Chapter 1
Generation of Continuous-Time Signals

Learning Objectives

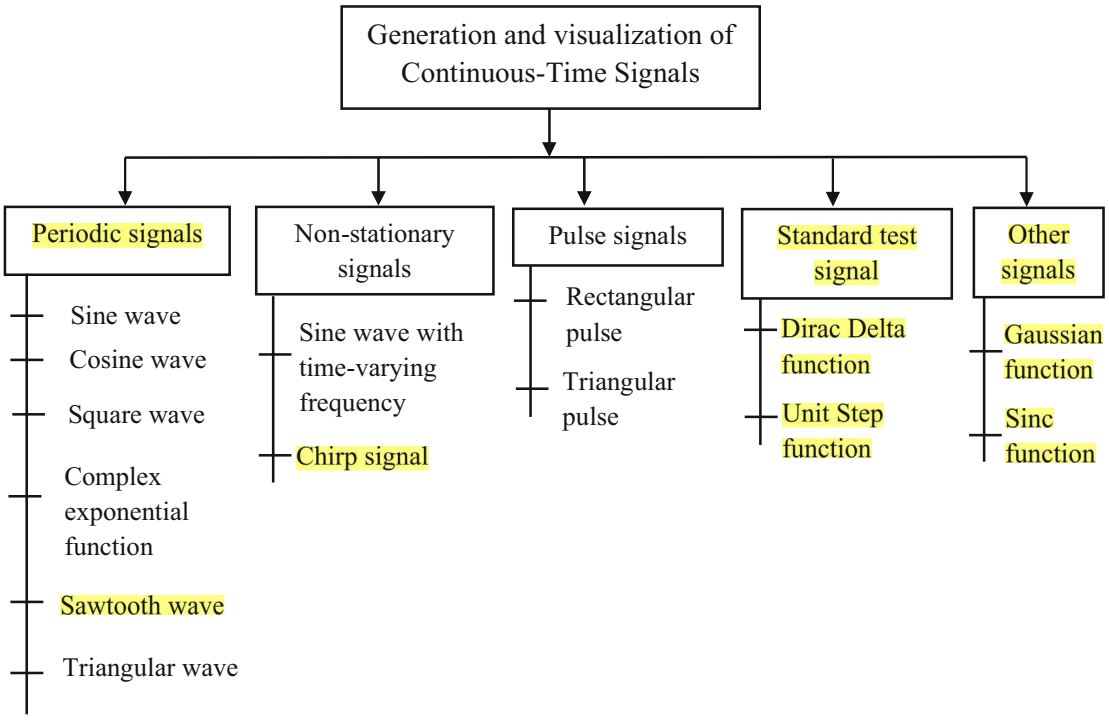
After completing this chapter, the reader should be able to

- Simulate and visualize periodic continuous-time signals.
- Simulate, visualize and interpret non-stationary signals.
- Simulate and visualize standard continuous-time test signals.
- Simulate and visualize continuous-time pulse signals.

Roadmap of the Chapter

This section discusses the flow of contents in this chapter. The objective of this chapter is to generate different types of continuous-time signals, pulse waveforms. The representation of different signals generated in this chapter is given in the form of a flow diagram, which is given below:

S. Esakkirajan et al., *Digital Signal Processing*,
https://doi.org/10.1007/978-981-99-6752-0_1



PreLab Questions

1. Give a few examples of real-world signals, which are continuous in nature.
2. Mention the built-in functions available in '*numpy*' library in python to generate data points of specific length to define the independent variable like time.
3. Explain the significance of a sinusoidal signal in signal processing.
4. What do you understand by the term '*phase*' of a signal?
5. Give a few examples of multidimensional signals.
6. Cite an example where the signal or a process can be modelled as a real exponential function.
7. Mention a few significant features of complex exponential signals.
8. Mention the salient features of the '*sinc*' function in signal processing. Is it an even or odd function?
9. Distinguish between stationary and non-stationary signal. Give examples of each category of signal.
10. List a few significant properties of the Gaussian function (signal).

1.1 Continuous-Time Signal

A signal corresponds to a physical quantity that varies with time, space, etc. Signals are represented mathematically as a function of one or more independent variables. The continuous-time signals are defined for a continuum of values of the independent variable. The continuous-time signal is generally represented as $x(t)$. Speech signal as a function of time is an example of continuous-time signal. The signal can

be either deterministic or random. Deterministic signals can be described by mathematical functions or expressions. In this chapter, the objective is to generate different types of continuous-time periodic signals, like sinusoidal signal, complex exponential signal, square wave, etc.; non-stationary signals, like chirp signal; standard test signals, like Dirac delta; unit step signal, etc.

1.1.1 Continuous-Time Periodic Signal

A periodic signal is one which repeats itself in an identical manner. Examples of continuous-time periodic signals include sinusoidal signal, complex exponential signal, square wave and sawtooth wave. In this section, python codes are developed to generate a sinusoidal signal, three-phase sinusoidal signal, complex exponential signal, etc. Also, sinusoids are Eigen functions of linear system. Continuous-time sinusoids are described by an amplitude, frequency and phase. Continuous-time sinusoids with distinct frequencies are always distinct.

Experiment 1.1 Generation of Sinusoidal Signal
The aim of this experiment is to generate sinusoidal signal. Sinusoidal signals are periodic functions, which are based on the sine or cosine function. The expression for the sinusoidal signal is given by

$$x(t) = A\sin(2\pi ft + \phi) \tag{1.1}$$

In the above equation, 'A' represents the amplitude of the signal, 'f' denotes the frequency of the signal and 'ϕ' indicates the phase of the signal. To generate sinusoidal signal, one should define three parameters: amplitude, frequency and phase. The independent-variable is '*time (t)*'. In amplitude modulation, the amplitude of the carrier is changed in accordance with the message, while the frequency and phase are kept constant. In frequency modulation, the frequency of the carrier is changed in accordance with the signal, while the amplitude and phase are kept constant. In phase modulation, the phase of the carrier is changed in accordance with the signal, while the amplitude and frequency are kept constant.

The steps involved in the generation of sinusoidal signal are summarized below:

Step 1: Defining the independent variable

The built-in function '*np.linspace*()' is used to generate the independent variable, which is the time axis.

Step 2: Defining the parameters of the sine wave

In this step, the three parameters of sine wave, namely, amplitude, frequency and phase are defined.

Step 3: Generation of sinusoidal signal

In this step, the mathematical expression to generate a sine wave is given by

Table 1.1 Built-in functions used in the program

S. No.	Built-in function used	Purpose
1	*np.sin*()	To generate sinusoidal function
2	*np.linspace*()	To generate equally interval data points in an interval
3	*plt.subplot*()	To plot more than one figure in the same plot

```
#Experiment 1: Generation of sinusoidal signal
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the independent variable
t=np.linspace(0,1,1000)
#Step 2: Defining the parameters of sine wave
A=5  #Amplitude of sine wave
f=5  #Frequency of sine wave
ph=0 #Phase of sine wave
#Step 3: Expression of sine wave
x=A*np.sin(2*np.pi*f*t+ph)
#Step 4: Plotting the sine wave
plt.plot(t,x),plt.xlabel('Time'),plt.ylabel('Amplitude')
plt.title('A={}V,F={} Hz,$\phi={}^\circ$'.format(A,f,ph))
```

Fig. 1.1 Python code to generate sinusoidal signal

$$x(t) = A\sin(2\pi ft + \phi)$$

Step 4: Plotting the sinusoidal signal

The built-in function *plt.plot*() is used to plot the generated signal. While plotting the waveform, it is important to mention that the label of x and y axes using *plt.xlabel*() and *plt.ylabel*() command. The command *plt.title*() is used to display the title of the plot.

Built-In Libraries

The built-in libraries used in the program are (1) Numpy and (2) Matplotlib. The '*numpy*' is a general purpose array-processing package. In this program, the numpy library is used to create array (*np.linspace*), and it is used to perform mathematical function (*np.sin*). Matplotlib is a data visualization library used to visualize the generated sinusoidal signal. The built-in functions used in the program is given in Table 1.1.

The python code used to generate sinusoidal waveform is shown in Fig. 1.1, and the corresponding output is shown in Fig. 1.2.

Inference

From Fig. 1.1, the following inferences can be made with respect to python code:

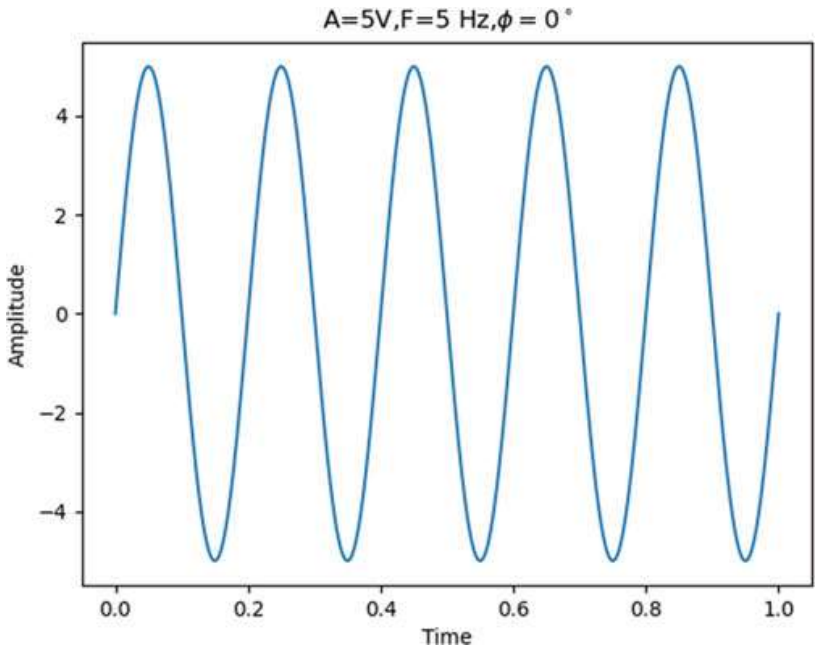


Fig. 1.2 Result of the python code shown in Fig. 1.1

1. The libraries used in the program are (a) Numpy and (b) Matplotlib.
2. The built-in function '*np.linspace*()' is used to generate the independent variable, which is the time axis. In this program, 1000 data points are generated between '0' and '1'.

From Fig. 1.2, it is possible to infer the following:

1. The phase of the signal is '0'; this implies that the waveform starts from the origin.
2. The amplitude of the sine wave is 5 V. The waveform oscillates between −5 and +5.
3. The frequency of the generated waveform is 5 Hz. The number of oscillations per second is 5.

Tasks

1. Write a python code to mark the peak of the sinusoidal signal.
2. Write a python code to compute the number of zero crossing of the sine wave.

