Workshop 3: Linear, Time-Invariant Systems

Objective: To analyze linear, time-invariant systems.

Outcome: After successful completion of this session, the student would be able to

1. Analyse continuous time systems using the convolution integral.
2. Analyse discrete time systems using the convolution sum.

Equipment Required:

1. A personal computer.
2. Python with NumPy, SciPy, adn Matplotlib

Components Required: None.

3.1 Continuous-Time Systems: Convolution Integral

We have

$$\boxed{y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau}$$

which is referred to as the convolution integral or the superposition integral. This corresponds to the representation of a continuous-time LTI system in terms of its response to a unit impulse.

$$y(t) = x(t) * h(t).$$

A continuous-time LTI system is completely characterized by its impulse response—i.e., by its response to a.single elementary signal, the unit impulse $\delta(t)$.

3.1.1 Implementing Convolution Using Numerical Integration

Let $x(t)$ be the input to an LTI system with unit impulse response $h(t)$, where

$$x(t) = e^{-at}u(t), a > 0$$

and

$$h(t) = u(t).$$

We wish to compute the output

$$y(t) = x(t) * h(t)$$

obtained when $x(t)$ is fed to the system represented by $h(t)$. Fig. 3.1 shows $x(t)$, $h(t)$ and related signals.

For $t < 0$, the product $x(\tau)$ and $h(t-\tau)$ is zero, consequently $y(t)$ is zero.

For $t > 0$,

$$x(\tau)h(t-\tau) = \begin{cases} e^{-a\tau}, & 0 < \tau < t, \\ 0, & \text{otherwise.} \end{cases}$$

$$y(t) = \int_0^t e^{-a\tau}d\tau = -\frac{1}{a}e^{-a\tau}\Big|_0^t$$
$$= \frac{1}{a}(1 - e^{-at})$$

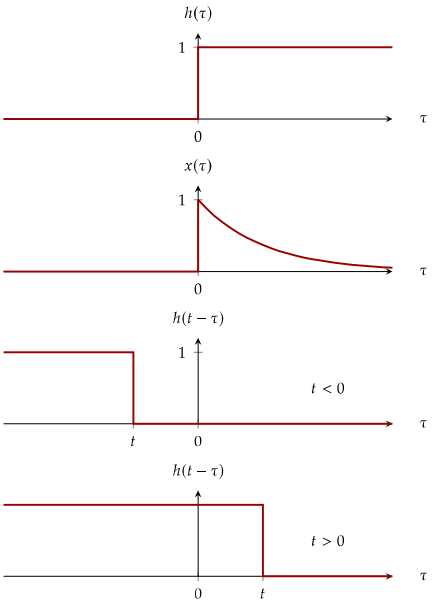


Figure 3.1: Calculation of convolution integral for the example

Thus for all t,

$$y(t) = \frac{1}{a}(1 - e^{-at})u(t)$$

Fig. 3.2 shows graph of the output.

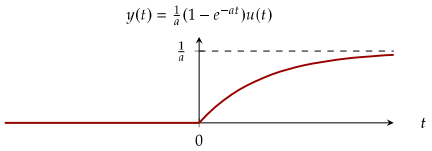


Figure 3.2: Response

The listing 3.1 shows the computation of the convolution integral to compute $y(t)$. Note that we have to use numerical integration.

```
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt
h = lambda t: (t > 0)*1.0
x = lambda t: (t > 0) * np.exp(-2*t) # a = -2
Fs = 50  # Sampling frequency for the plotting
T = 5    # Time range
t = np.arange(-T, T, 1/Fs)  # Time samples

plt.figure(figsize=(8,3))
plt.plot(t, h(t), label='$h(t)$')
plt.plot(t, x(t), label='$x(t)$')
plt.xlabel(r'$t$')
plt.legend()

# Plotting
t_ = 1 # For illustration, choose some value for t
```

