Part II

Signals Circuits and Systems

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 $t > 0$,

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

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

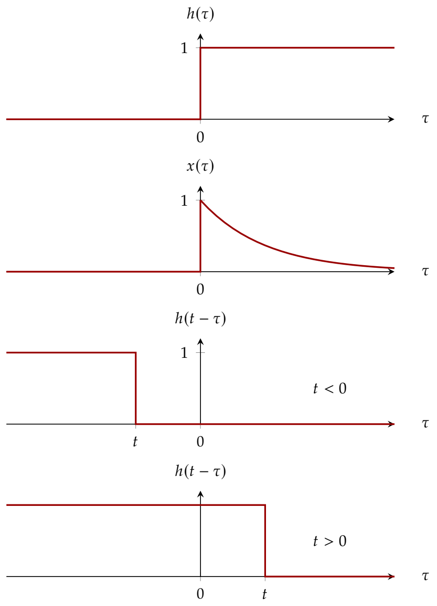


Figure 3.1: Calculation of convolution integral for the example

Thus for all $t$,

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

Fig. 3.2 shows graph of the output.

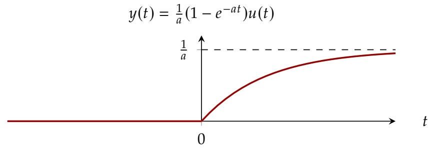


Figure 3.2: Response

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
```

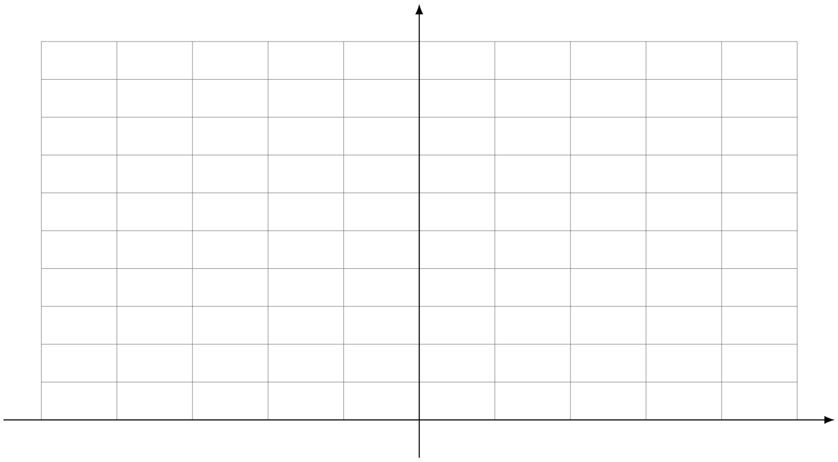
```
flipped = lambda tau: h(t_ − tau)
product = lambda tau: x(tau)*h(t_ − tau)
plt.figure(figsize=(8,3))
plt.plot(t, x(t), label=r'$x(\tau)$')
plt.plot(t, flipped(t), label=r'$h(t − \tau)$')
plt.plot(t, product(t), label=r'$x(\tau)h(t −\tau)$')

