CHAPTER 7

Signal Processing

In Chapter 6, you learned about data analysis in GNU Octave in detail. In this chapter, you will learn about signals, different types of signals, Fourier transform, and how to use signals in GNU Octave.

Signals

A signal, by definition, refers to a function used to convey information about a phenomenon. In electronics, you can think of signal as a voltage or current or radiation value. A signal can be of many types. It can be an audio, image, or video signal. Audio signals can be captured through a microphone. Images and videos can be captured through a camera. In the next two chapters, you will see in more detail how to work with audio, images, and videos.

Continuous and Discrete Signals

As defined above, a signal is a function. From mathematics, you know that signals can be continuous and discrete. In case of continuous signals or continuous-time signals, you can acquire the value at any arbitrary point where the signal is defined. Discrete signals are also referred to as a time series. As the name suggests, the values of the function are discrete. One of the examples of a discrete signal is a histogram, which you saw in Chapter 6. In discrete signals, you can only get the value at which the signal is defined.

A. Pajankar and S. Chandu, *GNU Octave by Example*,
https://doi.org/10.1007/978-1-4842-6086-9_7

Let's first create a new Jupyter notebook for the exercises in this chapter. In the first cell, type the following:

```
# Signal Processing
```

Set it as markdown and then run it.

You need to install the signal processing toolbox. Do so by running the following command:

```
pkg install -forge signal
```

Load the package by running the following command:

```
pkg load signal
```

Now, let's see how to create continuous and discrete signals in GNU Octave. First, here's a continuous signal:

```
t = linspace(0, 2*pi);
x = @(t) sin(t);
```

In this code, you create a function or signal which computes values for sine between 0 and 2π. Now plot it to see what the signal looks like:

```
%plot gnuplot
figure(1), plot(t, x(t)), grid on;
```

The output is shown in Figure 7-1.

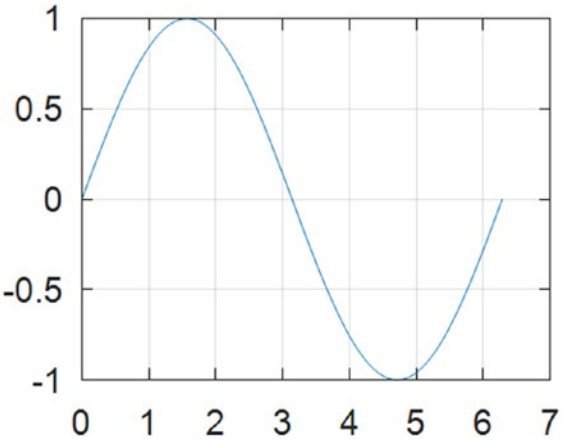

Figure 7-1. *Continuous sine signal*

You see a continuous sine signal plot.

Let's create a discrete time-series function for a `sin` function as follows:

```
t = [0, pi/4, pi/2, 3*pi/4, pi, 5*pi/4, 3*pi/2, 7*pi/4, 2*pi];
x = sin(t);
```

You will now plot the discrete sine signal to see how it looks:

```
%plot gnuplot
figure(2), stem(t, x), grid on;
```

The output is shown in Figure 7-2.

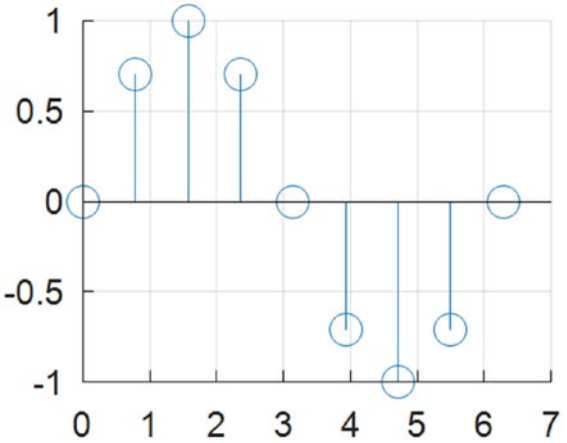

Figure 7-2. *Discrete sine signal*

You can see a sparse sine signal with a few selected points. Note that for value between 0 and 2π, you can get the value of the signal at any point in the continuous signal but for a discrete signal, you can obtain values only at the discrete values where the signal is defined. For example, you cannot get the value of x at t=2 in the discrete sine signal.

