *Here, explain how to copy the files to your local computer using Cornell Box.*]

cs

4.3 Identifying the Correct Audio Output and Playing a Test Sound

MATLAB is able to access all audio ports (outputs as well as inputs) of your computer. Each audio port is associated with a unique ID number. Unfortunately, these ID numbers can change depending on whether a device was connected to your computer before you started MATLAB.

Before you can play audio through your loudspeakers, you have to identify which ID number the computer audio output port has to which you connected the audio jack. To figure this out, run the function

```
get_audio_info
```

in the Command Window. If MATLAB complains that this function does not exist, then the function is *not* in your Current Folder. Use the buttons above the Current Folder window to navigate to the location where you put the files you just downloaded. Then, try running the command again.

After running the command `get_audio_info` you should see something that looks like this (note that the number and names of the devices can change from computer to computer):

```
Available audio inputs:
InpID=0 | Primary Sound Capture Driver (Windows DirectSound)
InpID=1 | Microphone ((LCS) USB Audio Device) (Windows DirectSound)
InpID=2 | Microphone Array (Realtek Audio) (Windows DirectSound)

Available audio outputs:
OutID=3 | Primary Sound Driver (Windows DirectSound)
OutID=4 | Speakers / Headphones (Realtek Audio) (Windows DirectSound)
```

Find the available audio output with name

```
Speakers / Headphones (Realtek Audio) (Windows DirectSound)
```

which is the output port that is connected to your audio jack. Write down the corresponding `OutID`. If, for example, the correct ID is "4," then type

```
OutID = 4
```

and hit enter. This creates a new variable called `OutID` that contains the value 4 so that you do not have to remember the ID associated with this particular output audio device. From now on you can simply use `OutID` to access the correct ID (unless you restart MATLAB or you clear this variable; in that case you have to re-define the variable). Now, if everything is set up correctly, you can run the following function

```
test_audio(OutID);
```

which should play a synthetic female voice saying "This is just a test. Nothing special." If you cannot hear anything, then check the following things:

- Are the loudspeakers connected to the audio jack and turned on?

- Is the volume on the loudspeakers on about 1/3 between "-" and "+"?
- Is the volume in Windows turned on (check the loudspeaker icon on the bottom right of the screen)?

If MATLAB throws an error, then it probably cannot find the functions; this means that your Current Folder does not contain the functions you have just downloaded. Make sure that your Current Folder contains these functions. ***In case you still have difficulties with the test sound, ask us and we will help!***

5 Playing Audio Signals

Acoustic waves are continuous in amplitude and time (they essentially represent fluctuations in air pressure over time). Unfortunately, computers cannot deal with continuous signals and we have to find a way to represent continuous signals. In fact, storing continuous signals would require an infinite amount of memory, which is clearly impossible. The key technique to represent audio signals in a computer is known as *sampling*. Put simply, sampling is the method to discretize continuous signals in time. We will now explain some of the basic principles of sampling. We will then demonstrate how to play some simple MATLAB-generated waveforms over the loudspeakers.

5.1 First Steps with Sampling

The basic idea of sampling is to take a continuous function (for example a real-world signal, such as speech) and measuring the amplitude of this signal at regular (equidistant) time instants. The rate at which regular "samples" are taken from this function is called the *sampling rate*, often abbreviated as f_s. The sampling rate is measured in Herz (Hz), which is nothing but 1/seconds. The sampling rate f_s tells you how many times per second you are measuring the continuous signal.

Let us consider a specific example of sampling a simple sine wave. Imagine that you want to measure a sine function that oscillates at exactly 50 Hz (this means that you would observe 50 full periods of the sine function every second). Furthermore, we want the sine wave to have a maximum amplitude of one. Mathematically, the continuous function that represents this sine wave is