# 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()
```

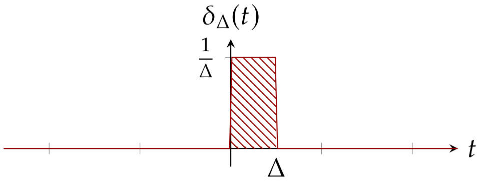
Listing 3.1: Computing the Convolution Integral

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



$\delta_\Delta(t) \rightarrow \delta(t)$ as $\Delta \rightarrow 0$.
area = 1

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

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

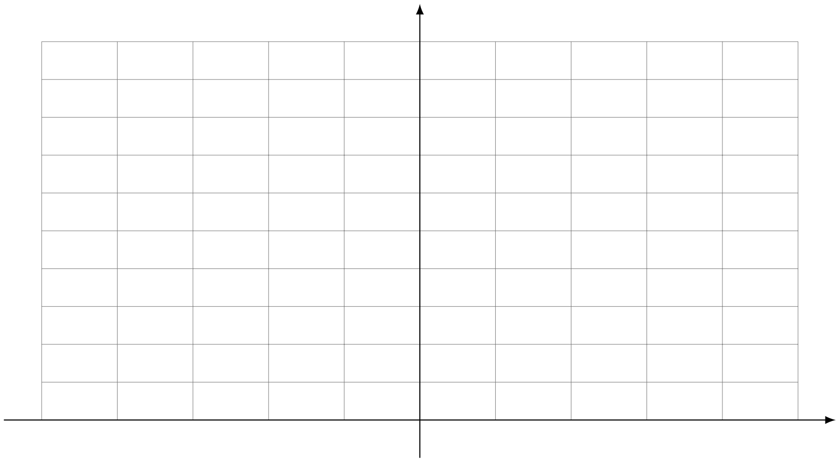
and

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

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

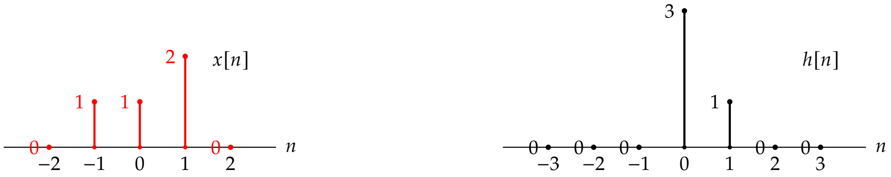


Figure 3.4: $x[n]$ and $h[n]$ for convolution implementation.

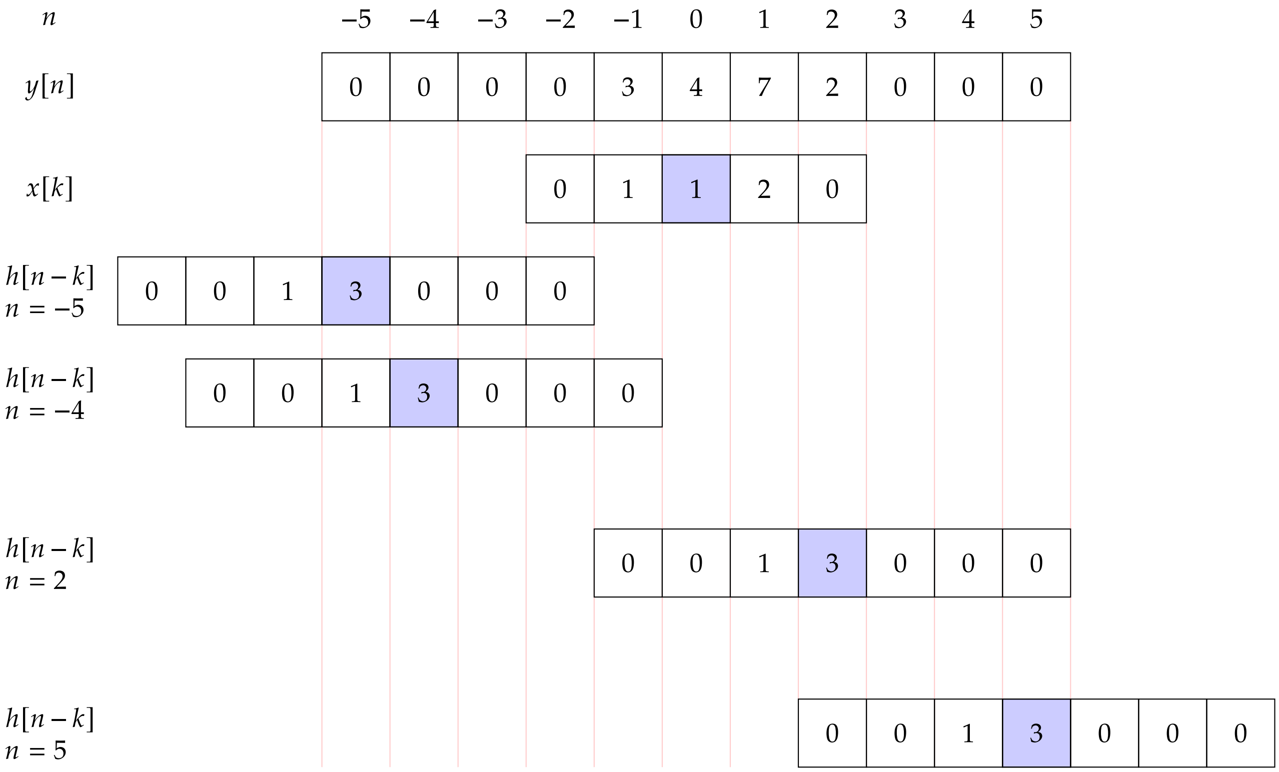


Figure 3.5: Illustration of convolution implementation.

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.*

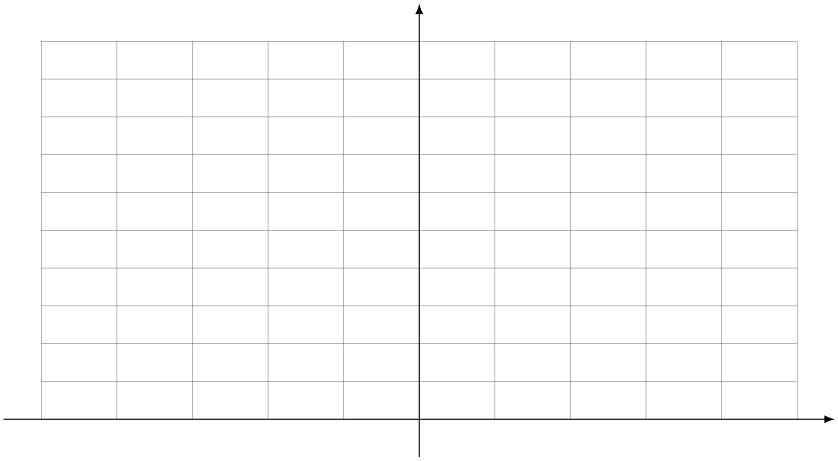


Fig. 3.6 shows $x[n]$ and $h[n]$ to be convolved.

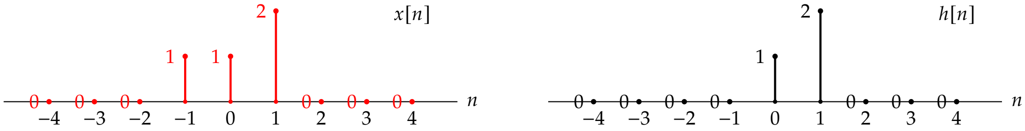


Figure 3.6: $x[n]$ and $h[n]$ for convolution.

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 impels 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 frequented 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.

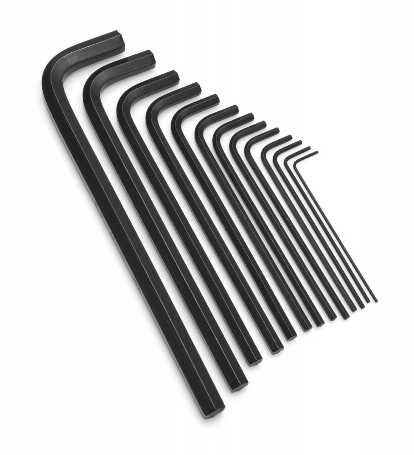


Figure 3.7: Allen keys.

Task 12. *Filter this image with the kernel*

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

Part III

Electronics

Workshop 3: Simple Zener Regulated DC Power Supply

Objective: To construct a simple Zener regulated DC power supply

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

1. Construct and test a Half Wave Rectifier (HWR)
2. Construct and test a Full Wave Rectifier (FWR)
3. Identify the effect of a capacitive filter on FWR output
4. Construct and test a Zener regulated DC power supply

Equipment Required:

1. 6 V/50 Hz AC source
2. Digital Oscilloscope
3. Analog multimeter
4. Breadboard and wires

Components Required:

1. Diodes: 1N4001 (4 nos.)
2. Zener Diode: 1N4732A
3. Bridge rectifier IC
4. Capacitors: 4.7μF, 220μF
5. Resistors: 10 kΩ, 1kΩ, 560Ω, 180Ω, 100Ω (2 nos.)

3.1 Half Wave Rectifier (HWR)

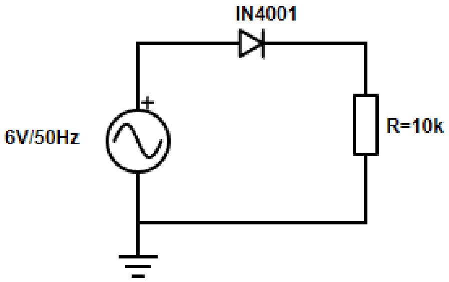


Figure 3.1: Schematic diagram of the HWR

Task 1. *Construct the circuit show in figure 3.1 on the breadboard with a 10 kΩ resistor as the load. Provide an AC input using the 6V/50Hz AC source.*