Analog and Digital Signals

An analog signal is a continuous signal; an example of an analog signal is an audio signal, which you will see in more detail in Chapter 8. These signals are smooth and you can get values with great precision, whereas a digital signal is a discrete signal that can take only a fixed number of values. A good example is the bits in a computer data stream. They can either be 0 or 1 and images, which you will see in more detail in Chapter 9. While we live in an analog world, we rely on computers for computation purposes, which is a digital world. Because of this, we tend to quantize our signals for faster computation. Quantization is the process of mapping a continuous set of values to a finite number of values.

Even and Odd Signals

If you recall functions from mathematics, every function can be expressed as a summation of even and odd signals. Even signals satisfy the following property:

```
f(-x) = f(x)
```

An example of an even signal is a `cos` function:

```
f(x) = cos(x)
f(-x) = cos(-x)
      = cos(x)
      = f(x)
```

And odd signals satisfy the property

```
f(-x) = -f(x)
```

An example of an odd signal is a `sin` function:

```
f(x) = sin(x)
f(-x) = sin(-x)
      = -sin(x)
      = -f(x)
```

In the Fourier transform section later in this chapter, you will see that a signal is a combination of `sin` and `cos` functions, which are even and odd functions. Therefore, you can use the properties of even and odd functions to form Fourier series properties. You can read more about the Fourier series and its properties by yourself.

Periodic and Non-Periodic Signals

Periodic signals are functions that repeat themselves after a fixed interval. Periodic functions satisfy the property

```
f(t) = f(t + T)
```

where T is the time period after which the signal repeats the same values.

Periodic signals can be both continuous and discrete. In addition to trigonometric functions, you can plot other period functions in GNU Octave.

Let's see how to plot a sawtooth signal:

```
t = 1:25;
sawtooth = sawtooth(t);
%plot gnuplot
figure(3), plot(t, sawtooth);
```

The sawtooth plot is shown in Figure 7-3.

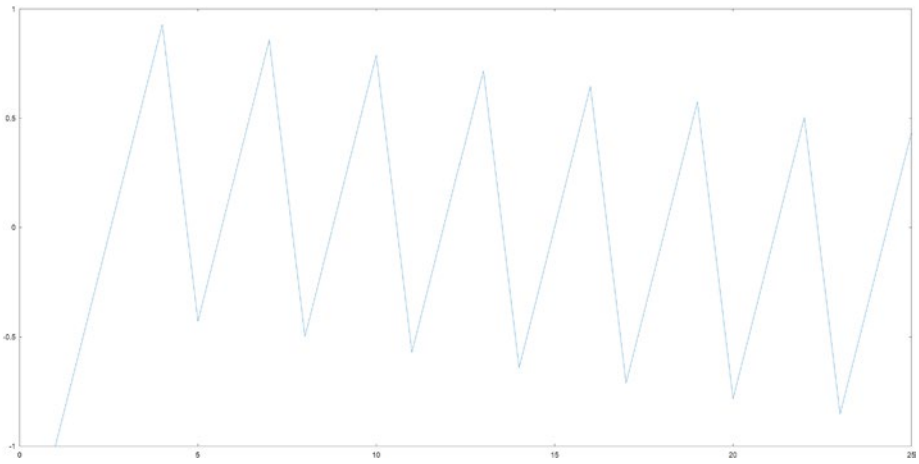

Figure 7-3. *Sawtooth signal plot*

Let's look at how to generate and plot a square signal:

```
t = 0:1/10000:1;
square = square(2*pi*5*t);
%plot gnuplot
figure(4), plot(t, square);
```

The square plot is shown in Figure 7-4.

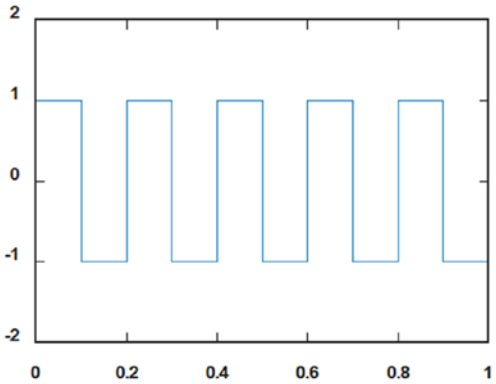

Figure 7-4. *Square signal plot*

You will now look at a few standard non-periodic signals used in signal processing. First, here's a triangular pulse:

```
t = -1:1/10:1;
triangle = tripuls(t, 0.001);
%plot gnuplot
figure(5), plot(t, triangle);
```

The triangular pulse is shown in Figure 7-5.