Experiment 1.2 Sinusoidal Signal with Different Phase

In this experiment, the objective is to generate sine wave of amplitude = 1 V, frequency = 5 Hz and four different phase angles, namely, 0°, 90°, 180° and 270°. The python code, which does this task, is shown in Fig. 1.3, and the corresponding output is shown in Fig. 1.4. The built-in libraries used in the program are (1) Numpy and (2) Matplotlib.

```
#Generation of sine wave of different phase angles
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,1,100)
#Parameters of sine wave
A=1  #Amplitude
f=5 #Frequency
phi=[0,90,180,270] #Phase
#Generation of sine wave
for i in range(len(phi)):
   x=A*np.sin(2*np.pi*f*t+phi[i]*np.pi/180)
   #Plotting the result
   plt.subplot(2,2,i+1)
   plt.plot(t,x),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
   plt.title('$\Phi ={}^\circ $'.format(phi[i]))
   plt.tight_layout()
```

Fig. 1.3 Python code to generate sine wave with different phase angle

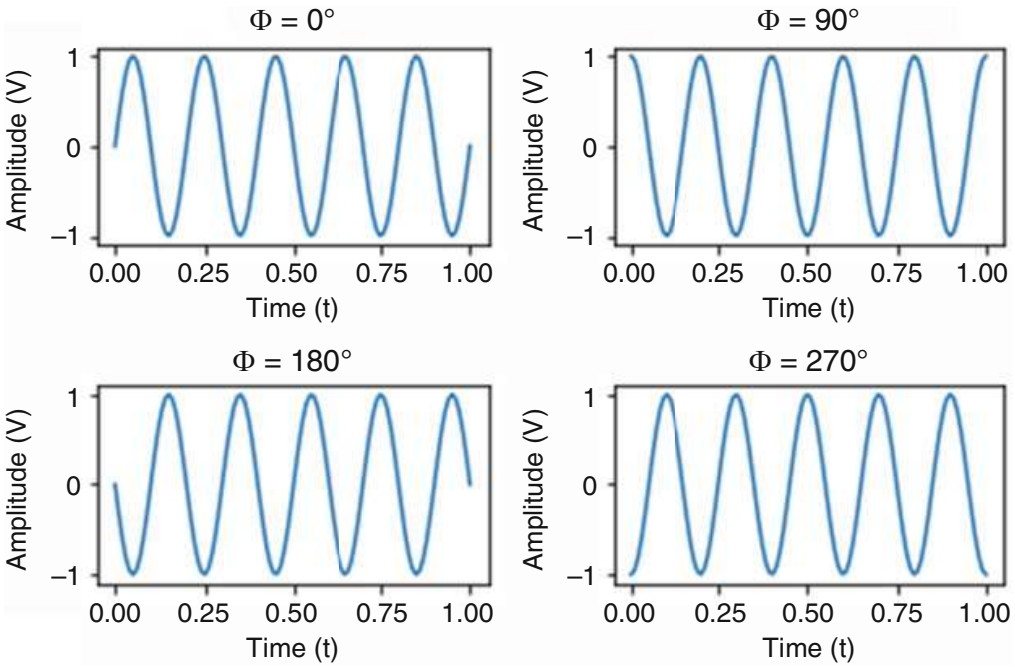


Fig. 1.4 Sine wave with different phase angle

Inference

From the python code shown in Fig. 1.3, the following inferences can be made. The phase angle is varied as 0°, 90°, 180° and 270°. The amplitude of the sine wave is fixed as 1 V and the frequency is fixed as 5 Hz.

```
#Generation of three phase sine wave
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,1,100)
#Parameters of sine wave
A=1  #Amplitude
f=5 #Frequency
#Three different phases of sine wave
phi_1,phi_2,phi_3=0, 120, 240
x1=A*np.sin(2*np.pi*f*t+phi_1*np.pi/180)
x2=A*np.sin(2*np.pi*f*t+phi_2*np.pi/180)
x3=A*np.sin(2*np.pi*f*t+phi_3*np.pi/180)
#Plotting the result
plt.plot(t,x1,'b',t,x2,'r',t,x3,'g')
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
plt.title('Three phase sinusoidal signal')
plt.legend(['Phase-1','Phase-2','Phase-3'],loc=1)
```

Fig. 1.5 Python code to generate three-phase sinusoidal signals

From Fig. 1.4, it is possible to infer that the starting point of the waveform is different for different phase angle. The phase parameter determines the time locations of the maxima and minima of the sinusoid.

Tasks

1. Write a python code to generate a sinusoidal signal whose phase is varying in a random manner. Assume the phase angle 'Φ' to follow uniform distribution in the range -1 to $+1$.
2. Write a python code to generate a sinusoidal signal, whose frequency is varying in a random manner. Assume the frequency 'f' to follow uniform distribution in the range -1 to $+1$.

Experiment 1.3 Generation of Three-Phase Sinusoidal Signal

The expressions for three-phase sinusoidal signals are given by

$$x_1(t) = A\sin(2\pi ft) \tag{1.2}$$

$$x_2(t) = A\sin(2\pi ft - 120^\circ) \tag{1.3}$$

$$x_3(t) = A\sin(2\pi ft - 240^\circ) \tag{1.4}$$

The amplitude and frequency of the three waveforms are equal. The phase shift between the signals is 120°. The python code, which generates the three-phase sinusoidal waveforms, is shown in Fig. 1.5, and the corresponding output is shown in Fig. 1.6.

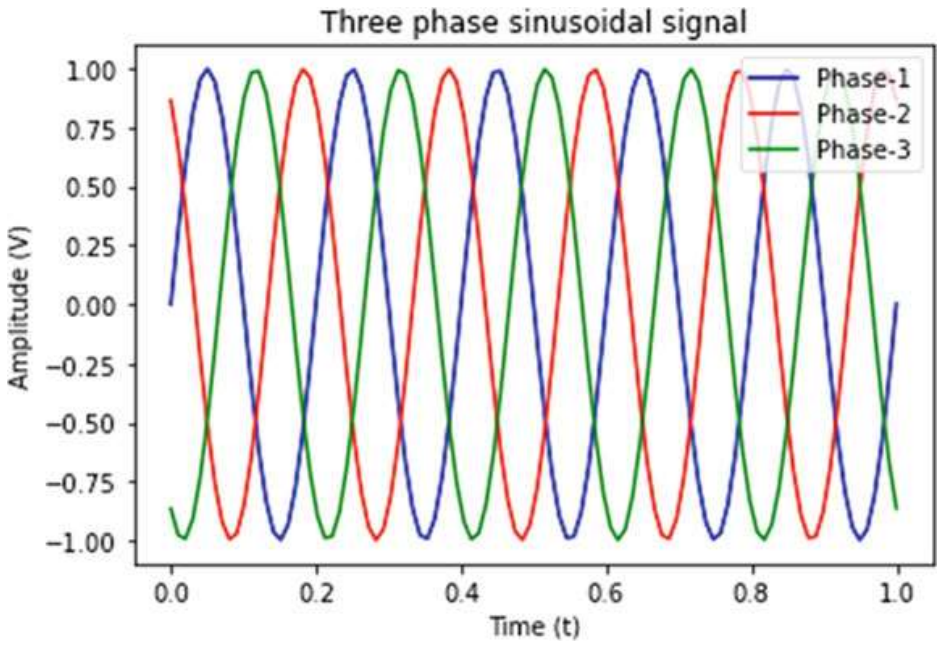


Fig. 1.6 Three-phase sinusoidal signals

Inference
From the python code to generate the three-phase sinusoidal signal, it is possible to observe that the amplitude of each signal is 1 V and frequency is 5 Hz. The phase shift between the signals is 120°.

Task
1. Change the value of amplitude A1, A2 and A3 of three-phase sinusoidal signal in the python code given in Fig. 1.5, and comment on the output waveform.

1.1.2 Exponential Function

Exponential function is of two types: (1) real exponential function and (2) complex exponential function. The real exponential function can be either an increasing function or it could be a decreasing function. The price of petrol is an example of exponentially increasing function. Radioactive decay is an example of exponentially decaying function.

Complex exponential function is of interest in signal processing. Complex exponential function is the basis function of Fourier transform. It is possible to obtain sine wave and cosine wave from complex exponential function. It is an Eigen function for a linear time-invariant system.

Experiment 1.4 Generation of Real Exponential Signal
The general expression for the real exponential signal is given by

Table 1.2 List of built-in functions used in the program

S. No.	Built-in function used	Purpose
1	np.exp()	To generate exponential function
2	np.linspace()	To generate equally interval data points in an interval
3	plt.subplot()	To plot more than one figure in the same plot

```
#Real exponential function
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the time axis
t=np.linspace(-1,1,1000)
#Step 2: Defining the parameter 'alpha'
a,b=2,-2
#Step 3: Generation of function
x1=np.exp(a*t)
x2=np.exp(b*t)
#Step 4: Plotting of the function
plt.subplot(2,1,1),plt.plot(t,x1)
plt.xlabel('Time'),plt.ylabel('Amplitude'),plt.title('Exponentially growing function')
plt.subplot(2,1,2),plt.plot(t,x2)
plt.xlabel('Time'),plt.ylabel('Amplitude'),plt.title('Exponentially decaying function')
plt.tight_layout()
```

Fig. 1.7 Python code to generate real exponential functions

$$x(t) = Ce^{\alpha t} \tag{1.5}$$

where 'C' and 'α' are real. If 'α' is greater than zero, it is exponentially growing function. If 'α' is less than zero, it is exponentially decreasing or decaying function. The built-in functions used in the program are given in Table 1.2.

The python code, which generates exponentially growing and decaying function for $\alpha = 2$ and $\alpha = -2$, is shown in Fig. 1.7, and the corresponding output is shown in Fig. 1.8.

Inference

In exponentially growing function, the value of the function (amplitude of the function) increases with an increase in time. In contrast, in exponentially decaying function, the amplitude of the function decays with respect to time.

Task

1. Write a python code to generate two real exponential functions (one growing and another decaying) with different amplitudes, and add these two functions. Comment on the observed result.