Task 2. *Observe the input AC voltage and the load voltage from the oscilloscope and draw the wave forms in the same diagram.*

Task 3. *Why do you think full-wave rectification is preferred instead of half-wave rectification?*

3.2 Bridge Rectifier (FWR)

The bridge rectifier in figure 3.2 is a full wave rectifier and converts AC to DC. The AC input is applied across A and B while the DC output is taken across P and Q.

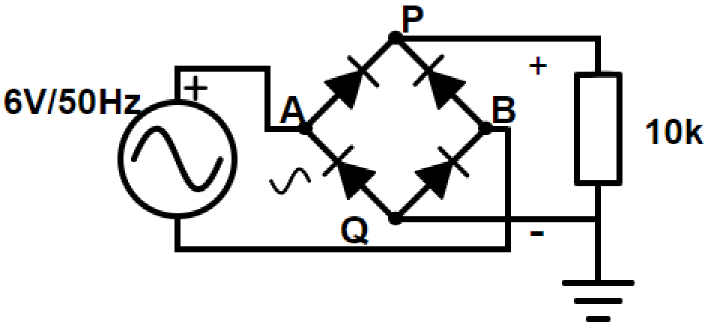


Figure 3.2: Schematic diagram of the FWR

The bridge rectifier is available in IC form as well. The AC inputs of the IC are usually marked with a ∼ sign and the DC outputs are marked with their corresponding polarity signs (+ and −).

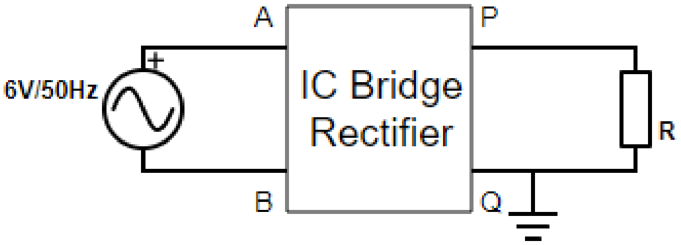


Figure 3.3: Bridge Rectifier IC

Task 4. *Fix the IC on the breadboard and apply a 6 V/50 Hz voltage (from the AC source) across the input terminals AB of the IC.*

Task 5. *Connect a 10 kΩ load resistor across the output of terminals PQ.*

Task 6. *Observe the AC input voltage across AB from the oscilloscope and draw the waveform.*

Task 7. *Observe the DC output voltage across PQ from the oscilloscope and draw the waveform.*

Task 8. *Using the multimeter, measure the DC voltage (V_{DC}) of the output.*

Task 9. *Measure the peak value of the output voltage (V_{max}) from the oscilloscope and estimate V_{DC} (Recall that, for a sinusoid, V_{DC}=$(2/\pi)V_{max}$).*

3.3 Capacitive Filter

The output of the rectifier is a varying DC voltage. It is almost always necessary to have steady DC voltages to get electronic circuits to function properly. Therefore, different smoothing circuits or filter circuits are used to produce a steady DC voltage at the rectifier output. The simplest filter is a capacitor.

Task 10. *Connect the 4.7 μF capacitor across the output of the rectifier as shown in figure 3.4.*

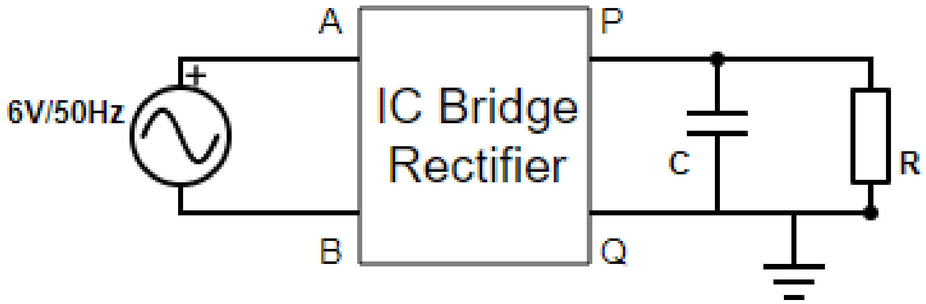


Figure 3.4: FWR with smoothing capacitor

Task 11. *Observe and draw the output voltage across the load resistor from the oscilloscope with a 4.7 μF capacitor fixed as C.*

Task 12. *Replace the 4.7 μF capacitor with C=220 μF capacitor and observe the output voltage waveform from the oscilloscope. Draw the waveform on the same sketch and notice the difference caused by increasing the value of the capacitor.*

