मौलाना आज़ाद
राष्ट्रीय प्रौद्योगिकी संस्थान भोपाल (म॰प्र॰) भारत
MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL (M. P.) INDIA

ASSIGNMENT

Data Communication Assignment

Sub Code: CSE-214

Vivek Kumar Ahirwar
191112419
CSE-3

Computer Science Department
MANIT, Bhopal

TABLE OF CONTENTS

ASSIGNMENT -1 **1**

SINE WAVE 1
SINE WAVE WITH NOISE 4
SQUARE WAVE USING MULTIPLE SINE WAVES 7

ASSGINEMENT -2 **12**

NRZ-L ENCODING AND NRZ-I ENCODING 12
NOISY SIGNAL - ADDING RANDOM NOISE 15
DETECTION OF BIT SEQUENCE BACK AND NUMBER OF BITS IN ERROR 21

191112419- Vivek Kumar Ahirwar

Data Communication Assignment 1

1 Generate a sine wave of specific frequency for specified time duration.

Sine waves represent periodic oscillations.

Sine waves have the shape of sine curve.

The X-axis of the sine curve represents the time.

The Y-axis of the sine curve represents the amplitude of the sine wave.

The amplitude of the sine wave at any point in Y is proportional to the sine of a variable.

The sine wave is given by the equation

$$A \sin(ft)$$

A - Amplitude t - Time f - Frequency

The sine curve goes through origin.

A cycle of sine wave is complete when the position of the sine wave starts from a position and comes to the same position after attaining its maximum and minimum amplitude during its course.

The time taken to complete one cycle is called the period of the sine wave.

The frequency of the sine wave is given by number of cycles per second.

'A' denotes amplitude of a sine wave.

The distance covered by a cycle measures the wavelength of the sine wave.

The wavelength of the sine wave is denoted by ⅄.

Examples of sine waves include the oscillations produced by the suspended weight on spring and the alternating current.

NumPy has the sin() function, which takes an array of values and provides the sine value for them.

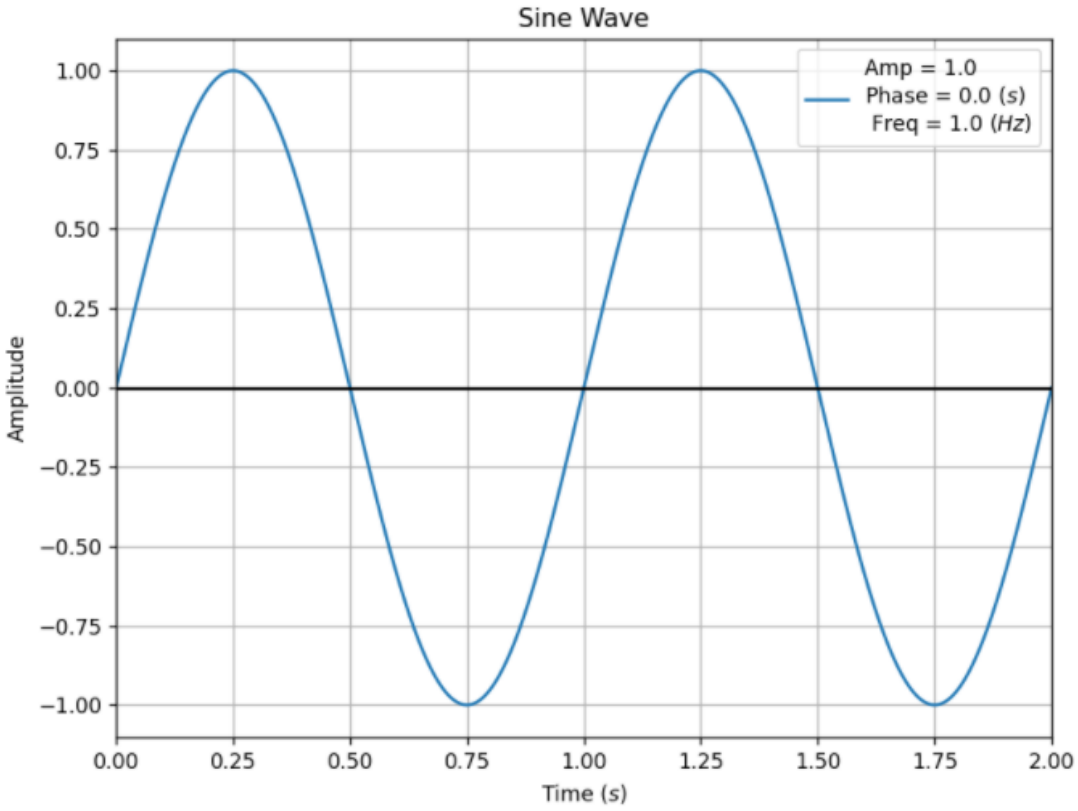
Using the numpy sin() function and the matplotlib plot()a sine wave can be drawn.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