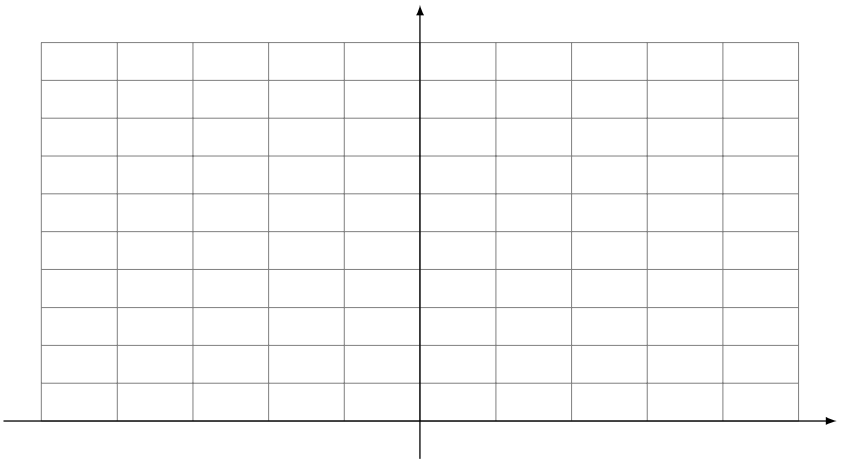
```
flipped = lambda tau: h(t_ − tau)
product = lambda tau: x(tau)*h(t_ − tau)
plt.figure(figsize=(8,3))
plt.plot(t, x(t), label=r'$x(\tau)$')
plt.plot(t, flipped(t), label=r'$h(t − \tau)$')
plt.plot(t, product(t), label=r'$x(\tau)h(t −\tau)$')

# Computing the convolution using integration
y = np.zeros(len(t))
for n, t_ in enumerate(t):
    product = lambda tau: x(tau) * h(t_ − tau)
    y[n] = integrate.simps(product(t), t) # Actual convolution at time t

plt.plot(t, y, label=r'$x(t)\ast h(t)$') # Plotting the output y
plt.xlabel(r'$t$')
plt.legend()
```

Listing 3.1: Computing the Convolution Integral

Task 1. *Sketch the output of the listing 3.1 above and comment.*



3.1.2 Convolving with a Signal Composed of Impulse Functions

We approximated impulse function $\delta(t)$ as

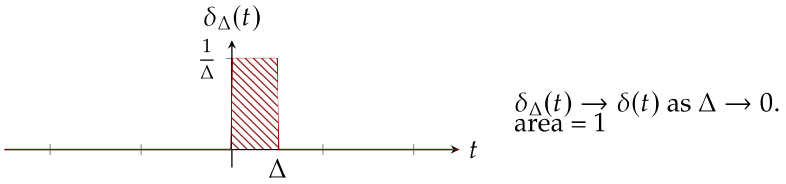


Figure 3.3: Approximation of $\delta(t)$.

Consider the following simple approximate implementation of $\delta(t)$.

```
fs = 1000  # Sampling frequency for the plotting
delta = lambda t: np.array([fs/10 if 0 < t_  and t_ < 1/(fs/10) else 0.0 for t_ in t])
```

Listing 3.2: A Simple Implementation of the Impulse Function

Task 2. *Use Simpson's rule integration (as shown in Listing 3.1), obtain the value of*

$$\int_{-\infty}^{\infty} \delta(t)dt.$$

Consider

$$x(t) = e^{-at}u(t), a > 0,$$

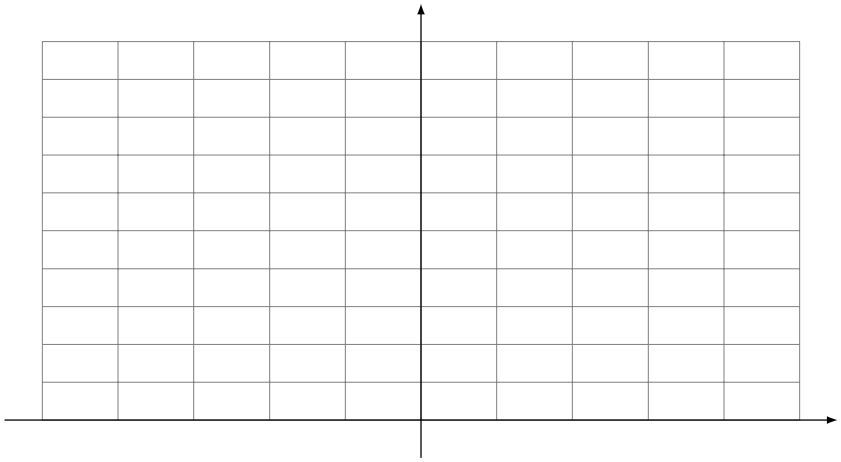
and

$$h(t) = \delta(t+2) + \delta(t-1).$$

Task 3. *Write an expression for*

$$y(t) = x(t) * h(t).$$

Task 4. *Compute $y(t)$ in a similar fashion as in Listing 3.1 and sketch.*



3.2 Discrete-Time Systems: Convolution Sum

Using the convolution we can express the response of an LTI system to an arbitrary input in terms of the system's response to the unit impulse. An LTI system is completely characterized by its response to a single signal, namely, its response to the unit impulse.

