CHAPTER 8

Audio Processing

In Chapter 7, you learned about processing signals with GNU Octave. Audio is a type of signal and its processing requires detailed knowledge of signal processing. So, as a continuation of the previous chapter, in this chapter you will learn how to process audio with GNU Octave. The following is the list of topics that you will explore in this chapter:

- Reading an audio file
- Creating your own audio file
- Plotting the sound wave signal

By the end of this chapter, you will be able to work with audio files and process audio signals.

Reading an Audio File

Create a new Jupyter notebook for this chapter. We have recorded an audio file named `sample.wav`. As you can see, it is in WAV (Waveform Audio File Format). You can use other file formats like OGG or MP3 too. Create a string for the filename as follows:

```
file = 'sample.wav'
```


A. Pajankar and S. Chandu, *GNU Octave by Example*,
https://doi.org/10.1007/978-1-4842-6086-9_8

You can retrieve information about the audio file with the function `audioinfo()` as follows:

```
info = audioinfo (file)
```

The output is as follows:

```
info =

  scalar structure containing the fields:

    Filename = C:\Users\Ashwin\OneDrive\GNU Octave Book\First_
    Drafts\Chapter08\programs\sample.wav
    CompressionMethod =
    NumChannels =  2
    SampleRate =  44100
    TotalSamples =  70560
    Duration =  1.6000
    BitsPerSample =  16
    BitRate = -1
    Title =
    Artist =
    Comment =
```

You can read the data stored in the audio file into GNU Octave numerical arrays with the function `audioread()` as follows:

```
[M, fs] = audioread(file);
```

It returns two values. Depending on the number of channels, `M` is a one- or two-column array. We recorded a stereo audio clip so it has two channels. You can also see the number of channels in the previous output. `fs` is the sampling frequency (mentioned as sample rate in the previous output). It is 44100 Hz in this case, which is one of the standard values in the domain of audio. It is usually used by digital audio CDs. The other standard frequency is 48 kHz (48000 Hz).

The function audioread() has many parameters. You can use it as follows to read the file in the native datatype of the stored audio:

```
[M, Fs] = audioread(file, datatypes = 'native');
```

You can also specify the datatype in which you want to read the audio file:

```
[M, Fs] = audioread(file, datatypes = 'uint8');
```

Creating Your Own Audio File

You can create your own signals and write them as an audio file. You have to use function audiowrite() for this. The following is an example:

```
filename='sine.wav';
fs=44100;
t=0:1/fs:10;
w=2*pi*440*t;
signal=sin(w);
audiowrite(filename, signal, fs);
```

The example creates a sine wave and you can even play it using an audio player. The duration of the wave is 10 seconds. You can play it with a built-in audio player in GNU Octave using the functions audioplayer() and play(), as follows:

```
[M, fs]=audioread(filename);
player=audioplayer(M, fs, 8);
play(player)
```

Plotting the Sound Wave Signal

Let's see how to use the function `plot()` to plot the audio wave signal. Create two small audio signals of 0.01 seconds for this, as follows:

```
signal1='signal1.ogg';
signal2='signal2.ogg';
fs=44100;
t=0:1/fs:0.01;
w1=2*pi*440*t;
w2=2*pi*660*t;
audiowrite(signal1,sin(w1),fs);
audiowrite(signal2,sin(w2),fs);
```

The signals have different frequencies. You visualize the first signal, `signal1`, as follows:

```
%plot gnuplot
[M1, fs] = audioread(signal1);
plot(M1)
```

The output is shown in Figure 8-1.

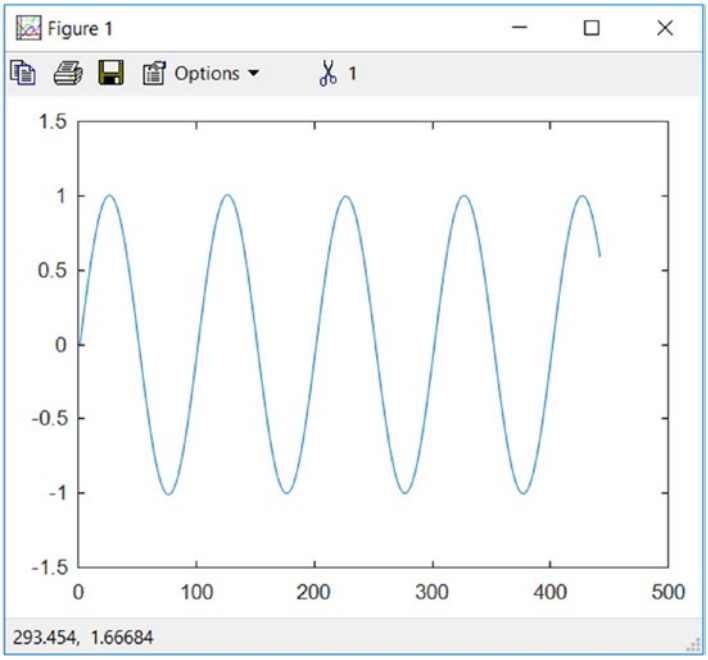

Figure 8-1. *Sine wave*

You can add two signals and visualize as follows:

```
[M2, fs] = audioread(signal2);
plot(M1+M2)
```

The output is shown in Figure 8-2.