Experiment 1.5 Forward Characteristics of PN Junction Diode

The equation of current through the diode is given by

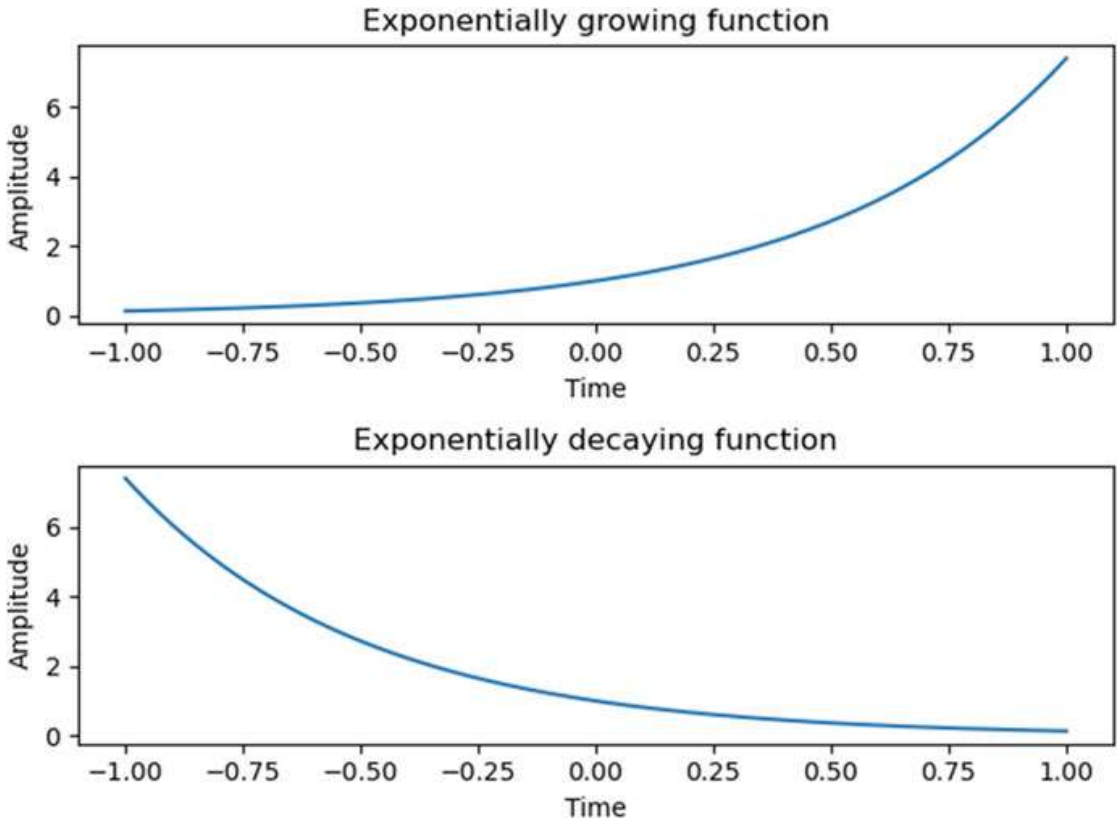


Fig. 1.8 Exponentially growing and decaying functions

$$I_D = I_s\left\{e^{\frac{V_D}{\eta V_T}} - 1\right\} \tag{1.6}$$

In Eq. (1.6), I_s represents the reverse saturation current; V_D is the voltage drop across the diode and I_D is the current through the diode; V_T is the volt-equivalent of temperature, which is 26 mV at room temperature; and η is the ideality factor, which is material dependent. The python code, which simulates the *V–I* characteristics of PN junction diode by assuming $\eta = 1$, $V_T = 26\,\text{mV}$, and $I_s = 1\,\text{mA}$ is shown in Fig. 1.9, and the corresponding output is shown in Fig. 1.10.

Inference

From the forward characteristics shown in Fig. 1.10, it is possible to observe that the diode current increases after crossing the threshold voltage, generally termed '*knee voltage*'. If one considers the current through the diode as a function, then the function is an exponentially growing function.

Experiment 1.6 Radioactive Decay Function

The equation of radioactive decay is given by $N(t) = N_0 e^{-\lambda t}$. The python code, which implements this equation, is shown in Fig. 1.11, and the corresponding output is shown in Fig. 1.12.

```
#Forward characteristics of PN junction diode
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the voltage axis
v=np.arange(0,1,0.001)
#Step 2: Parameters
vt=0.026  #Volt-equivalent of temp.
i_s=1/1000 #Reverse saturation current
n=1 #ideality factor
#Step 3: Equation of current through diode
i=i_s*(np.exp(v/(n*vt))-1)
#Step 4: Plotting the characteristics
plt.plot(v,i),plt.xlabel('Forward voltage')
plt.ylabel('Forward current')
plt.title('Forward characteristics of PN junction diode')
```

Fig. 1.9 Python code to plot the forward characteristics of PN junction diode

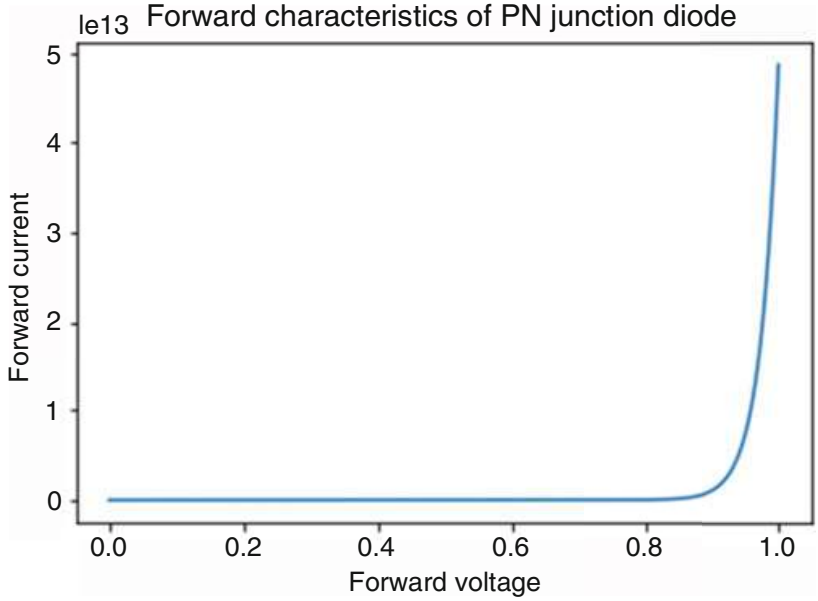


Fig. 1.10 Forward characteristics of PN junction diode

Inference

From Fig. 1.12, it is possible to observe that the radioactive decay activity can be modelled by an exponentially decaying function.

Experiment 1.7 Complex Exponential Function

Generate two complex exponential signals $x_1(t) = e^{j\Omega t}$ and $x_2(t) = e^{-j\Omega t}$. Here the frequency of the signal is fixed as $f = 5$ Hz. After signal generation, extract the magnitude and phase of the two signals, and comment on the observed output.

The built-in functions used in the python program are given in Table 1.3.

```
#Radioactive decay
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the time axis
t=np.linspace(0,80,10)
#Step 2: Parameters
A0=400  #Initial value
T=24
#Step 3: Equation of current through diode
A=A0*np.exp(-t/T)
#Step 4: Plotting the characteristics
plt.plot(t,A),plt.xlabel('Time (Hours)')
plt.ylabel('Counts per second')
plt.title('Radio active decay')
```

Fig. 1.11 Python code for radioactive decay function

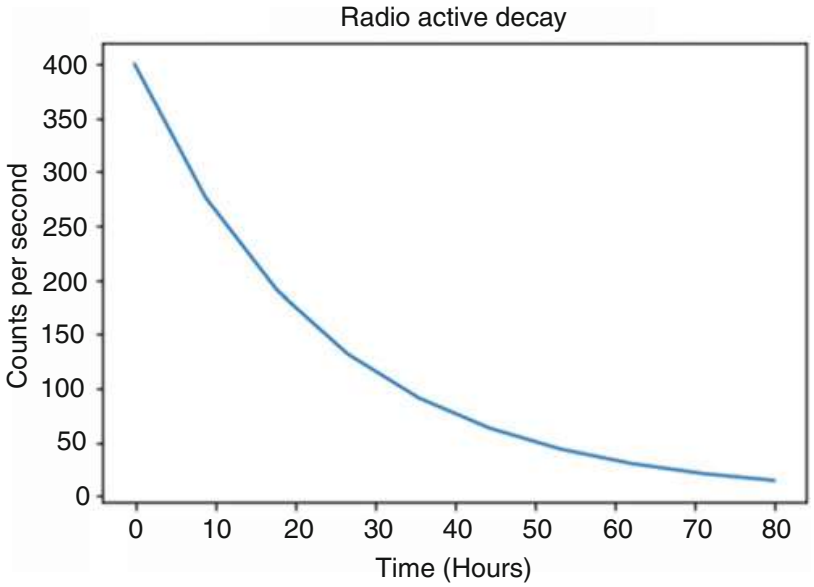


Fig. 1.12 Result of python code shown in Fig. 1.11

Table 1.3 Python built-in functions used in Experiment 1.7

S. No.	Built-in function used	Purpose
1	np.exp()	To generate an exponential function
2	np.abs()	To obtain the magnitude value of a complex number
3	np.angle()	To obtain the phase value of the complex number
4	np.linspace()	To generate equally interval data points in an interval
5	plt.subplot()	To plot more than one figure in the same plot

```
#Complex exponential signals
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Generation of signals x1(t) and x2(t)
t=np.linspace(-1,1,100)
f=5
x1=np.exp(1j*2*np.pi*f*t)
x2=np.exp(-1j*2*np.pi*f*t)
#Step 2: Plotting the signals, its magnitude, and phase
plt.subplot(3,2,1),plt.plot(t,x1)
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
plt.title('$e^{j\Omega t}$'),plt.subplot(3,2,2),plt.plot(t,x2)
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
plt.title('$e^{-j\Omega t}$'),plt.subplot(3,2,3),plt.plot(t,np.abs(x1))
plt.xlabel('Time (t)'),plt.ylabel('Magnitude (V)')
plt.title('|$e^{j\Omega t}$|'),plt.subplot(3,2,4),plt.plot(t,np.abs(x2))
plt.xlabel('Time (t)'),plt.ylabel('Magnitude (V)')
plt.title('|$e^{-j\Omega t}$|'), plt.subplot(3,2,5),plt.plot(t,np.angle(x1)*360/(2*np.pi))
plt.xlabel('Time (t)'),plt.ylabel('$Phase(^\circ)')
plt.title('$\Phi(x_1(t))$'),plt.subplot(3,2,6),plt.plot(t,np.angle(x2)*360/(2*np.pi))
plt.xlabel('Time (t)'),plt.ylabel('$Phase(^\circ)'),plt.title('$\Phi(x_2(t))$')
plt.tight_layout()
```