The convolution of the sequence $x[n]$ and $h[n]$ is given by

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k], \tag{3.1}$$

which we represent symbolically as

$$y[n] = x[n] * h[n] \tag{3.2}$$

Fig. 3.4 shows $x[n]$ and $h[n]$ to be convolved. Fig. 3.5 illustrates how the convolution is implemented. Note

Figure 3.4: $x[n]$ and $h[n]$ for convolution implementation.

that there can be an overlap between $x[k]$ and $h[n-k]$ only when $n \in [-5, 5]$. Therefore, length of the array

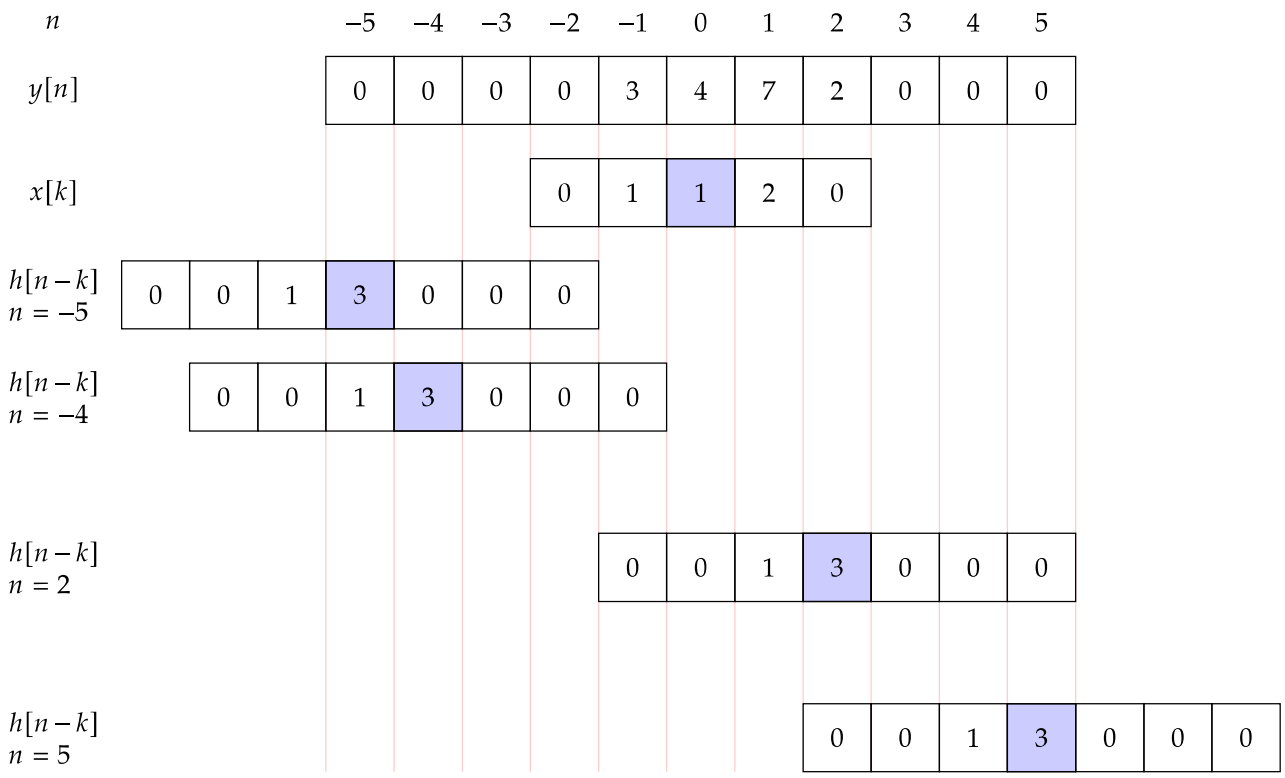


Figure 3.5: Illustration of convolution implementation.

for $y[n]$ is len(x) + len(h) − 1. We must carefully estimate the indices of lower and upper limits of overlapping region for each x and h. The listing 3.3 shows the actual implementation of the convolution sum using NumPy arrays. Note that we cannot have negative indices. Therefore, the implementation has a difference from Fig. 3.5.

```
x = np.array([0, 1, 1, 2, 0])
h = np.array([0, 0, 0, 3, 1, 0, 0])
hr = np.flip(h)
xo = 2
ho = 4
y = np.zeros(len(x) + len(h) – 1)
for n in range(len(y)):
   xkmin = max(0, n – len(h) + 1)
   xkmax = min(len(x), n + 1)
   hkmin = max(0, len(h) – n –1)
   hkmax = min(len(h), len(x) + len(h) –n –1)
   y[n] = np.sum(x[xkmin:xkmax]*hr[hkmin:hkmax])
   print("y[{0}] =  x[{1}:{2}]*h[{3}:{4}] = {5}".format(n, xkmin, xkmax, hkmin, hkmax, y[n]))
```

Listing 3.3: Implementation of DT convolution.