[2]:
```
fig, ax = plt.subplots(1, figsize=(8, 6))


def draw_plot(time_duration,A, f, phase):

    ax.clear()



    t = np.linspace(0,time_duration,1000)
    ##### t= np.arange(0,10,0.01);

    y = A * np.sin(2 * np.pi * f * t + phase * 2 * np.pi)

    units = 'Amp = {}  \nPhase = {} $(s)$ \n Freq = {} $(Hz)$'

    plt.plot(t, y, label=units.format(A, phase, f))
    plt.legend(loc=1)
    ax.set_xlim(t[0], t[-1])

    plt.grid(True)
    plt.axhline(y=0, color='k')

    plt.xlabel('Time $(s)$')
    plt.ylabel('Amplitude')
    plt.title("Sine Wave")

    plt.show()



time_duration = wd.FloatSlider(min=0,max=10,value=2,description = "Time$(s)$")

A = wd.FloatSlider(min=0, max=10, value=1, description="Amp:")

f = wd.FloatSlider(min=0, max=10, value=1, description="Freq:")

phase = wd.FloatSlider(min=-5, max=5, value=0, description="Phase:")

wd.interactive(draw_plot,time_duration=time_duration, A=A, f=f, phase=phase)
```

```
<IPython.core.display.Javascript object>
```



Time(s) 2.00

Amp: 1.00

Freq: 1.00

Phase: 0.00

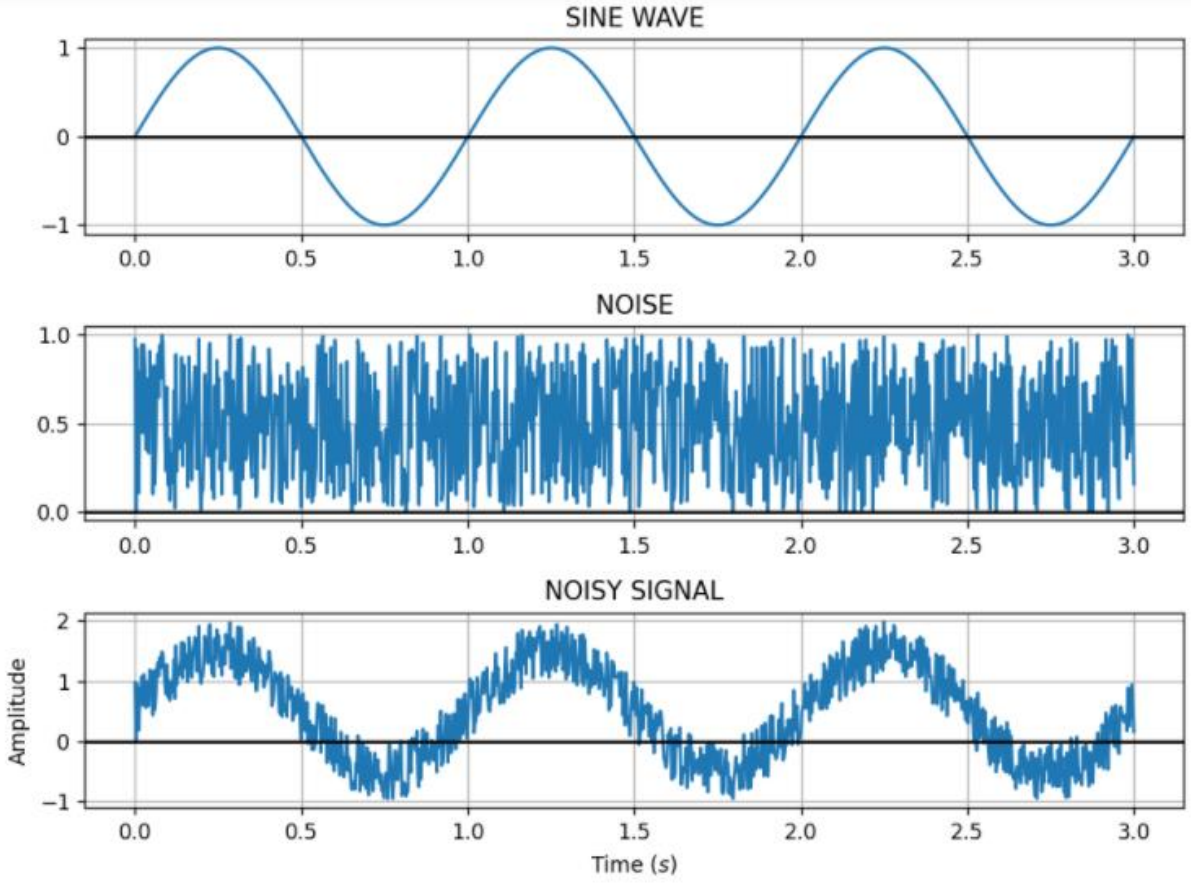
2 Generate a sine wave with noise.