Fig. 1.13 Generation of complex exponential signals

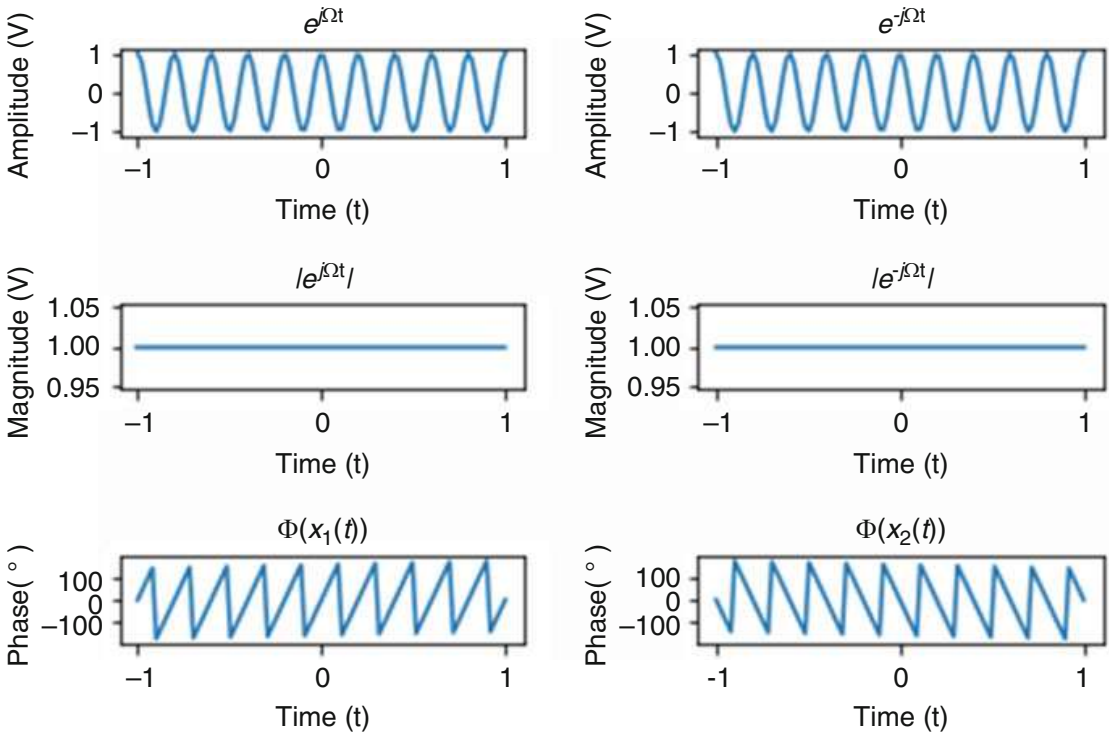


Fig. 1.14 Magnitude and phase of complex exponential signals

```
#Generation of sinusoidal signal
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Generation of rotating phasor
t=np.linspace(0,1,100)
f=5
x1=np.exp(1j*2*np.pi*f*t)
x2=np.exp(-1j*2*np.pi*f*t)
#Step 2: Generation of sine and cosine wave
x_cos=(x1+x2)/2
x_sin=(x1-x2)/(2*1j)
#Step 3: Plotting the result
plt.subplot(2,1,1),plt.plot(t,x_cos),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
plt.title('Cosine wave'),plt.subplot(2,1,2),plt.plot(t,x_sin,'r')
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),plt.title('Sine wave')
plt.tight_layout()
```

Fig. 1.15 Generation of sinusoidal signals from complex exponential functions

The python code, which performs the task mentioned above, is shown in Fig. 1.13, and the corresponding output is shown in Fig. 1.14.

Inferences

From Figs. 1.13 and 1.14, the following inferences can be made:

1. Two complex exponential signals $x_1(t) = e^{j2\pi ft}$ $x_2(t) = e^{-j2\pi ft}$ with the frequency value $f = 5$ Hz are generated. The signals $x_1(t)$ and $x_2(t)$ look alike.
2. The magnitude and phase responses of the two signals are plotted. The magnitude of the signal $x_1(t) = e^{j\Omega t}$ is given by $|x_1(t)| = \sqrt{\cos^2(\Omega t) + \sin^2(\Omega t)} = 1$. Similarly, the magnitude of the signal $x_2(t) = e^{-j\Omega t}$ is given by $|x_2(t)| = \sqrt{\cos^2(\Omega t) + \sin^2(\Omega t)} = 1$. Thus, the magnitudes of the two signals are alike.
3. The phase of the signal $x_1(t) = e^{j\Omega t}$ is expressed as $\phi(x_1(t)) = \tan^{-1}\left(\frac{\sin(\Omega t)}{\cos(\Omega t)}\right)$. Upon simplifying the expression, we get $\phi(x_1(t)) = \tan^{-1}(\tan(\Omega t))$, which results in $\phi(x_1(t)) = \Omega t$. The phase of the signal $x_2(t) = e^{-j\Omega t}$ is expressed as $\phi(x_1(t)) = \tan^{-1}\left(-\frac{\sin(\Omega t)}{\cos(\Omega t)}\right)$. Upon simplifying the expression, we get $\phi(x_1(t)) = \tan^{-1}(-\tan(\Omega t))$, which results in $\phi(x_1(t)) = -\Omega t$. The phases of the two signals are different. This implies that the signal $e^{j2\pi ft}$ and $e^{-j2\pi ft}$ represents two phasors, rotating in the opposite direction.

Experiment 1.8 Generation of 'Sine' and 'Cosine' Functions from 'Complex Exponential Function'

This experiment aims to prove that sinusoidal signal can be generated through two phasors rotating in the opposite direction. Mathematically it is expressed as

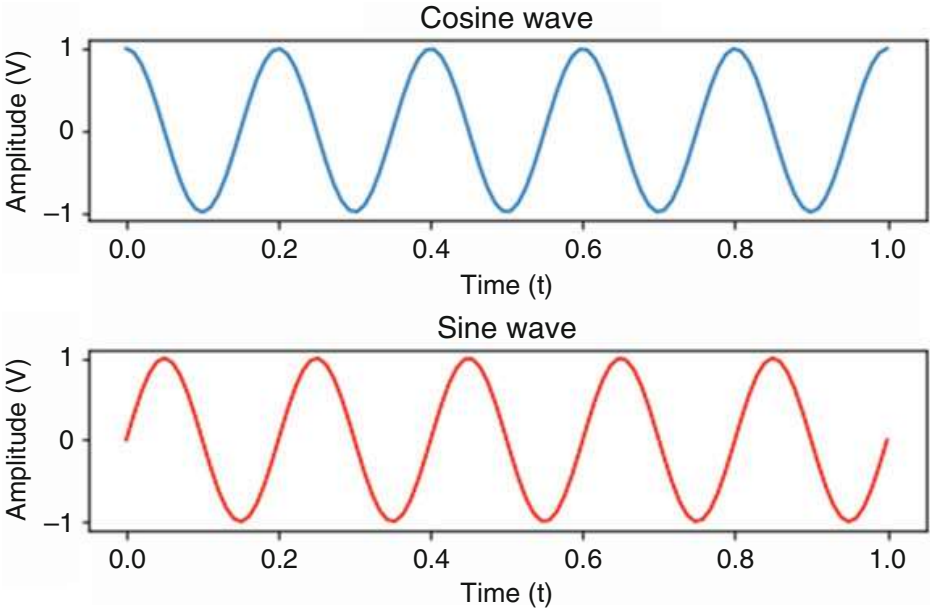


Fig. 1.16 Sine and cosine wave from complex exponential functions

$\cos(\Omega t) = \frac{e^{j\Omega t} + e^{-j\Omega t}}{2}$ and $\sin(\Omega t) = \frac{e^{j\Omega t} - e^{-j\Omega t}}{2j}$. The python code, which generates the cosine and sine wave using a rotating phasor, is shown in Fig. 1.15, and the corresponding output is shown in Fig. 1.16.

Inference

From Fig. 1.16, it is possible to observe that there is a phase difference of 90° between the sine and cosine waveforms.

Task

1. Add the square of the sine and cosine wave obtained in Experiment 1.8, and plot the resultant waveform. Comment on the observed output. [Hint: $\sin^2(\theta) + \cos^2(\theta) = 1$]

Experiment 1.9 Modulating Sinusoidal Signal with an Exponential Signal

This experiment discusses the sinusoidal signal multiplied with a growing and decaying real exponential signal. The python code, which accomplishes this task, is shown in Fig. 1.17, and the corresponding output is shown in Fig. 1.18.

Inference

The following inferences can be made from Fig. 1.18

1. The sinusoidal signal amplitude varies between −1 and +1.
2. Upon multiplying the sinusoidal signal with growing exponential, the amplitude value increases; hence, the plot is shown in the range −5 to +5.
3. Upon multiplying the sinusoidal signal with decaying exponential, the amplitude of the input sinusoidal signal decreases, which is shown between −0.5 and +0.5.

```
#Multiplying sinusoidal signal with an exponential signal
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the time axis
t=np.linspace(0,1,1000)
#Step 2: Defining the parameter
a=2  #Parameter for exponentially growing function
b=-2 #Parameter for exponentially decaying function
#Step 3: Generation of function
x1=np.sin(2*np.pi*5*t)
x2=np.exp(a*t)*np.sin(2*np.pi*5*t)
x3=np.exp(b*t)*np.sin(2*np.pi*5*t)
#Step 4: Plotting of the function
plt.subplot(3,1,1),plt.plot(t,x1)
plt.xlabel('Time'),plt.ylabel('Amplitude'),plt.title('Sinusoidal signal')
plt.subplot(3,1,2),plt.plot(t,x2)
plt.xlabel('Time'),plt.ylabel('Amplitude'),plt.title('Exponentially increasing sinusoidal signal')
plt.subplot(3,1,3),plt.plot(t,x3)
plt.xlabel('Time'),plt.ylabel('Amplitude'),plt.title('Exponentially decaying sinusoidal signal')
plt.tight_layout()
```

Fig. 1.17 Python code to modulate sinusoidal signal by exponential signal

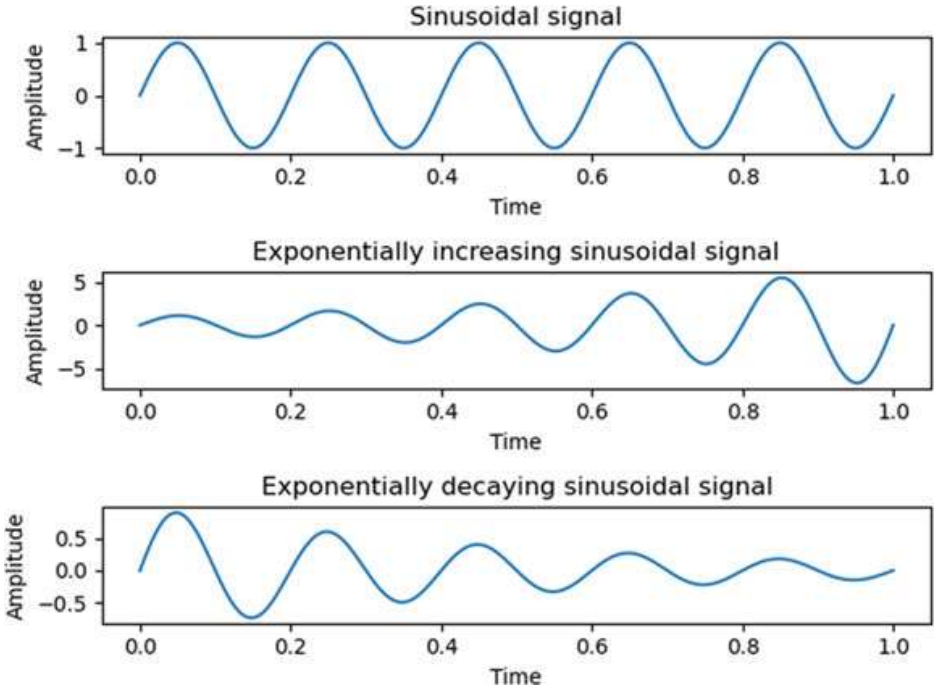


Fig. 1.18 Result of python code shown in Fig. 1.17

1.2 Non-stationary Signal

A stationary signal is one whose statistical characteristics do not change with respect to time. If the signal characteristics change with respect to time, then it is a non-stationary signal. Example of non-stationary signal is a chirp signal whose frequency varies with respect to time. Most of the real-world signals, like the alarm sound from the clock or the sound of an ambulance, are non-stationary. In this section, few stationary and non-stationary signals are generated.