Task 13. *Make a comment on your observations of the effect of the capacitive filter.*

3.4 Zener Regulator

The DC output voltage after the capacitor filter is a smooth DC signal. However, it would vary if the load resistor varies. Hence, the DC output voltage has to be stabilized. A simple Zener regulator can produce a reasonably stable DC output.

Connect the Zener diode and a series resistor as shown in figure, such that the output DC voltage is made stable or regulated.

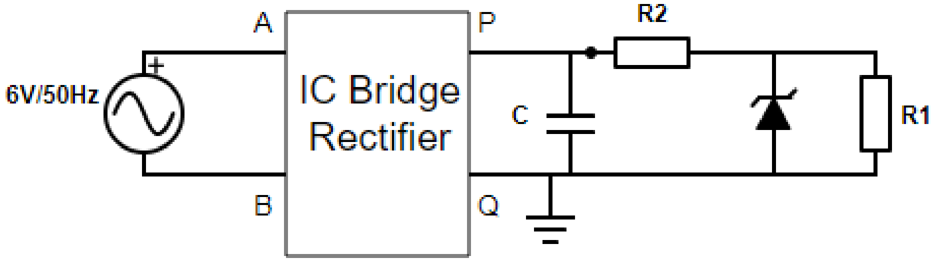


Figure 3.5: FWR with smoothing capacitor and Zener regulator. V_z=4.7 V, C = 220 μF, R_1=10 kΩ, R_2=100 Ω

Task 14. *Observe and record the DC output across the load resistor (V_{out}) from the oscilloscope.*

Task 15. *Change the load resistor R_L to the values mentioned in table 3.1 and record V_{out}.*

$R_L(\Omega)$	$V_{RL}(V)$
10,000	
560	
100	

Table 3.1: Effect of R_L on the regulated output voltage

Task 16. *Explain the behaviour of the DC output voltage as R_L varies.*

♣ The End ♣

Part IV

Telecommunication

Workshop 3: Digital Modulation Schemes

Objective: To identify digital modulation schemes using Matlab

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

1. Demonstrate an understanding of amplitude shift keying (ASK)
2. Demonstrate an understanding of frequency shift keying (FSK)
3. Demonstrate an understanding of phase shift keying (PSK)
4. Develop skills in using Matlab as a tool for communication systems study

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online

Components Required:

1. None

3.1 Digital Modulation

At this point we understand the basics of digital data transmission. In the previous practical we analyzed a baseband communication system, where one's and zero's are simply represented using two finite duration signals.

In this practical we will be analyzing digital modulation schemes. Digital modulation is the process of encoding a digital information signal into the amplitude, phase, or frequency of the transmitted signal. Hence, there are three major digital modulation techniques.

- Amplitude-Shift Keying (ASK)
- Frequency-Shift Keying (FSK)
- Phase-Shift Keying (PSK)