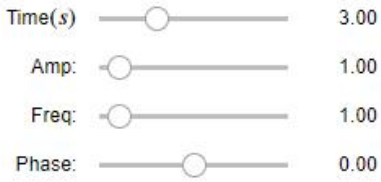
[8]:
```
import matplotlib.pyplot as plt
import numpy as np

import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

[9]:
```
fig, ax = plt.subplots(1, figsize=(8, 6))

def draw_plot(time_duration,A, f, phase):


    plt.clf()

    # Sampling rate 1000 hz / second
    t = np.linspace(0, time_duration , 1000 , endpoint = True)
    y = A * np.sin(2 * np.pi * f * t + phase * 2 * np.pi)
    noise = np.random.rand(len(y))
    corrupt = y + noise

#     units = 'Amp = {}  \nPhase = {} $(s)$ \n Freq = {} $(Hz)$'
#     plt.plot(t, y, label=units.format(A, phase, f))
#     plt.legend(loc=1)


    plt.subplot (3,1,1)
    plt.title (" SINE WAVE ")
    plt.plot (t,y)
    plt.grid (which= "Both")
    plt.axhline(y=0, color='k')

    plt.subplot (3,1,2)
    plt.title (" NOISE ")
    plt.plot(t,noise)
    plt.grid (which= "Both")
    plt.axhline(y=0, color='k')

    plt.subplot (3,1,3)
```

```
    plt.title (" NOISY SIGNAL ")
    plt.plot (t,corrupt)
    plt.grid (which= "Both")
    plt.axhline(y=0, color='k')


    plt.xlabel('Time $(s)$')
    plt.ylabel('Amplitude')

    plt.tight_layout()
    plt.show()



time_duration = wd.FloatSlider(min=0,max=10,value=2,description = "Time$(s)$")

A = wd.FloatSlider(min=0, max=10, value=1, description="Amp:")

f = wd.FloatSlider(min=0, max=10, value=1, description="Freq:")

phase = wd.FloatSlider(min=-5, max=5, value=0, description="Phase:")