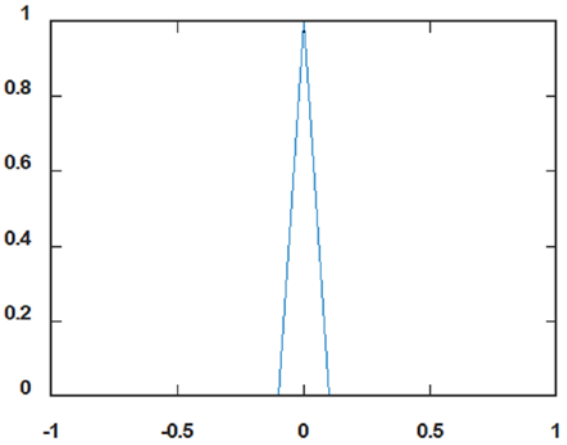

Figure 7-5. *Triangular pulse plot*

You can do the same to create a rectangular pulse or a Gaussian pulse. Note that this signal is not periodic in nature and does not satisfy the condition for periodic signals.

These are the fundamentals of some basic properties of signals and systems. You can learn more about the properties of signals by yourself.

Now let's look into a special kind of signal, the function `sinc()`. The mathematical equation for a `sinc` function is

```
sinc(t) = sin(t)/t
```

You can plot it in GNU Octave by calling the function `sinc()` as follows:

```
t = linspace(-5,5);
sinc = sinc(t);
%plot gnuplot
figure(6), plot(t, sinc);
```

Figure 7-6 shows the plot of the function `sinc()`.

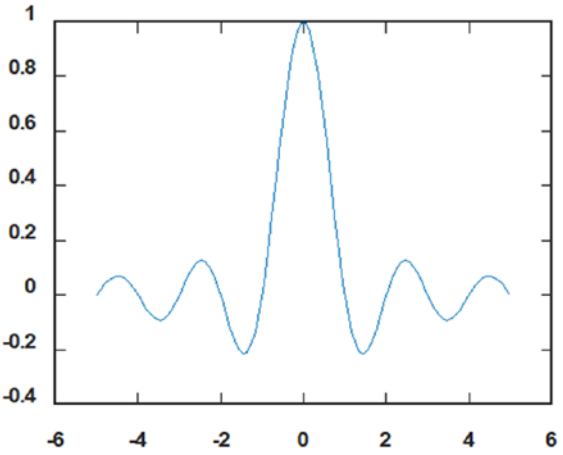

Figure 7-6. *Sinc function*

The Fourier transform of a unit pulse function is a sinc function. If you notice carefully, the sinc function takes value 1 when x is 0 and takes the value 0 for integer multiples of π.

In the next section, you will learn about the Fourier transform and how to compute a Fourier transform using GNU Octave.

Fourier Transform

In the previous section, you looked at functions that are a function of time. If you recollect from physics, time and frequency are the inverse of each other:

```
t = 1/f
```

A Fourier transform comes from the Fourier series. It is a way of expressing the function as a summation of a bunch of sinusoidal functions. The Fourier transform function is defined as follows:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

A Fourier transform has a lot of applications, not just differential equations in mathematics but also in signal processing and Linear Time-Invariant (LTI) systems. As discussed, computers work with discrete values and the input signal is converted to discrete values. The Fourier transform for discrete signal is called a Discrete Fourier Transform (DFT), which is defined as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn}$$

As this forms the basic operation of many signal processing systems, you want the transform operation to be fast. Hence, the Fast Fourier Transform (FFT) is used and is available in the signal processing toolbox. This is a fast way of computing DFT.

Here's how to compute FFT on a 1D signal:

```
t = 0:1/1000:2-(1/1000);
sin_fn = 10*sin(2*pi*10*t);
t2 = length(sin_fn);
t2 = 2^nextpow2(t2);
sin_ft = fft(sin_fn, t2);
%plot gnuplot
figure(7),
subplot(2, 1, 1), plot(t, sin_fn);
subplot(2, 1, 2), plot(abs(sin_ft));
```

In this code, you compute a Fourier transform of a sine function using FFT. The Fourier transform computed has real and complex values, hence you will plot the absolute of the FFT. The result is shown in Figure 7-7.