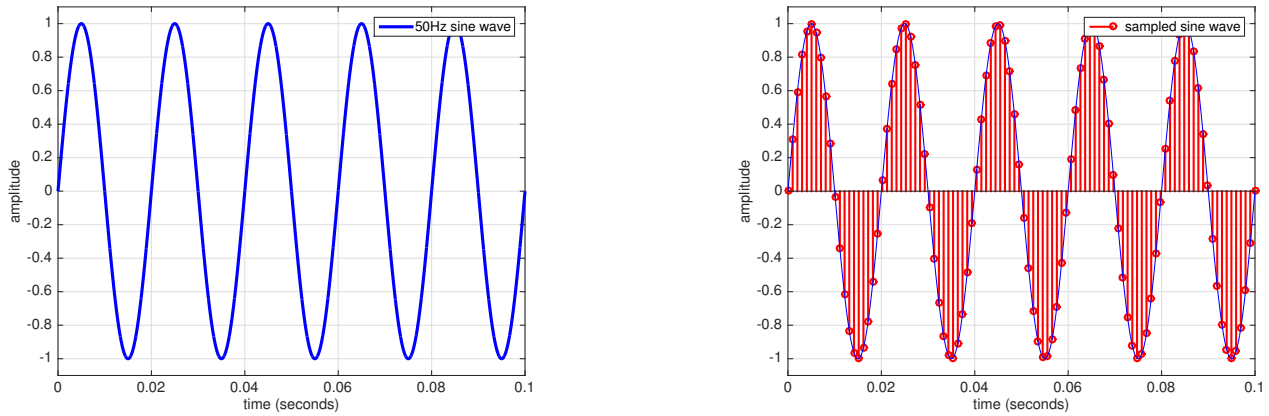
$$f(t) = \sin(2\pi f t), \tag{2}$$

where the variable t represents time (measured in seconds) and f the frequency (measured in Hz) of the sine wave. For our example, assume that the frequency of the sine wave is set to $f = 50$ Hz. The left side of Figure 4 shows the continuous sine wave measured for exactly 0.1 seconds. As expected for a sine wave that oscillates at 50 Hz, you can observe exactly five periods in 0.1 seconds (as you would observe exactly 50 periods for one second). You can also see the maximum amplitude of one.

Let us now try to represent this continuous function in a computer. Let us assume that we are taking 1000 samples of the function every second—this corresponds to using a sampling rate of $f_s = 1000$ Hz (meaning 1000 samples per second). With this sampling rate, you would take exactly 100 samples in an interval of 0.1 seconds. The right side of Figure 4 shows exactly this situation. The red vertical lines correspond to samples and we are taking exactly 100 samples during the 0.1 seconds. As you can see, the red samples approximately represent the continuous function (which is superimposed as a thin blue line). It is now key to realize that we can store these samples in a *vector* in MATLAB!

To create a sampled version of the continuous sine function in (2) with a sampling rate of $f_s = 1000$ Hz, our goal is to read out exactly 1000 values per second. Mathematically, this is equivalent to only looking at the time instants $t = n/f_s$ where n are integer values (e.g., $n \in \{\ldots, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, \ldots\}$). Hence, sampling of the function in (2) corresponds to:

$$f(n/f_s) = \sin(2\pi f n/f_s), \tag{3}$$



(a) 50 Hz sine waveform for 0.1 s.

(b) Sampled version of sine waveform.

Figure 4: Example of sampling a 50 Hz sine waveform at a sampling rate of $f_s = 1000$ Hz.

where we replaced t by n/f_s. Observe that we have discretized (or sampled) the continuous function $f(t)$ and we are taking measurements every $1/f_s$th second, which is also know as the *sampling period* (measured in seconds) and defined as $T_s = 1/f_s$. This means that every T_s seconds, we are taking a new sample. The only missing step is to put these sampled measurements of the sine wave into a MATLAB vector.

In MATLAB, we can easily create a sampled version of the 50 Hz sine wave with amplitude of one. First, we have to generate a vector of that contains the sampling instants. Fortunately, you have learned the `linspace` command, which is perfect for this purpose. By using the command

```
t = linspace(0,0.1,1000*0.1);
```

we are generating a vector `t` that contains the sampling instants from time 0 to time 0.1 and measures exactly $f_s * 0.1 = 100$ time instants. (Note that if $f_s * 0.1$ is not integer, MATLAB will just round this argument of the `linspace` function to the next integer.) We can now evaluate the sine function at exactly these sampling instants. Simply type

```
y = sin(2*pi*50*t);
```

which generates a row vector that contains the samples of our 50 Hz sine function! In words, we measured the continuous sine function at regular intervals of $T_s = 1/f_s$ seconds. If you now type

```
stem(t,y)
```

then you can see the sampled sine function.

Activity 1: Generate sine waveforms with other frequencies

Re-do the above steps to generate, sample, and plot sine waves with the following frequencies: 100 Hz, 200 Hz, 300 Hz, 495 Hz, and 1050 Hz. Leave the sampling rate at $f_s = 1000$ Hz and plot the sampled signal from 0 to 0.1 seconds. Here, it may be useful to write a short MATLAB script in which you can simply change the frequency of the sine wave.

What happens if the frequency of your sine wave approaches or exceeds half of the sampling rate? Can you explain what is going on? We would be happy to hear your explanations!