In simple terms, we will use sinusoids to encode bits or a group of bits. Consider the sinusoid $x(t) = A\sin(2\pi f t + \phi)$, where A is the amplitude, f is the frequency and ϕ is the phase. We can change one of these three properties to obtain distinct finite duration signals, which we use to encode bits or groups of bits.

For example we will consider ASK. Here, we first divide the bit stream into blocks of length n, where n is referred to as the order of the modulation scheme. Observe that each block may have 2^n possible strings. Hence, we will create 2^n sinusoids with distinct amplitudes each of which is mapped to a single string of length n. Then we encode each block with the respective sinusoid. This is known as 2^n-level ASK modulation. In case of FSK and PSK, we change the frequency and the phase of the sinusoids instead of the amplitude to generate the 2^n sinusoids.

3.2 Pre-Lab

We use constellation diagrams to represent digital modulation schemes. Read about constellation diagrams and understand how they are drawn (https://en.wikipedia.org/wiki/Constellation_diagram).

Task 1. *Draw the constellation diagrams of a 2-level ASK schemes where the amplitudes are (1) bit 0 → -1 and bit 1 → 1 (2) bit 0 → 0 and bit 1 → 1 side by side.*

Task 2. *Draw the constellation diagrams of a 2-level PSK schemes where the phases are (1) bit 0 → 0 and bit 1 → π (2) bit 0 → 0 and bit 1 → $\pi/2$ side by side.*

3.3 Amplitude-Shift Keying

We will first generate a 2-level ASK. Save the following function as ask.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=ask(bit_pattern,n)
Cf = 1.2E6; % Carrier frequency 1.2 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt = 1E−8;
fs = 1/delt;
samples_per_bit=250;
tmax = (samples_per_bit∗length(bit_pattern)−1)∗delt;
t= 0:delt:tmax; % Time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)∗samples_per_bit+sample)=bit_pattern(bit_no);
 end
end

% See what it looks like
figure;
subplot(2,1,1);plot(t,bits);
ylabel('Amplitude');
title('Info signal');
axis([0 tmax −2 2]);
% ASK modulation
ASK=[];
if n==2
 for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (1)∗sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (0)∗sin(Wc);
 end
 ASK=[ASK mod];
 end
 subplot(2,1,2); plot(t,ASK);
 ylabel('Amplitude');
 title('ASK Modulated Signal');
 axis([0 tmax −2 2]);
end
end
```