wd.interactive(draw_plot,time_duration=time_duration, A=A, f=f, phase=phase)
```

```
<IPython.core.display.Javascript object>
```

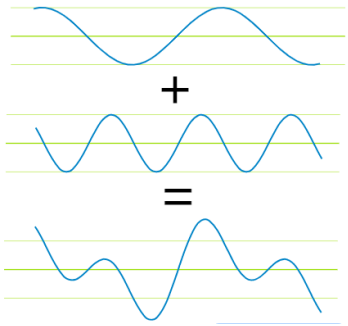

SINE WAVE
NOISE
NOISY SIGNAL
Amplitude
Time (s)


Time(s)
3.00
Amp:
1.00
Freq:
1.00
Phase:
0.00

3 Construct a square wave using multiple sine waves.

Sine and cosine waves can make other functions!

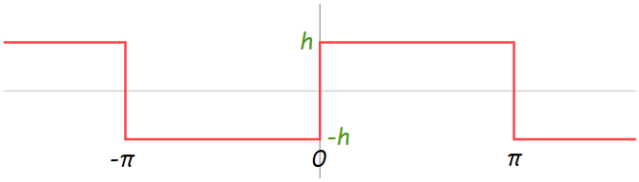
Here two different sine waves add together to make a new wave:



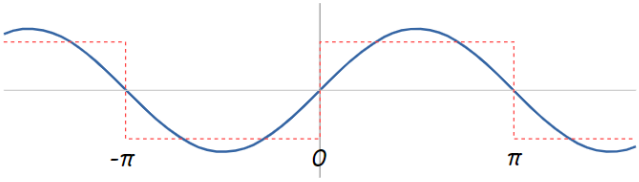
Square Wave

Can we use sine waves to make a **square wave**?

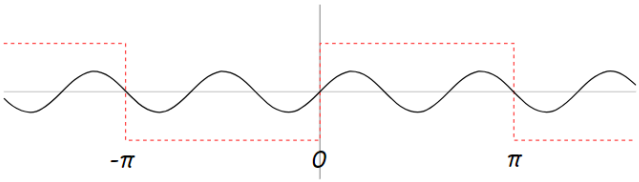
Our target is this square wave:



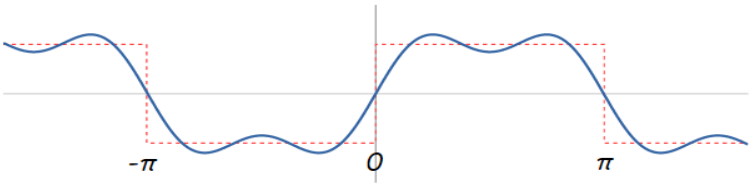
Start with **sin(x)**:



Then take **sin(3x)/3**:

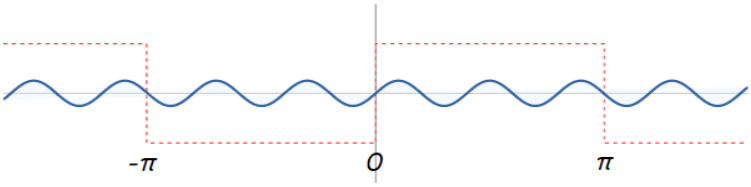


And add it to make **sin(x)+sin(3x)/3**:

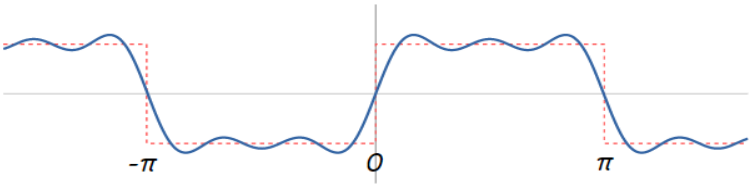


Can you see how it starts to look a little like a square wave?

Now take **sin(5x)/5**:

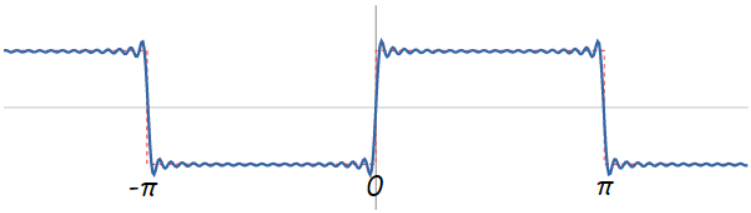


Add it also, to make **sin(x)+sin(3x)/3+sin(5x)/5**:

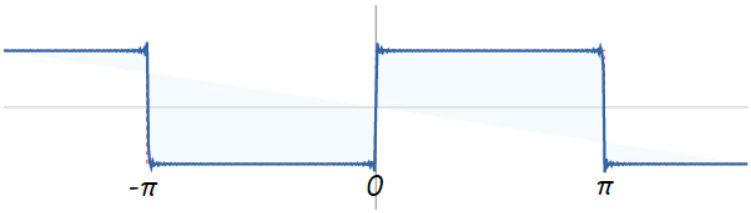


Getting better! Let's add a lot more sine waves.

Using 20 sine waves we get **sin(x)+sin(3x)/3+sin(5x)/5 + ... + sin(39x)/39**:



Using 100 sine waves we get **sin(x)+sin(3x)/3+sin(5x)/5 + ... + sin(199x)/199**:



And if we could add infinite sine waves in that pattern we would **have** a square wave!

So we can say that:

$$\text{a square wave} = \sin(x) + \sin(3x)/3 + \sin(5x)/5 + \ldots \text{ (infinitely)}$$

That is the idea of a Fourier series.

By adding infinite sine (and or cosine) waves we can make other functions, even if they are a bit weird.

Finding the Coefficients

How did we know to use sin(3x)/3, sin(5x)/5, etc?

There are formulas!

First let us write down a full series of sines and cosines, with a name for all coefficients:

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cos(nx\frac{\pi}{L}) + \sum_{n=1}^{\infty} b_n \sin(nx\frac{\pi}{L})$$

Where:

- f(x) is the function we want (such as a square wave)
- L is **half of the period** of the function
- a_0, a_n and b_n are **coefficients** that we need to calculate!

To find the coefficients a_0, a_n and b_n we use these formulas:

$$a_0 = \frac{1}{2L} \int_{-L}^{L} f(x)\, dx$$

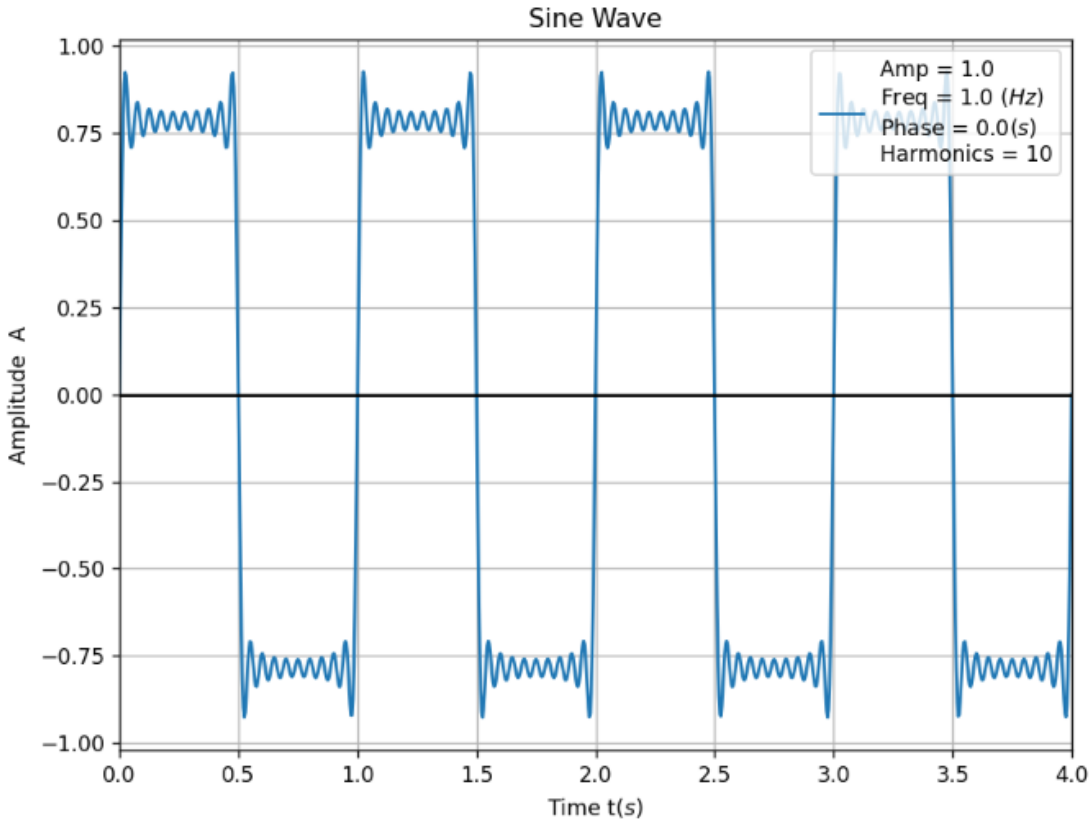
$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos(nx\frac{\pi}{L})\, dx$$

$$b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin(nx\frac{\pi}{L})\, dx$$

[5]:
```
import matplotlib.pyplot as plt
import numpy as np