From the above activity, it becomes clear that one must use a sampling rate f_s that is larger than twice the highest frequency contained in the signal that you are trying to represent. This is a central result in signal processing called the Shannon-Nyquist sampling theorem. What is even more surprisingly is that if you have a signal that only contains frequencies below $f_s/2$, then sampling it at a sampling rate of f_s does not lose any information—this means that you can perfectly represent such signals in a computer.[1]

5.2 Play a Sinusoid

Before, you have seen how to sample waveforms. We now use this knowledge to play such waveforms from the loudspeakers. Let us first look at the simplified block diagram in Figure 5, which illustrates the main components involved in playing audio signals from a computer. MATLAB stores a sampled waveform that is transmitted to the sound-card (in the ACCEL lab, the sound-card is built into the computer). The key component that converts the sampled signals into an analog waveform is called digital-to-analog converter (DAC, for short). Put simply, the DAC takes the digital samples and generates voltages (electronic signals) that correspond to the amplitudes at the predefined sampling rate f_s; this playback sampling rate should be the same as the one used when sampling the original signal. An amplifier then transmits the analog signal to a loudspeaker, which converts an electronic signal into acoustic sound waves. We can then hear these waves using our ears that are able to detect changes in air pressure over time.

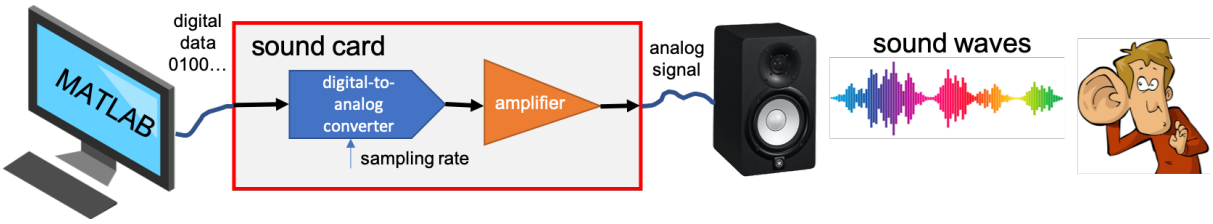


Figure 5: Simplified block diagram of a computer, sound-card, and loudspeaker. The sound-card converts the digital sampled signal into an analog signal that can be played back by a loudspeaker.

Important: Figure 5 represents a communication system where MATLAB, the sound-card, and the loudspeaker are the* transmitter*, the air is the* wireless channel*, and our ears and brain are the* receiver*. By playing a speech signal, we are actually transmitting information over the air. During this project, we will replace our ears and brain with a microphone and computer and transmit digital information over the air. What is even more interesting, your cellphone is doing exactly the same thing, except that the loudspeaker and microphone are replaced by antennas that transmit and receive electromagnetic waves rather than acoustic sound waves—otherwise, the principle of the communication system is the same.

Playing back sampled signals from MATLAB is very easy. First, you need to know that the sound-card built into the ACCEL lab computer use a default sampling rate of $f_s = 44,100$ Hz. Note that this is a very common sampling rate that is also used in Compact Discs (if anyone remembers them?). This sampling rate is high enough to represent frequencies up to 22,050 Hz which is inaudible to most of us (unless you have extremely good ears). For the rest of the project, we define the sampling rate as

```
FS = 44100
```

[1]This statement assumes that you use infinite precision when representing the amplitudes. In practice, one also has to quantize the amplitudes to store and process signals on a computer. This aspect is known as quantization and can be ignored in our project.

It is a good idea to define this variable so that you do not have to type 44100 every time we try to play, analyze, or record audio signals. Let us now generate a test signal, which we then play through our loudspeakers. Our goal is to generate a sine wave that lasts two seconds at a frequency of 440 Hz (which is the same frequency of the A4 key on a piano). As before, we first generate a vector that contains the sampling time instants from $t = 0$ seconds to $t = 2$ seconds:

```
t = linspace(0,2,FS*2);
```

Note that the third argument `FS*2` ensures that we are taking a total of $f_s * 2$ samples (since we are taking f_s samples per second, we have to take $f_s * 2$ samples in two seconds). The vector `t` now contains all the necessary sampling instants. Let us now generate a vector that contains the sampled values of a sine wave that oscillates at 440 Hz. As before, we execute the following MATLAB command:

```
y = sin(2*pi*440*t);
```

Now, we are ready to play this vector of samples to through the sound-card and loudspeakers. Simply use the command

```
play_audio(y,FS,OutID);
```

which plays back your sine wave. Note that the function `play_audio` requires three arguments. The first argument is the vector that contains the samples. The second argument contains the sampling rate; clearly, you need to tell the sound-card what sampling rate was used when generating the sampled vector `FS`. The last argument is the output ID of your sound-card. In case you forgot what the value of `OutID` is, re-run the function `get_audio_info`. If you can hear a tone: ***congratulations***, you have successfully transmitted your first signal over the air! If you cannot hear anything, go again through the steps detailed in Section 4.3 or talk to one of us—we are here to help!