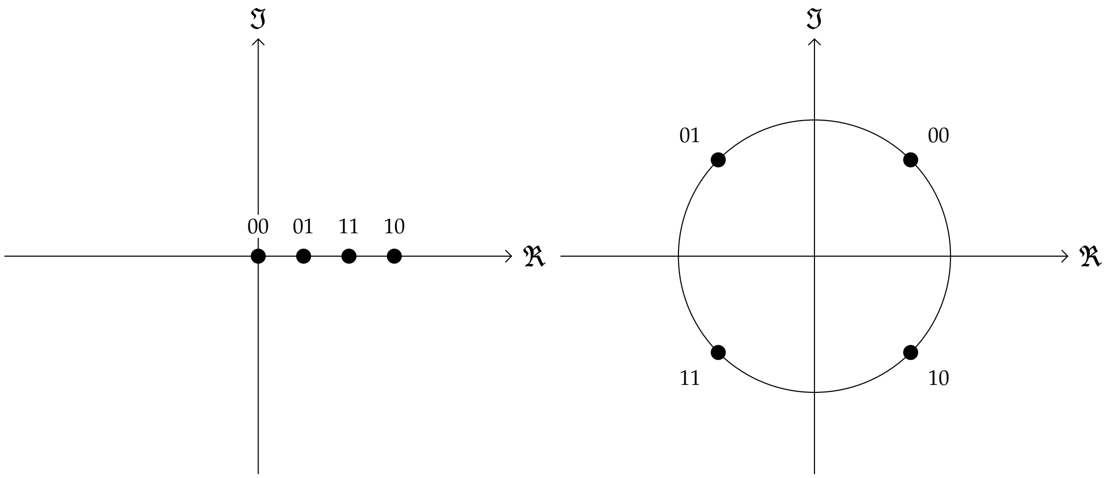
Generate the binary ASK modulation using the following.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
ask(bit_pattern,2);
```

Observe the binary ASK modulation and show the output to a Laboratory Instructor.

Task 3. *Fill the Table in the Task Sheet based on you observations.*

Task 4. *Extend the function to generate 4-level ASK modulation, as specified by the signal constellation in Figure 3.1a, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*



(a) 4-level ASK

(b) QPSK

Figure 3.1: Constellation diagrams

3.4 Frequency-Shift Keying

We will now move onto FSK. Generate the FSK modulation using the following script.