import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

[7]:
```
fig, ax = plt.subplots(1, figsize=(8, 6))


def draw_plot(time_duration,A, f, phase,harmonics):

    ax.clear()

    t = np.linspace(0,time_duration,1000)
    ##### t= np.arange(0,10,0.01);

    y=0

    for ham in range(1,2*harmonics,2):
#     y = A * np.sin(2 * np.pi * f * t + phase * 2 * np.pi)
        y= y + (1/ham) * A * np.sin(2 * np.pi * ham * f * t + phase * 2 * np.pi)

    units = 'Amp = {} \nFreq = {} $(Hz)$\nPhase = {}$(s)$\nHarmonics = {}'

    ax.plot(t, y, label=units.format(A,f,phase,harmonics ))
    ax.legend(loc=1)
    ax.set_xlim(t[0], t[-1])

    plt.grid(True)
    plt.axhline(y=0, color='k')

    plt.xlabel('Time t$(s)$')
    plt.ylabel('Amplitude  A')
    plt.title("Sine Wave")

    plt.show()


time_duration = wd.FloatSlider(min=0,max=10,value=2,description = "Time$(s)$")

A = wd.FloatSlider(min=0, max=10, value=1, description="Amp:")

f = wd.FloatSlider(min=0, max=10, value=1, description="Freq:")

phase = wd.FloatSlider(min=-5, max=5, value=0, description="Phase:")

harmonics = wd.IntSlider(min=1, max=100, value=10, description="Harmonics:")

wd.interactive(draw_plot,time_duration=time_duration, A=A, f=f,␣
 ↪phase=phase,harmonics=harmonics)
```

```
<IPython.core.display.Javascript object>
```



Download plot

Time(s) 4.00

Amp: 1.00

Freq: 1.00

Phase: 0.00

Harmonics: 10

Data Communication Assignment 2

1 NRZ-L Encoding and NRZ-I Encoding

NRZ-L Encoding is a line encoding technique, specifically a serializer line code used to send information bitwise. Conventionally, 1 is represented by one physical level -1, while 0 is represented by another level 1.

In bipolar NRZL encoding, the signal essentially swings from one level to another.

NRZ-I Encoding is another serializer line encoding technique, used to send information bitwise.

The two-level NRZI signal distinguishes data bits by the presence or absence of a transition, meaning that a 1 is represented by a transition from the previous encoded bit, while 0 is represented by no transition.

NRZ-I encoding is used in USBs, but the opposite convention i.e. "change on 0" is used for encoding.