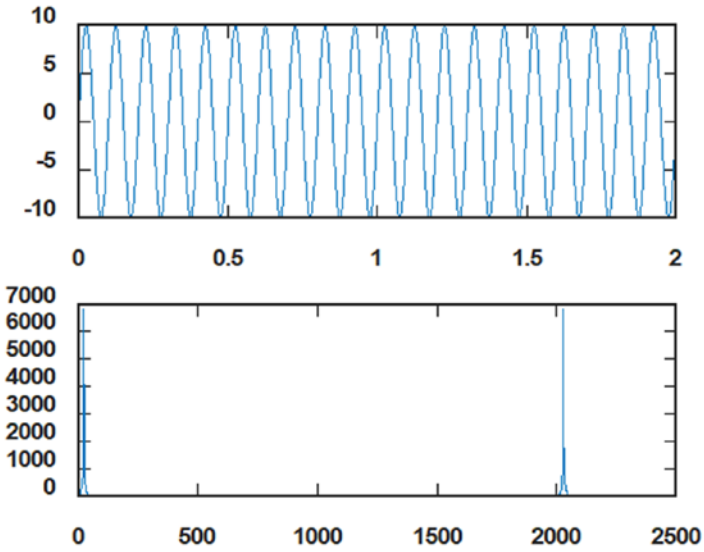

Figure 7-7. *A Fourier transform of a sine signal*

Notice the prominent peak around 2000. This corresponds to the frequency of the sine signal. Unlike the time domain signal, its Fourier transform is very sparse, which makes certain computation in the frequency domain (Fourier transform of the time signal) much faster.

Now, you will add two `sin` functions, one with higher frequency and the other with lower frequency:

```
t = 0:1/1000:2-(1/1000);
sin_fn1 = 10*sin(2*pi*10*t);
sin_fn2 = 10*sin(2*pi*30*t);
sin_fn = sin_fn1+sin_fn2;
t2 = length(sin_fn);
t2 = 2^nextpow2(t2);
sin_ft = fft(sin_fn);
%plot gnuplot
figure(8),
subplot(2, 1, 1), plot(t, sin_fn);
subplot(2, 1, 2), plot(abs(sin_ft));
```

The result is shown in Figure 7-8.

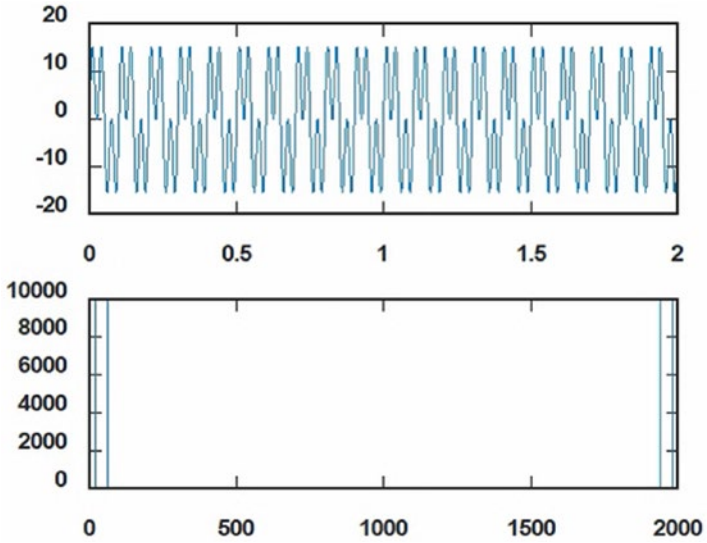

Figure 7-8. *A Fourier transform of the summation of two sine signals*

You see two peaks corresponding to the two different frequencies of the two different sine signals.

Note If you have heard of low-pass filtering, in the frequency domain, the frequency peaks pertaining to the high frequency are removed, which essentially smoothens the signal in the time domain.

You can construct the original signal from a Fourier transformed signal. In other words, to convert the signal from the frequency domain to the time domain, you can use `ifft`. You can explore the inverse Fourier transform function in GNU Octave by yourself.

Note We will discuss FFT in the 2D domain in Chapter 9.

If you have heard about convolution operations, a Fourier transform simplifies the computation of a convolution operation by a multiplication of the Fourier transform of the functions. This is a very interesting operation and with the growing demand for deep learning, these fundamentals are important. You can learn more about this by yourself.

Summary

In this chapter, you learned about signals, various types of signals, and the Fourier transform.

In the next chapter, you will look at audio processing in GNU Octave.