Experiment 1.10 Generation of Stationary and Non-stationary Signal
This experiment deals with the generation of stationary and non-stationary signal. The expression for stationary signal is given by

$$x_1(t) = \sin(2\pi ft) \tag{1.7}$$

The above expression generates a sinusoidal signal of specific frequency. The expression for non-stationary signal is given by

$$x_2(t) = \sin\left(2\pi ft^2\right) \tag{1.8}$$

The frequency of the signal changes with respect to time; hence, it is considered as non-stationary. The python code, which generates the two signals and the corresponding output, is shown in Fig. 1.19, and Fig. 1.20.

```
#Stationary and non-stationary signal
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Generation of signals
t=np.linspace(0,1,1000)
f=5
x1=np.sin(2*np.pi*f*t)
x2=np.sin(2*np.pi*f*t**2)
#Step 2: Plotting of signals
plt.subplot(2,1,1),plt.plot(t,x1),plt.xlabel('Time (t)'),
plt.ylabel('Amplitude'),plt.title('Stationary signal')
plt.subplot(2,1,2),plt.plot(t,x2,'r'),plt.xlabel('Time (t)'),
plt.ylabel('Amplitude'),plt.title('Non-stationary signal')
plt.tight_layout()
```

Fig. 1.19 Python code to generate stationary and non-stationary signal

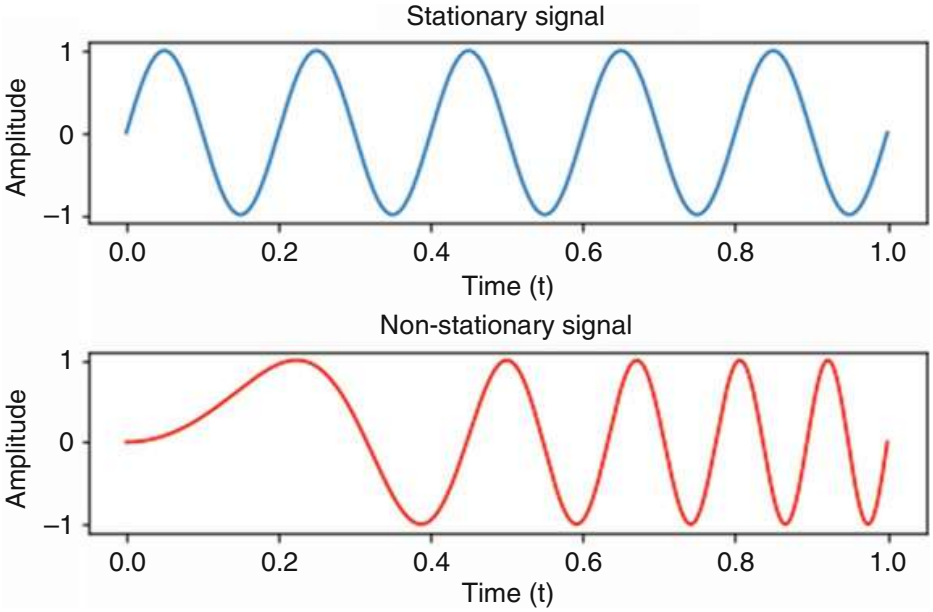


Fig. 1.20 Result of python code shown in Fig. 1.19

Inference

By observing Fig. 1.20, it is possible to conclude that the frequency of the stationary signal does not change with respect to time. On the other hand, the frequency of the non-stationary signal increases with time increases.

Experiment 1.11 Generation of Non-stationary Sinusoidal Signal

The objective of this experiment is to append sinusoidal signals of different frequencies. Signal-1 is generated with 5 Hz frequency appearing first, DC signal next and 10 Hz frequency occurs last. Signal-2 is obtained by interchanging the first and last part of signal-1, which means high frequency occurs first and low frequency occurs next. The python code, which performs this task, is shown in Fig. 1.21, and the corresponding output is shown in Fig. 1.22.

Inference

The signals in Fig. 1.22 are considered as non-stationary, because the signal frequency varies with respect to time. Signal-1 and Signal-2 contain the same frequency components at different instants.

Experiment 1.12 Generation of Chirp Signal

The objective of this experiment is to generate chirp signal. The chirp signal can be considered as a frequency swept sinusoidal signal. Four different methods of frequency sweep are (1) linear, (2) quadratic, (3) logarithmic and (4) hyperbolic. In this experiment, the frequency sweep is from 10 Hz to 1 Hz, as considered. The python code, which generates the chirp signals, are shown in Fig. 1.23, and the corresponding output is shown in Fig. 1.24.

```
import numpy as np
import matplotlib.pyplot as plt
t1=np.linspace(0,1,100)
#Defining signal frequencies
f1,f2,f3=0,5,10
#Generation of signal-1
x1=np.sin(2*np.pi*f2*t1)
#Generation of signal-2
x2=np.sin(2*np.pi*f1*t1)
x3=np.sin(2*np.pi*f3*t1)
x=np.concatenate([x1,x2,x3])
y=np.concatenate([x3,x2,x1])
#Plotting the result
t=np.linspace(0,1,300)
plt.subplot(2,1,1),plt.plot(t,x),plt.xlabel('Time(t)'),plt.ylabel('Amplitude (V)')
plt.title('Signal-1'),plt.subplot(2,1,2),plt.plot(t,y)
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),plt.title('Signal-2')
plt.tight_layout()
```

Fig. 1.21 Generation of non-stationary signal

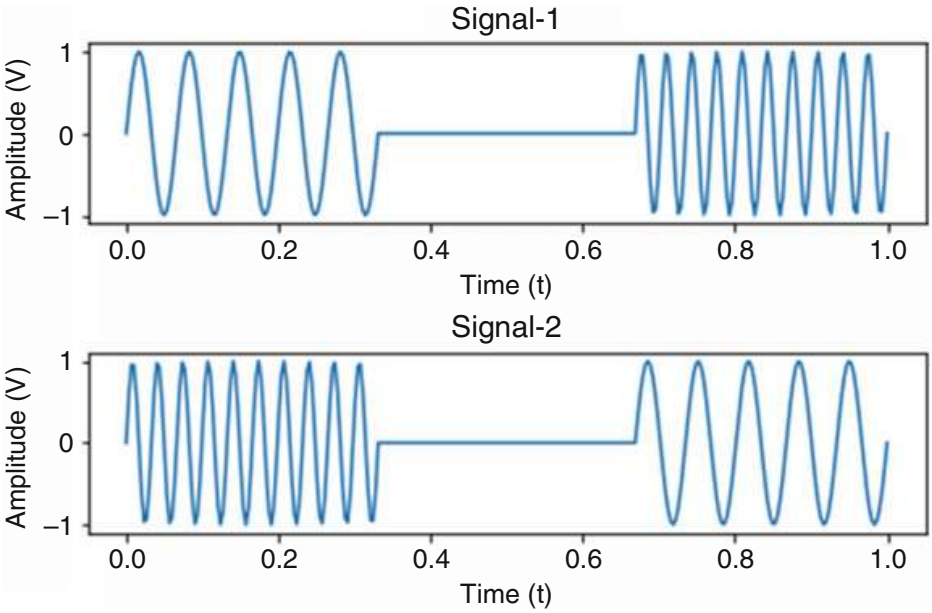


Fig. 1.22 Result of non-stationary signals

Inference

From Fig. 1.24, it is possible to observe that the frequency varies with respect to time in all four types of chirp signals; hence, they are considered non-stationary signals.

```
#Generation of chirp signals
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import chirp
#Step 1: Generation of chirp signal
t=np.linspace(0,10,10000)
x1= chirp(t, f0=10, f1=1, t1=10, method='linear')
x2= chirp(t, f0=10, f1=1, t1=10, method='quadratic')
x3= chirp(t, f0=10, f1=1, t1=10, method='logarithmic')
x4= chirp(t, f0=10, f1=1, t1=10, method='hyperbolic')
#Step 2: Plotting the signals
plt.subplot(2,2,1),plt.plot(t,x1),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),
plt.title('Linear chirp'),plt.subplot(2,2,2),plt.plot(t,x2),plt.xlabel('Time (t)'),
plt.ylabel('Amplitude (V)'),plt.title('Quadratic chirp'),plt.subplot(2,2,3),
plt.plot(t,x3),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),
plt.title('Logarithmic chirp'),plt.subplot(2,2,4),plt.plot(t,x4),plt.xlabel('Time (t)'),
plt.ylabel('Amplitude (V)'),plt.title('Hyperbolic chirp')
plt.tight_layout()
```

Fig. 1.23 Generation of chirp signals

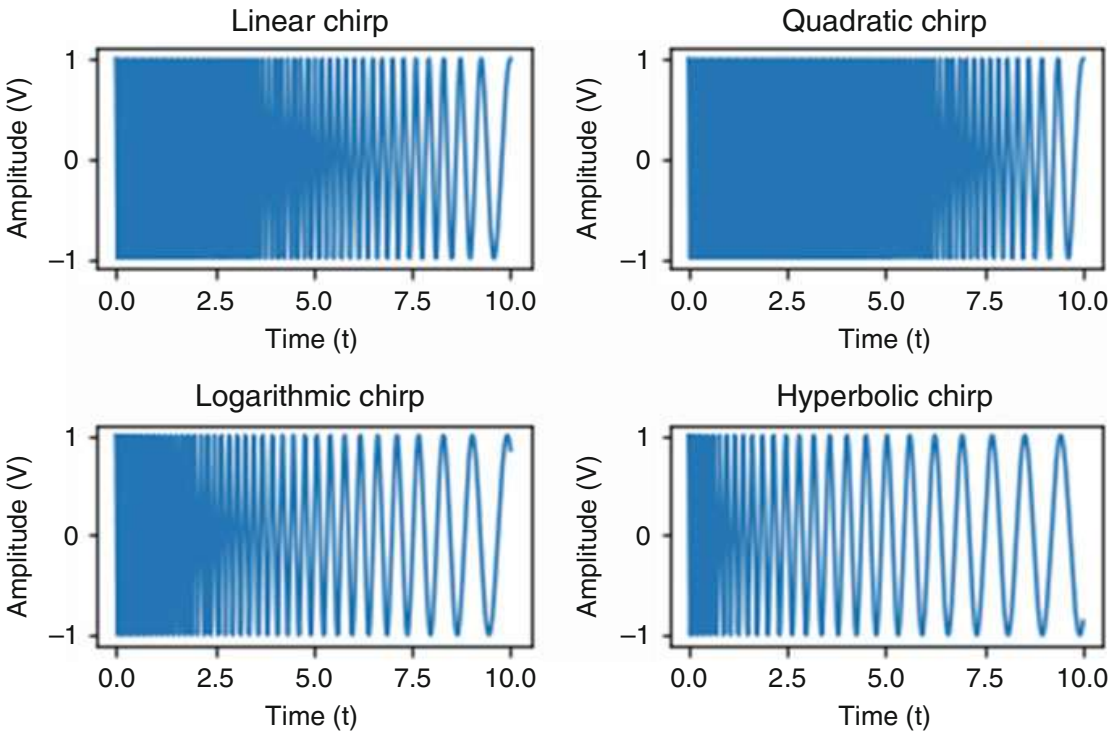


Fig. 1.24 Simulation result of chirp signals

1.3 Non-sinusoidal Waveform

The non-sinusoidal waveform generation considered in this section includes square waveform, triangular waveform, sawtooth waveform, sinc and Gaussian signals.