1.0.1 A-1 : Take input as bit sequence of n-bits, plot its NRZ-L and NRZ-I line coding.

1.0.2 Keep time axis resolution in milli-seconds.

[13]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

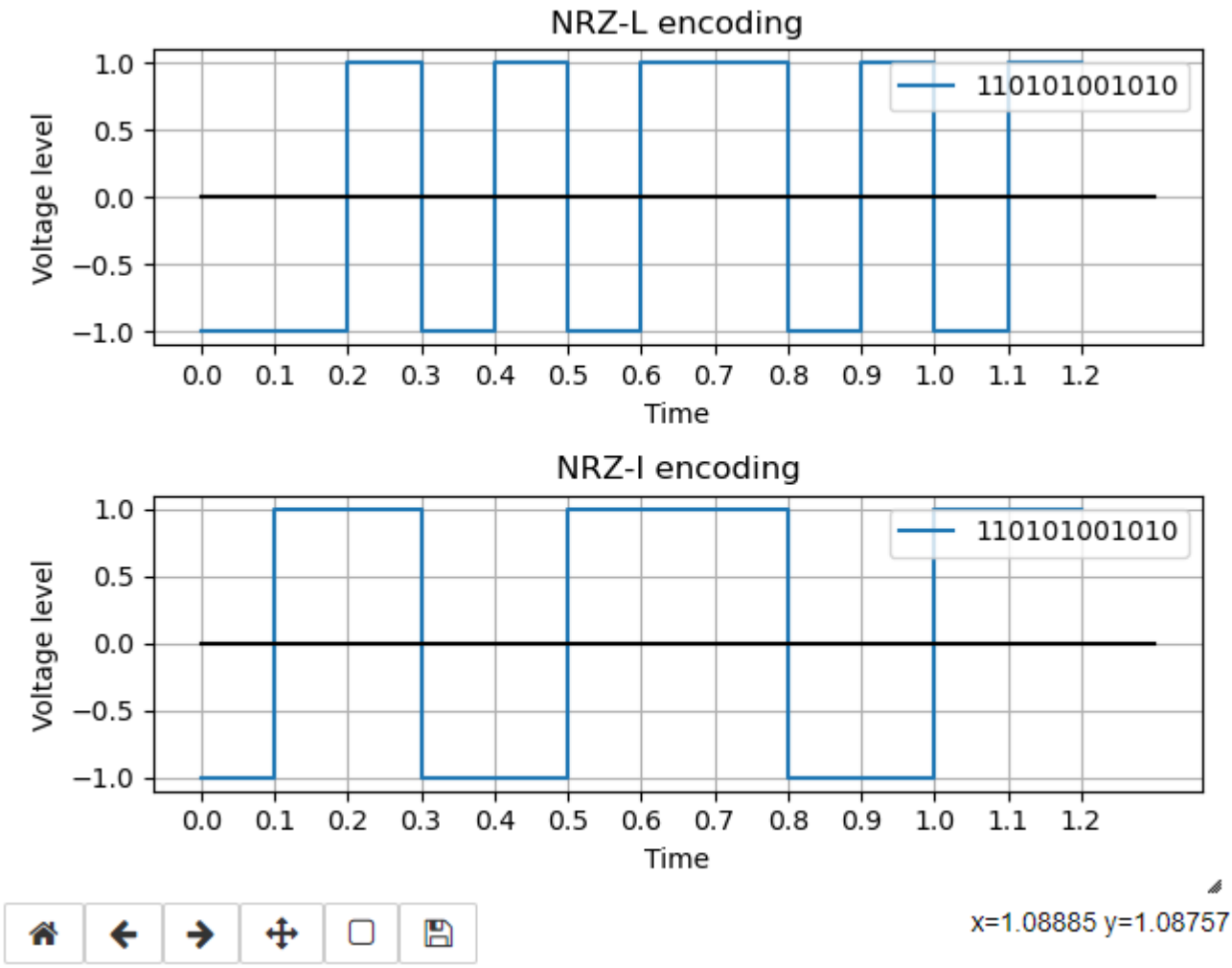
[14]:
```
print("Enter the bit sequence")
# bit_data=input("Enter the bit sequence")


def draw_plot(bit_data):

    plt.clf()

    bit_data = str(bit_data)

    # Creating two plots for two graphs
```

```
fig,(ax1,ax2)=plt.subplots(nrows=2,ncols=1)
x=[]
nrzl=[]
nrzi=[]
x_val=0


# Generating values for nrzl and nrzi
for it in range(len(bit_data)):

    if int(bit_data[it])==0:
        if len(nrzi) == 0:
            val=1
        else:
            val=nrzi[-1]
        for _ in range(8):
            x+=[round(random.uniform(x_val,x_val+0.1),4)]
            nrzl+=[1]
            nrzi+=[val]
        x+=[x_val,x_val+0.1]
        nrzl+=[1,1]
        nrzi+=[val,val]
        x_val+=0.1
    else:
        if len(nrzi)==0:
            val=-1
        else:
            val=-nrzi[-1]
        for _ in range(8):
            x += [round(random.uniform(x_val, x_val + 0.1),4)]
            nrzl += [-1]
            nrzi+=[val]
        x+=[x_val,x_val+0.1]
        nrzl+=[-1,-1]
        nrzi+=[val,val]
        x_val+=0.1
x. sort()


# list of values to be plotted on x-axis
val_to_be_pl=[0]
x_val=0.1
for _ in range(len(bit_data)):
     val_to_be_pl+=[x_val]
     x_val+=0.1
ax1.plot(x,nrzl,label=bit_data)
ax1.set_title("NRZ-L encoding")
```

```
    ax1.set_xlabel("Time")
    ax1.set_ylabel("Voltage level")
    ax1.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax1.set_xticks(val_to_be_pl)
    ax1.legend(loc = 1)
    ax1.grid(axis='both')

    ax2.plot(x,nrzi,label=bit_data)
    ax2.set_title("NRZ-I encoding")
    ax2.set_xlabel("Time")
    ax2.set_ylabel("Voltage level")
    ax2.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax2.set_xticks(val_to_be_pl)
    ax2.legend(loc = 1)
    ax2.grid(axis='both')


    plt.tight_layout()
    plt.show()


bit_data = wd.IntText(value=110101001010,  description='Bit Data:
 ↔',disabled=False)

# time_duration = wd.FloatSlider(min=0,max=10,value=2,description = "Time$(s)$")



wd.interactive(draw_plot,bit_data=bit_data)
```