```
%fsk.m
close all;
clear all;
bit_pattern=[ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ];
Cf0 = 0.8E6; % Carrier frequency to encode binary 0, 0.8 MHz;
Cf1 = 2.4E6; % Carrier frequency to encode binary 1, 2.4 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt=1E−8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit∗length(bit_pattern)−1)∗delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)∗samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
```

```
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax −2 2]);
% FSK Modulation
FSK=[];
for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit =
(bit_no−1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit−1)*delt;
 Wc = Cf1*2*pi*t_bit;
 mod = (1)*sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit =
(bit_no−1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit−1)*delt;
 Wc = Cf0*2*pi*t_bit;
 mod = (1)*sin(Wc);
 end
 FSK=[FSK mod];
end
subplot(2,1,2); plot(t,FSK);
ylabel ('Amplitude');
title ('FSK Modulated Signal');
axis([0 tmax −2 2]);
```

Observe FSK modulation scheme and show the output to a Laboratory Instructor.

Task 5. *Fill the Table in the Task Sheet.*

3.5 Phase-Shift Keying

Finally, we will analyze phase-shift keying. Save the Matlab function given below as npsk.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=npsk(bit_pattern,n)
Cf = 4E5; %Carrier frequency 0.4 MHz;
% We will represent the signal as samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
delt=1E−8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit*length(bit_pattern)−1)*delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)*samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax −2 2]);
if n==2
 % BPSK Modulation
 BPSK=[];
 for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (1)∗sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (1)∗sin(Wc+pi);
 end
 BPSK=[BPSK mod];
 end
 subplot(2,1,2); plot(t,BPSK);
 ylabel ('Amplitude');
 title ('BPSK Modulated Signal');
 axis([0 tmax −2 2]);
end
end
```

Generate the binary PSK modulation (BPSK) using the following code.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
npsk(bit_pattern,2);
```

Observe BPSK modulation and show the output to a Laboratory Instructor.

Task 6. *Fill the Table in the Task Sheet.*

Task 7. *Extend the function to generate the Quaternary Phase Shift Keying (QPSK) modulation as specified by the signal constellation in Figure 3.1b, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*

Task 8. *Sketch constellation diagrams for 8-level and 16-level PSK, side by side.*

Task 9. *Comment on the advantages and disadvantages of higher order modulation schemes.*

♣ The End ♣