1.3.1 Square Waveform

A square waveform is a non-sinusoidal periodic waveform. A square wave represents a sudden variation from 'ON' to 'OFF' state and vice versa. Duty cycle is the percentage of time a square wave remains high versus low over one period. Square waves are useful in modelling digital signals. Sine wave contains single frequency, whereas square wave contains a very wide bandwidth of frequencies.

Experiment 1.13 Generation of Square Waveform
The objective is to generate square wave with different duty cycle. The duty option refers to which fraction of the whole duty cycle the signal will be in its 'high' state. The python code, which generates square wave of frequency 5 Hz, is shown in Fig. 1.25, and the corresponding output is shown in Fig. 1.26.

Inference
From Fig. 1.26, it is possible to interpret that the generated waveform is a square waveform of a fundamental frequency of 5 Hz. With increase in the duty, the '*ON time*' of the generated square wave increases. The square waveform takes only binary value, which is either +1 or −1. The state change from +1 to −1 and −1 to +1 occurs immediately.

```
#Square wave with a different duty cycle
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
t=np.linspace(0,1,100)
f=5
duty=[0.15,0.25,0.5,0.75]
for i in range(len(duty)):
   x=signal.square(2*np.pi*f*t,duty[i])
   plt.subplot(2,2,i+1)
   plt.plot(t,x),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
   plt.ylim(-2,2),plt.title('Square wave (duty={})'.format(duty[i]))
   plt.tight_layout()
```

Fig. 1.25 Generation of a square wave

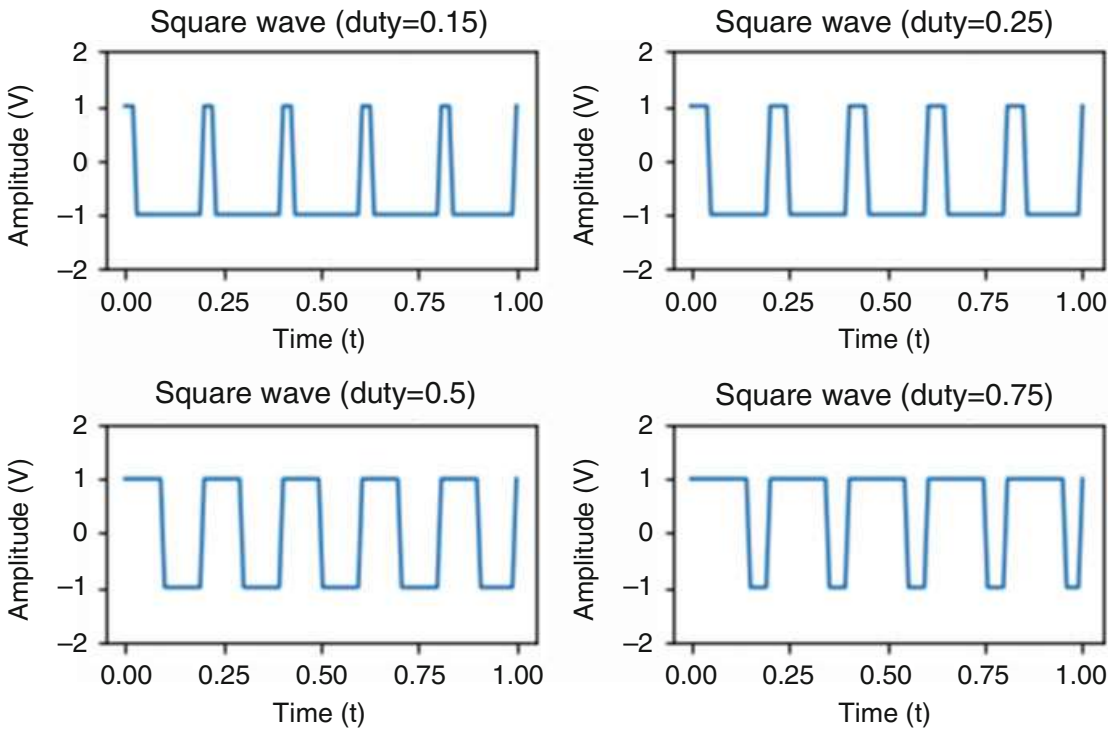


Fig. 1.26 Result of python code shown in Fig. 1.25

1.3.2 Triangle and Sawtooth Waveform

Triangle and sawtooth waveforms are useful for exploring non-linearity in the circuit. A triangle waveform has uniform rise and fall time, whereas in a sawtooth waveform, the rise and fall times are markedly different.

Experiment 1.14 Generation of Sawtooth and Triangular Waveforms
The python code, which generates the sawtooth waveform of frequency 5 Hz, is shown in Fig. 1.27, and the corresponding output is shown in Fig. 1.28.

Inference
When the width is 0.5, the sawtooth waveform turns out to be a triangular waveform. In square waveform, the state change from −1 to +1 and from +1 to −1 occurs instantaneously, whereas, in triangular waveform, the change of state from −1 to +1 and from +1 to −1 occurs gradually.

1.3.3 Sinc Function

A sinc function is represented as

```
#Sawtooth wave with different width
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
t=np.linspace(0,1,1000)
f=5
width=[0.1,0.2,0.5,0.8]
for i in range(len(width)):
   x=signal.sawtooth(2*np.pi*f*t,width[i])
   plt.subplot(2,2,i+1)
   plt.plot(t,x),plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)')
   plt.ylim(-2,2),plt.title('Sawtooth wave (width={})'.format(width[i]))
   plt.tight_layout()
```

Fig. 1.27 Python code to generate sawtooth waveform

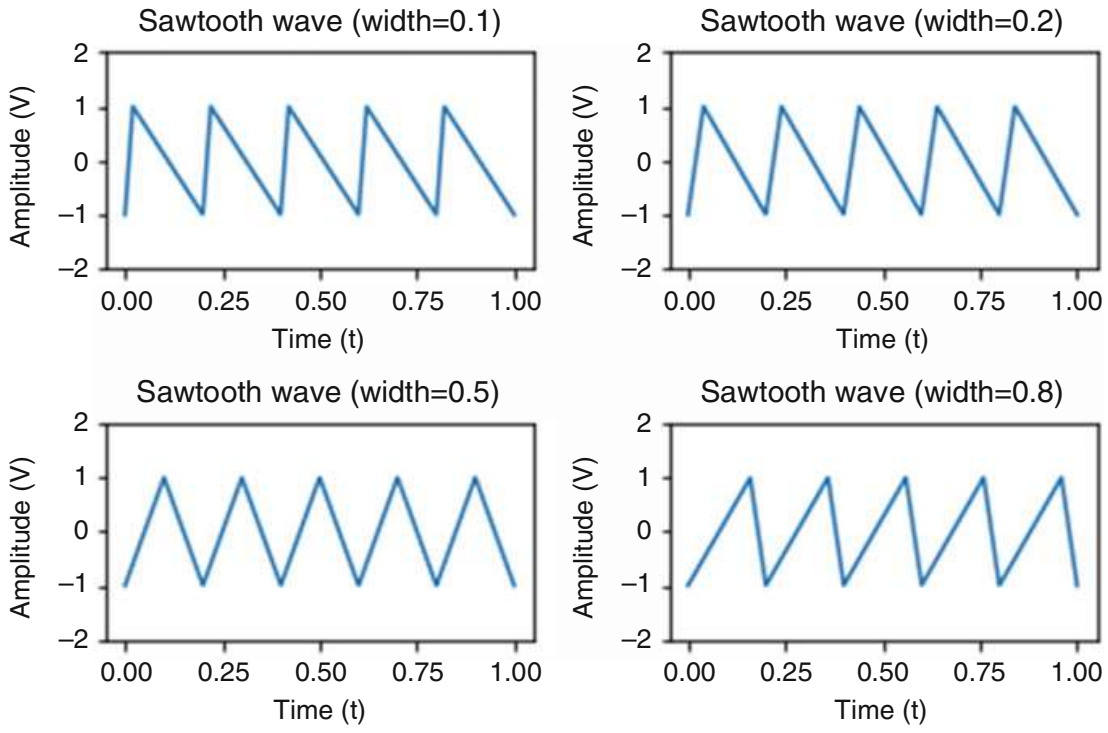


Fig. 1.28 Simulation result of sawtooth waveform

$$x(t) = \frac{\sin \pi t}{\pi t}, \; -\infty < t < \infty \tag{1.9}$$

A sinc function is an even function with a unit area. It is a symmetric function with respect to the origin. Fourier transform of sinc function will result in rectangular function and vice versa. Thus, sinc function is the impulse response of the ideal lowpass filter.

Fig. 1.29 Python code to generate *sinc* function

```
#Generation of sinc function
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Defining the independent variable
t=np.linspace(-10,10,1000)
#Step 2: Generating sinc function
x=np.sinc(t)
#Step 3: Plotting the sinc function
plt.plot(t,x),plt.xlabel('Time'),plt.ylabel('Amplitude')
plt.title('Sinc function')
plt.tight_layout()
```

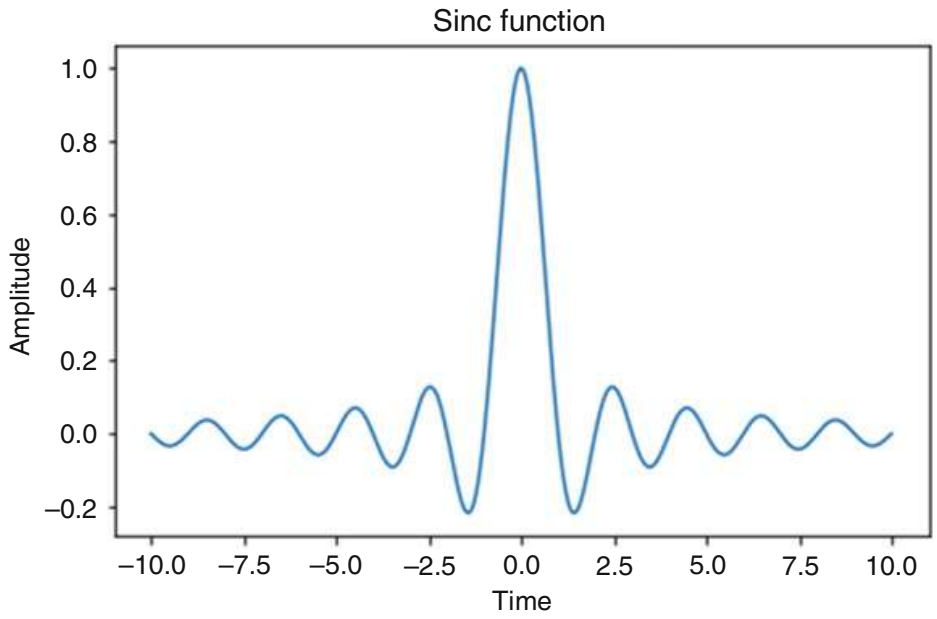


Fig. 1.30 Result of python code shown in Fig. 1.29

Experiment 1.15 Generation of Sinc Function

The expression for sinc function is given by $\sin c(t) = \frac{\sin(\pi t)}{\pi t}$. In this experiment, the sinc function is generated using the built-in function (*sinc*()) available in numpy library. The python code, which generates the sinc function, is shown in Fig. 1.29, and the corresponding output is shown in Fig. 1.30.