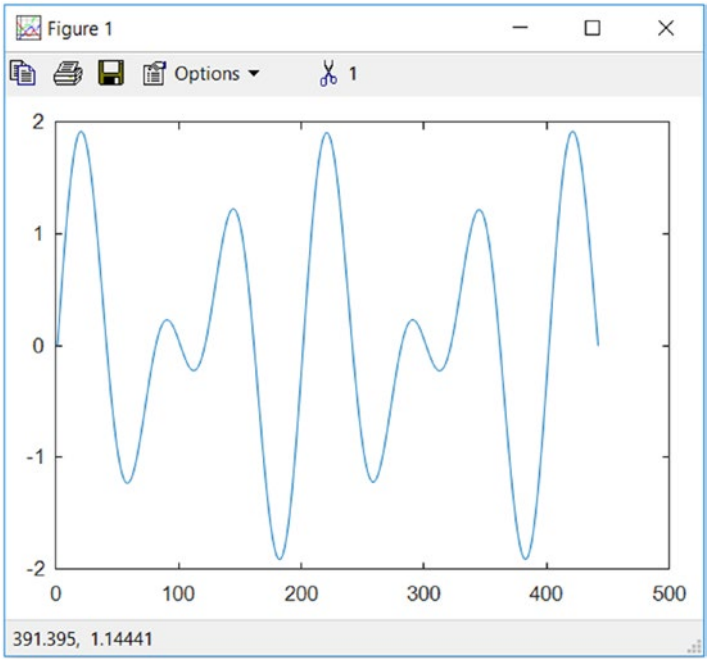

Figure 8-2. *Two added sine waves*

You can multiply two sinusoidal functions as follows:

```
audiowrite('product.wav', M1.*M2, fs);
[M3, fs]=audioread('product.wav');
plot(M3);
```

The output is shown in Figure 8-3.

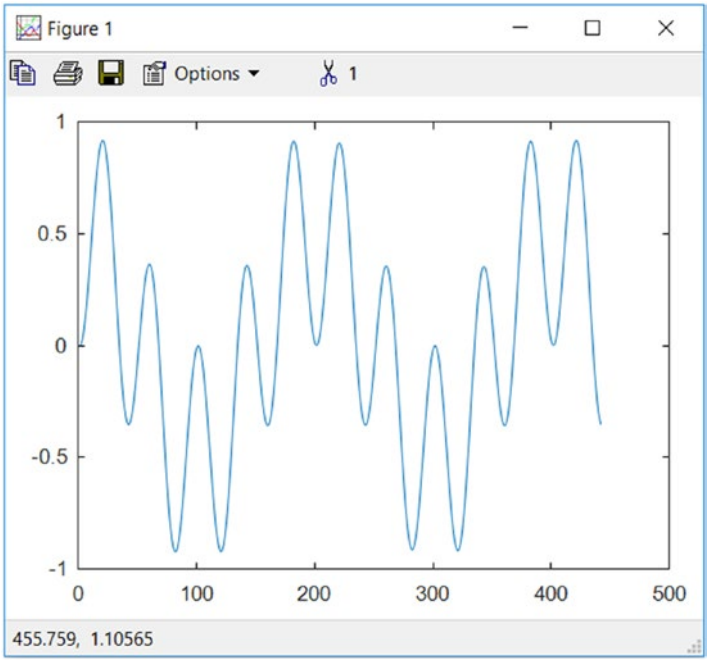

Figure 8-3. *Two sine waves multiplied*

You can divide two signals as follows:

```
audiowrite('div.wav', M1./M2, fs);
[M4, fs]=audioread('div.wav');
plot(M4);
```

The output is shown in Figure 8-4.

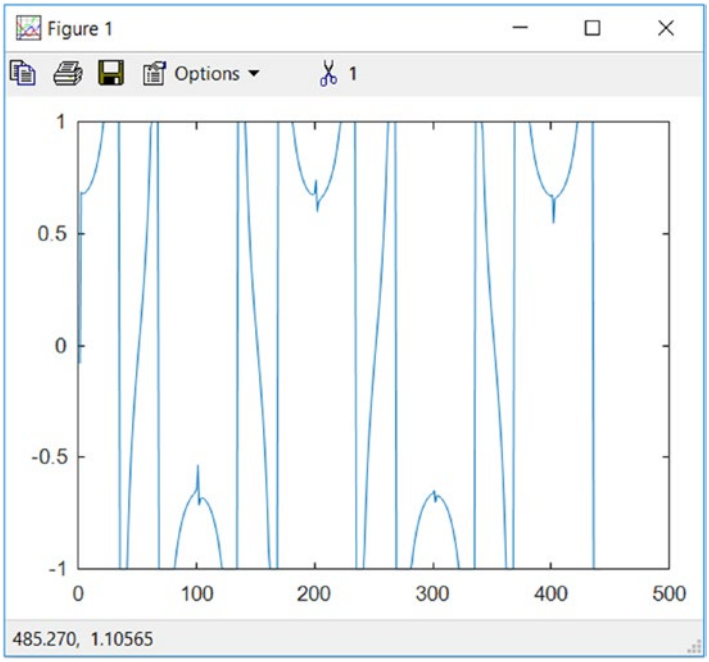

Figure 8-4. *Division of sine waves*

This is how you work with audio signals.

Summary

In this chapter, you learned and demonstrated how to process audio signals. You also saw how to read and write audio signals. You have seen how to perform mathematical operations on audio signals. As discussed, audio processing is a form of signal processing and these techniques are very useful in the domain of audio processing.

The next chapter teaches you more complex applications of signal processing with GNU Octave. You will learn about image and video processing in detail in the next chapter.