Task 5. *Sketch the output of the listing 3.1 above and comment.*

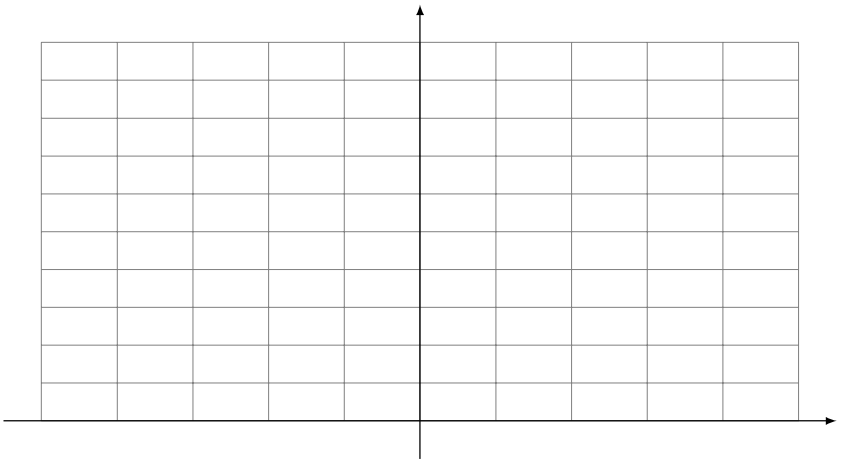


Fig. 3.6 shows $x[n]$ and $h[n]$ to be convolved.

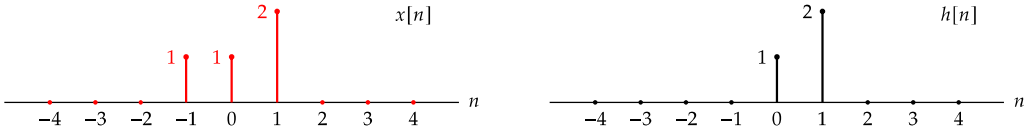


Figure 3.6: $x[n]$ and $h[n]$ for convolution.

Task 6. *Study the listing 3.3 and use a for-loop to compute the convolution sum at each* $n \in [-4, 4]$

Task 7. *Use scipy.signal.convolve*[1] *to compute the above convolution. Describe the effect of modes full, valid, and same.*

3.3 An Application in Audio Signal Filtering

The following code 3.4 is for reading a .wav file and generating the filter coefficients of a so-called finite impulse response (FIR) filter. We can consider these coefficients as the impulse response of the filter and compute the filtered output using convolution. It also plots the frequency response of the filter. Note that frequencies are normalized. The signal can be written to disk as a .wav file as shown at the end of the code snippet.

```
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
data, samplerate = sf.read('audio_file.wav')

nyquist = samplerate//2
fc = 2000/nyquist
n = 121
b = signal.firwin(n, fc, pass_zero=True)
w, h = signal.freqz(b)
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
ax1.set_title('Digital filter frequency response')
ax1.plot(w, 20 * np.log10(abs(h)), 'b')
ax1.set_ylabel('Amplitude [dB]', color='b')
ax1.set_xlabel('Frequency [rad/sample]')
ax2 = ax1.twinx()
angles = np.unwrap(np.angle(h))
```

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html

```
ax2.plot(w, angles, 'g')
ax2.set_ylabel('Angle (radians)', color='g')
ax2.grid()
ax2.axis('tight')
plt.show()


% Your code here for convolution.

sf.write('audio_file_filtered.wav', np.vstack((ch1, ch2)).T + data, samplerate)
```

Listing 3.4: Filtering using convolution.

Task 8. *Noting that data may have a pair of channels (stereo) use convolution to filter the audio signal.*

Task 9. *Creatively achieve various filtering effects.*

3.4 Convolution Sum in 2-D

In 2-D, as applicable in image processing, convolution sum with a kernel $h[m,n]$ with non-zero values in $(m,n) \in ([-a,a],[-b,b])$ is

$$(h * x)[m,n] = h[m,n] * x[m,n] = \sum_{s=-a}^{a} \sum_{k=-b}^{b} h[s,t]x[m-s,n-t].$$

Consider the "image"

$$x[m,n] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the filtering kernel

$$h = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Note that there in only one pixel in the image which is non-zero.

Task 10. *Convolve the image $x[m,n]$ with filter $h[m,n]$. Hint: Use signal.convolve2d.*

Task 11. *Interpret the above result.*

3.5 Application: Using Convolution to Filter an Image

Fig. 3.7 shows an image of a set of Allen keys. In this section, we will filter this image with a filtering kernel called the Sobel horizontal kernel. We can read an image using the following snippet in 3.5.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
x = mpimg.imread('allenkeys.png')
fig, ax = plt.subplots(1,2)
ax[0].imshow(x, cmap='gray')
```

Listing 3.5: Image reading.

Figure 3.7: Allen keys.

Task 12. *Filter this image with the kernel*

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$