Inference

From Fig. 1.30, the following observations can be made:

1. Sinc function has a main lobe and side lobes.
2. The sinc function is symmetric with respect to the origin. It is an even function.
3. The sinc function attains the maximum value at the origin.

Task

1. Write a python code to prove that *sinc* function is an even function.

1.3.4 Pulse Signal

A rectangular pulse can be considered as a positive going edge, followed by negative going one. Convolution of two rectangular pulses results in a triangular pulse.

Experiment 1.16 Generation of Rectangular and Triangular Pulse Signal

The expression for rectangular pulse is given by $x(t) = \begin{cases} 1, & |t| < 1 \\ 0, \text{otherwise} \end{cases}$. It can be considered as a positive going edge followed by negative going one. Rectangular pulse represents a drastic variation from level 0 to 1 and from 1 to 0. The expression for the triangular pulse is $x(t) = \begin{cases} 1 - \frac{|t|}{T}, & \text{for } |t| < T \\ 0, & \text{otherwise} \end{cases}$. The triangular pulse represents a gradual variation from level 0 to 1 and from 1 to 0.

The python code, which generates the rectangular and triangular pulse signal, is shown in Fig. 1.31, and the corresponding output is shown in Fig. 1.32.

Inference

Figure 1.32 shows that rectangular pulse exhibits a drastic change in amplitude from 0 to 1 V, whereas triangular pulse exhibits a gradual variation in amplitude from 0 to 1 V. In later section, it will be proved that the convolution of two rectangular pulses will result in a triangular pulse.

```
#Generation of rectangular and triangular pulse signal
import numpy as np
import matplotlib.pyplot as plt
#Step 1: Generation of signals
t=np.linspace(-2,2,100)
rect_pulse=abs(t)<1  #Rectangular pulse
tri_pulse=(1 - abs(t)) * (abs(t) < 1) #Triangular pulse
#Step 2: Plotting of the pulse signals
plt.subplot(2,1,1),plt.plot(t,rect_pulse)
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),plt.title('Rectangular pulse')
plt.subplot(2,1,2),plt.plot(t,tri_pulse)
plt.xlabel('Time (t)'),plt.ylabel('Amplitude (V)'),plt.title('Triangular pulse')
plt.tight_layout()
```

Fig. 1.31 Generation of rectangular and triangular pulse

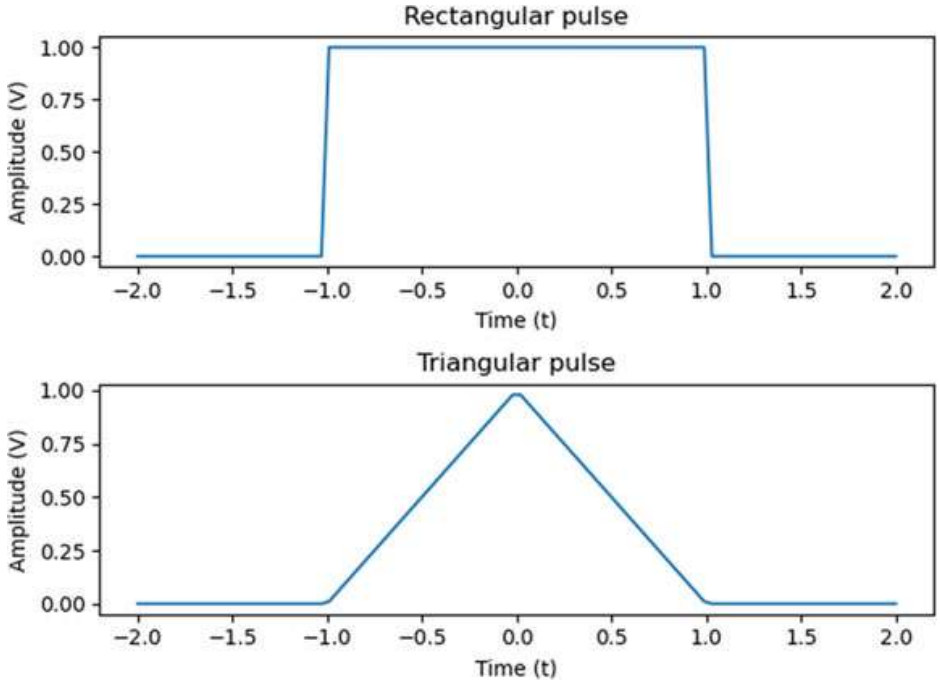


Fig. 1.32 Rectangular and triangular pulse signal

Task

1. Write a python code to illustrate the fact that convolution of two rectangular pulse signals results in a triangular pulse.

1.3.5 Gaussian Function

The Gaussian function is expressed as

$$x(t) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{1.10}$$

where 'μ' represents the mean and 'σ' represents the standard deviation. Fourier transform of a Gaussian function results in another Gaussian function. The product of two Gaussian functions is a Gaussian function. Gaussian window is an optimal window for time-frequency localization. Smoothening by Gaussian function is widely employed in image processing.

Experiment 1.17 Generation of Gaussian Function

The Gaussian function is widely used in signal processing, image processing and communication fields. The expression for Gaussian function with the mean value 'μ' and standard deviation 'σ' is given by $x(t) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(t-\mu)^2}{2\sigma^2}}$. This experiment aims to

```
#Generation of Gaussian function
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(-10,10,1000)
mu=0   #Mean value
sigma=[0.01,0.5,1,10] #Standard deviation
for i in range(len(sigma)):
   k=1/np.sqrt(2*np.pi*sigma[i])
   x=k*np.exp(-np.power(t-mu,2.)/2*np.power(sigma[i],2.))
   plt.subplot(2,2,i+1),plt.plot(t,x),plt.xlabel('Time'),
   plt.ylabel('Amplitude'),plt.title('$\sigma={} $'.format(sigma[i]))
   plt.tight_layout()
```

Fig. 1.33 Gaussian function for different values of standard deviation

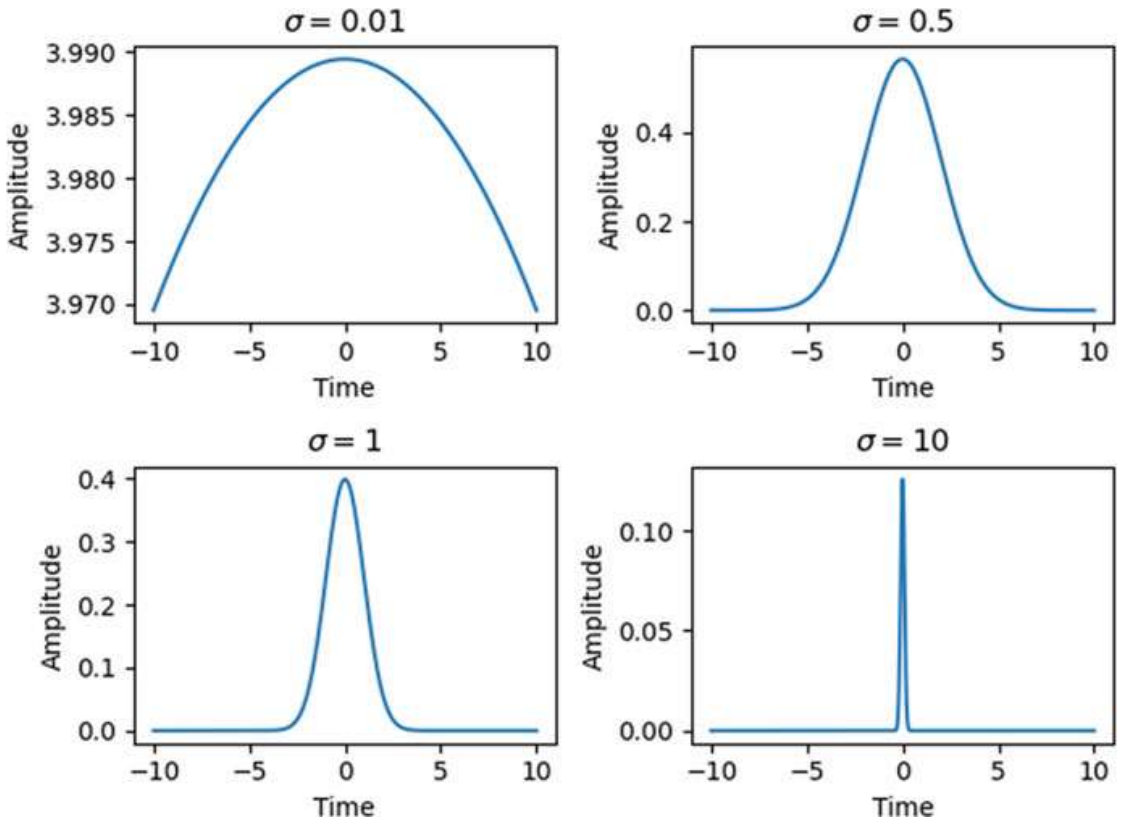


Fig. 1.34 Gaussian function for different values of standard deviation

generate Gaussian function for different values of standard deviation, namely, 0.01, 0.5, 1 and 10. The mean value is taken as zero. The python code, which generates the Gaussian function, is shown in Fig. 1.33, and the corresponding output is shown in Fig. 1.34.

Inference

From Fig. 1.34, it is possible to observe the following facts:

Fig. 1.35 Python code to generate sinusoidal note

```
#Hearing sinusoidal signal
import numpy as np
import sounddevice as sd
f=1000 #Signal frequency
fs=8000 #Sampling rate
t=np.linspace(0,1,fs)
x=np.sin(2*np.pi*f*t)
sd.play(x,fs)
```

1. The Gaussian function is characterized by two parameters, which are mean and standard deviation.
2. The mean value of the Gaussian function is zero; hence, the maximum value occurs at the origin.
3. With an increase in the value of standard deviation, the narrower the Gaussian function.

Task

1. Write a python code to prove that the multiplication of two Gaussian functions results in a Gaussian function.

Experiment 1.18 Hearing a Sinusoidal Signal

Human ears can hear sound in the frequency range from 20 Hz to 20 kHz. In this experiment, sine wave of particular frequency is heard as a tone. The sampling frequency is chosen as 8000 Hz, and the signal frequency is chosen as 1000 Hz. The library functions used are (1) Numpy and (2) Sounddevice. The built-in function in sound device library (*sd.play*) is used to play the sound. The python code, which generates the sinusoidal tone, is shown in Fig. 1.35. The user can hear the audio using headphone.