Activity 2: Play back different signals

Generate test signals at different frequencies. For example, double the frequency from 440 Hz to 880 Hz. You should hear a signal at the same frequency as the A5 key on a piano. You can also try to make a shorter signal (for example only half a second). You can also try to play back other periodic functions but you have to make sure that the largest amplitude does not exceed the interval $[-1, +1]$. For example, try the following signal:

```
y = sign(sin(2*pi*440*t));
```

What does the `sign` command do? You can plot the sampled waveform to see what this command does. Also, how does this command change the sound?

5.3 Load, Visualize, and Play Existing Signals

You now understand the basic principles of sampling and you are able to play back a synthetically generated waveform (a sine!). You will now learn how (i) to load general waveforms (representing speech, music, and other signals) that have been sampled before, (ii) to display these waveforms, and (iii) to play these waveforms back! The zip-file you downloaded contains a number of example waveforms in the `examples` folder. Type the following command to load the `examples/noise-white.wav` waveform into MATLAB:

```
[y,FS] = load_audio('examples/noise-white.wav');
```

This command loads the waveform "noise-white.wav" into the vector y and also reads the sampling rate FS that has been used when this waveform was generated (or recorded). Check what the sampling rate of that waveform was—it should be 44,100 Hz. It is often interesting to look at the sample waveform. Type the command

```
plot_signal(y,FS);
```

This command generates a plot of the sampled waveform contained in the vector y with sampling rate FS. You can see that this signal is 5 seconds long. If you zoom in (with the magnifying glass), you can see that the waveform is completely chaotic—completely different to the periodic sine waves we generated before. Do you have an idea how this waveform will sound when played back? Let's try it by running the following command:

```
play_audio(y,FS,OutID);
```

What you hear is so-called "white noise," which is a random signal that contains all frequencies below $f_s/2$. Have you ever tuned your radio to a frequency that contained no transmitting radio station? This is exactly how this sounds like. In fact, if you tune your radio to an unused frequency, then you can hear the same noise. This noise comes mostly from the radio receiver that tries to pick up a radio signal where there is none. Noise is the main reason why there are transmission errors over wired and wireless communication channels—you will learn more about this later.

Activity 3: Play back wav-files

The `examples` folder contains a number of example waveforms that have been sampled and stored in a format that MATLAB can read. Load a waveform and play the waveform back. By visualizing the waveform, try to identify why a certain signal sounds the way it looks. Try the same thing with the other waveforms in this folder. In most cases, visualizing the signal reveals why a signal sounds the way it looks (or looks the way it sounds)—for example, look at the signal `examples/speech-male.wav` and listen to it.

Here, it is useful to create a MATLAB script that loads a certain wav-file, visualizes it, and plays it back. Maybe you remember that you can define strings in MATLAB. For example, add

```
filename = 'examples/noise-pink.wav'
```

to your MATLAB script to define the file you want to load. You can then pass this string to the `load_audio` command as follows:

```
[y,FS] = load_audio(filename);
```

This can make your MATLAB script look even more clean.

5.4 What Happens if we Change the Sampling Rate?

Load the example waveform

```
[y,FS] = load_audio('examples/speech-male.wav');
```

By typing `length(y)` you will see that this signal consists of 46,104 samples. Furthermore, by inspecting the variable FS (which was set while loading the male speech waveform) you will see that this particular wav-file was sampled at only $f_s = 16,000$ Hz. Now, use the command

```
play_audio(y,44100,OutID);
```

which plays back the sampled signal at a new sampling rate of 44,100 Hz. Here's what happened: The original waveform was sampled 16,000 times per second but you are playing it back by generating 44,100 samples per second, which is approximately 2.75× faster. By playing something 2.75× faster, you effectively increase the pitch of the signal by 2.75× and, at the same time, you are reducing the length of the signal by 2.75×. Clearly, it is extremely important to use the same sampling rate when generating or recording a signal, and when playing the signal back. Unless you want to speed up or slow down certain signals. You can also try the following command:

```
play_audio(y,8000,OutID);
```

Here, you hear the same signal but at 2× lower pitch (as we increase the sampling period by a factor of two; remember that $T_s = 1/f_s$), which also takes 2× longer.

Activity 4: Messing Around with Sampling Rates

Try to play back other waveforms in the `examples` folder using different sampling rates. Note that this effect is heavily used in the movie "Alvin and the Chipmunks," where voice actors first recorded a song at slightly slower speed and then, played the record back faster which increased the pitch... Also, "scratching" of vinyl records uses the same principle—there, one modulates the sampling rate over time from slow to fast (and vice versa). Many other audio effects (such as flanger, chorus, and phaser) that are used to change the sound of electric guitars in rock music use the same idea of modulating the sampling rate.

Important: If you are interested in creating even more interesting audio effects, then talk to us. There are many simple things one can do in MATLAB to manipulate audio signals!