Inference

From Fig. 1.35, the following inferences can be made:

1. The signal frequency is 1000 Hz, and the sampling frequency is 8000 Hz.
2. The library used to hear the audio is '*sounddevice*' library.
3. The built-in function (*sd.play*) is used to hear the audio.

Task

1. Human ear can hear an audio signal whose frequency is between 20 Hz and 20 kHz. Generate 10 Hz sinusoidal waveform; try to hear the waveform. It should not be audible. Now increase the frequency of sine wave to 100 Hz; now it should be possible to hear the sinusoid as a single note.

Experiment 1.19 Hearing Amplitude Modulated Sinusoidal Signal

The impact of modulating the amplitude of the sinusoidal signal is observed in this experiment. In this experiment, the amplitude of the sinusoidal signal is modulated by both exponentially decaying and growing functions. The python code, which

Fig. 1.36 Amplitude modulated sinusoidal signal

```
#Hearing amplitude modulated sinusoidal signal
import numpy as np
import sounddevice as sd
f=1000 #Signal frequency
fs=8000 #Sampling rate
a=-5  #Decaying factor
b=5   #Growing factor
t=np.linspace(0,1,fs)
x1=np.exp(a*t)*np.sin(2*np.pi*f*t)
x2=np.exp(b*t)*np.sin(2*np.pi*f*t)
sd.play(x1,fs)
sd.wait()
sd.play(x2,fs)
```

Table 1.4 Built-in functions used in Experiment 1.19

S. No.	Built-in function used	Purpose
1	np.exp()	To generate an exponential function
2	np.sin()	To generate a sinusoidal function
3	sd.play()	To play the audio signal
4	sd.wait()	To pause the audio signal

performs this task is shown in Fig. 1.36. The built-in functions used in the program are summarized in Table 1.4.

Inference
The following inferences can be made from Fig. 1.36:

1. The signal 'x1' refers to a sine wave modulated by an exponentially decaying function.
2. The signal 'x2' refers to a sine wave modulated by an exponentially growing function.

Experiment 1.20 Generation of Amplitude Modulated Signal
In amplitude modulation, the amplitude of the carrier signal is varied in accordance with the message signal. The expression for amplitude modulated signal is given by

$$x(t) = (1 + m\sin(2\pi f_m t))\sin(2\pi f_c t) \tag{1.11}$$

In the above expression, 'm' denotes the modulation index, f_m represents the frequency of the modulating signal and f_c denotes frequency of the carrier signal. The python code, which generates the amplitude modulated signal for different modulating indices, is shown in Fig. 1.37, and the corresponding output is shown in Fig. 1.38.

Inference
From Figs. 1.37 and 1.38, the following inferences can be drawn:

```
#Amplitude modulation
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,1,1000)
fm=10 #Frequency of modulating signal
fc=100 #Frequency of carrier signal
message=np.sin(2*np.pi*fm*t)
carrier=np.sin(2*np.pi*fc*t)
m=[0.25,0.5,1,1.5]  #modulation index
for i in range(len(m)):
    mod_sig=(1+m[i]*message)*carrier
    plt.subplot(2,2,i+1),plt.plot(t,mod_sig)
    plt.xlabel('Time'),plt.ylabel('Amplitude')
    plt.title('Modulated signal with m={}'.format(m[i]))
    plt.tight_layout()
```

Fig. 1.37 Python code to generate amplitude modulated signal

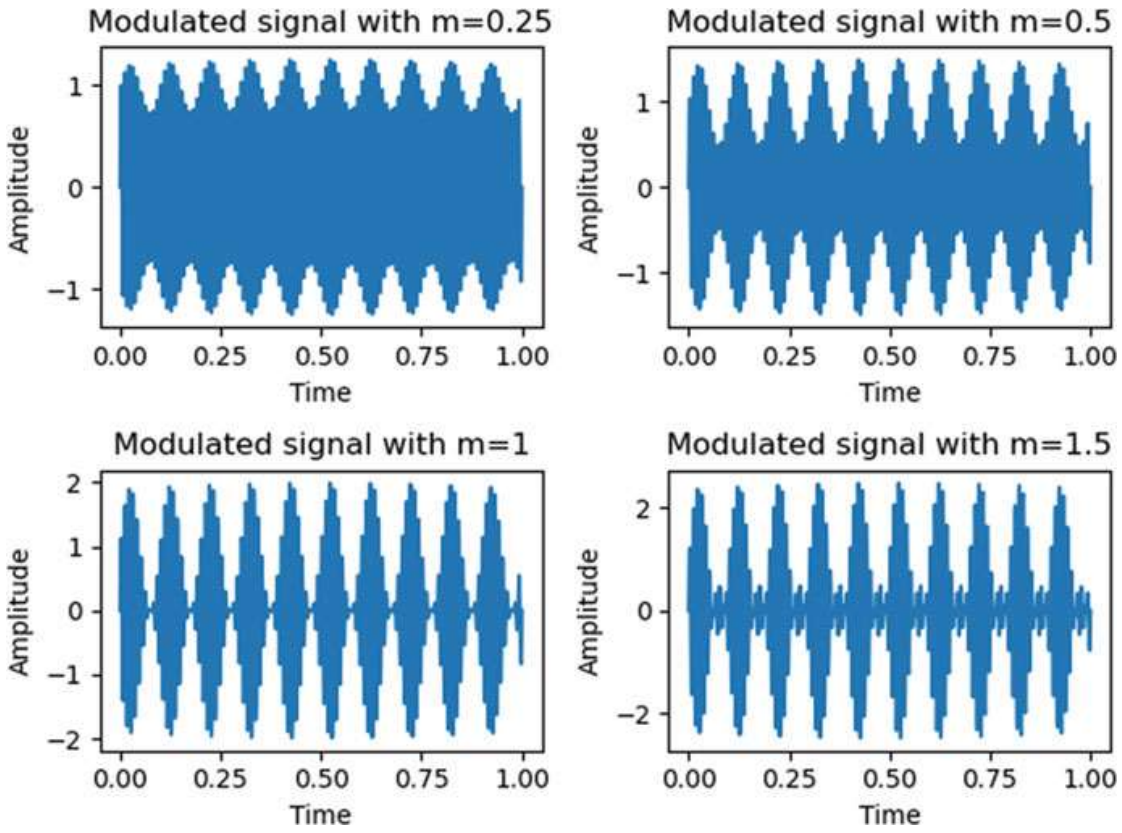


Fig. 1.38 Amplitude modulated signal with different modulation indices

1. The frequency of the message signal is 10 Hz; the frequency of the carrier signal is 100 Hz. The modulation index is varied as 0.25, 0.5, 1.0 and 1.5.
2. It is possible to observe that the amplitude of the carrier is changed in accordance with the message signal.

3. Modulation index less than one corresponds to under modulated signal. Modulation index greater than one corresponds to over modulated signal. Modulation index equal to one corresponds to perfect modulation.

Exercises

1. Generate the following sinusoidal signal $x(t) = A \sin(2\pi ft + \phi)$ with the amplitude $A = 2$ V, frequency $f = 10$ Hz and phase $\phi = 0$. Let the length of the signal be 100 samples. Store this signal in your system in a particular folder along with the time stamp in an Excel sheet. From the Excel sheet, read the data and the time stamp and plot the signal.
2. Write a python code to generate the sinusoidal signal of 1 V amplitude, 5 Hz frequency and phase $\phi = 0$. Mark the positive peak of the waveform. That is the positive peak of the waveform should be marked with 'x' mark.
3. Write a python code to compute the number of zero crossings of sine wave of 2 V amplitude, 5 Hz frequency and phase $\phi = 0$.
4. Write a python code to generate the seven nodes '*sa*', '*re*', '*ga*' and '*ma'*. Use the sounddevice library to play the seven notes.
5. Generate the Gaussian function, which is given by $x(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}}$ for different mean values $\mu = 0, 1, 2, 4$ with the fixed standard deviation value $\sigma = 1$. Use subplot to plot the generated Gaussian functions.

Objective Questions

1. What will the signal's length be if the following code is executed?

```
t=np.linspace(0,1,100)
x=np.sin(2*np.pi*5*t)
```

   A. 10
   B. 50
   C. 75
   D. 100

2. What will be the magnitude of the variables '*x*' and '*y*' if the following code segment is executed?

```
t=np.linspace(0,1,100)
f=0
x=np.sin(2*np.pi*f*t)
y=np.cos(2*np.pi*f*t)
```

   A. 1 and 0, respectively
   B. −1 and 1, respectively
   C. 0 and 1, respectively
   D. 1 and −1, respectively

3. What will be the output plot if the following segment of code is executed?

```
t=np.linspace(0,1,100)
x=np.sin(2*np.pi*5*t)
y=np.cos(2*np.pi*5*t)
plt.plot(t,(x**2+y**2))
```

A. DC signal of magnitude 1
B. DC signal of magnitude 5
C. Sine wave of frequency 5 Hz
D. Cosine wave of frequency 5 Hz

4. What will be stored in the variable '*z*' if executing the following code segment?

```
t=np.linspace(0,1,100)
x=np.exp(1j*2*np.pi*5*t)
y=np.exp(-1j*2*np.pi*5*t)
z=(x+y)/2
```

A. Sine wave of 5 Hz frequency
B. Cosine wave of 5 Hz frequency
C. Square wave of 5 Hz frequency
D. Sawtooth wave of 5 Hz frequency

5. The phase difference between each signal in a three-phase sinusoidal signal is

A. 45°
B. 90°
C. 120°
D. 240°

6. What will be stored in the variable '*z*' if executing the following code segment?

```
t=np.linspace(0,1,100)
x=np.exp(1j*2*np.pi*5*t)
y=np.abs(x)
```

A. Phase angle of the signal 'x'
B. Magnitude of the signal 'x'
C. Frequency of the signal 'x'
D. Number of zero crossings of the signal 'x'

7. The audible frequency range for human beings is

A. 10 Hz to 100 kHz
B. 20 Hz to 20 kHz
C. 1 to 1000 Hz
D. 200 Hz to 2 MHz

8. What will the signal's length be if the following code segment is executed?

```
fs=8000
t=np.linspace(0,1,fs)
```

A. 1000
B. 2000
C. 4000
D. 8000

9. Identify the statement that is WRONG with respect to sinc function

   A. Sinc function is an even function.
   B. Sinc function is an odd function.
   C. Fourier transform of sinc function will result in a rectangular function.
   D. Sinc function can be used for signal interpolation.

10. The magnitude of the function $x(t) = \mathrm{e}^{-j\Omega t}$ is

    A. 1
    B. 0
    C. -1
    D. Infinity

Bibliography

1. Alan V. Oppenheim, and Alan S. Willsky. “Signals and Systems”, Prentice Hall, 1996.
2. Simon Haykin, and Bary Van Veen, “Signals and Systems”, Wiley, 2005.
3. Hwei P. Hsu, “Signals and Systems”, Schaum’s outline series, McGraw Hill Education, 2017.
4. Charles L. Phillips, John M. Parr, and Eve A. Riskin, “Signals, Systems, and Transforms”, Pearson, 2013.
5. Mark Lutz, “Learning Python”, O’Reilly Media, 2013.