Enter the bit sequence

Bit Data: 110101001010



1.0.3 A-2: Add random noise to above generated Digital Signal and plot the noisy signal

[15]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

[16]:
```
print("Enter the bit sequence")

def draw_plot(bit_data,vol_pos,vol_neg):

    plt.clf()

    bit_data = str(bit_data)



    # Creating Three Plots
    fig2, ax = plt.subplots(nrows=1,ncols=1,figsize=(7,2))

    fig1,(ax1,ax2)=plt.subplots(nrows=2,ncols=1)

    fig3,(ax3,ax4)=plt.subplots(nrows=2,ncols=1)

    x=[]
    nrzl=[]
    nrzi=[]
    x_val=0

    # Generating values for nrzl and nrzi
    for it in range(len(bit_data)):
        if int(bit_data[it])==0:
            if len(nrzi) == 0:
                val=1
            else:
                val=nrzi[-1]
            for _ in range(18):
                x+=[round(random.uniform(x_val,x_val+0.1),4)]
                nrzl+=[1]
                nrzi+=[val]
            x+=[x_val,x_val+0.1]
            nrzl+=[1,1]
            nrzi+=[val,val]
            x_val+=0.1
        else:
            if len(nrzi)==0:
                val=-1
            else:
                val=-nrzi[-1]
            for _ in range(18):
                x += [round(random.uniform(x_val, x_val + 0.1),4)]
                nrzl += [-1]
                nrzi+=[val]
            x+=[x_val,x_val+0.1]
```

```
            nrzl+=[-1,-1]
            nrzi+=[val,val]
            x_val+=0.1
x. sort()
nrzl_with_noise=[]
nrzi_with_noise=[]
rand_noise=[]


#Generating random noise signal
for j in range(len(x)):
    rand_noise+=[round(random.uniform(vol_neg,vol_pos),4)]
    nrzl_with_noise+=[rand_noise[j]+nrzl[j]]
    nrzi_with_noise+=[rand_noise[j]+nrzi[j]]

# list of values to be plotted on x-axis
val_to_be_pl=[0]
x_val=0.1
for _ in range(len(bit_data)):
     val_to_be_pl+=[x_val]
     x_val+=0.1



# Noise Plot
ax.plot(x,rand_noise)
ax.set_title("Random noise")
ax.set_xlabel("Time")
ax.set_ylabel("Voltage level")
ax.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
ax.set_xticks(val_to_be_pl)
ax.grid(axis='both')

#fig1 will consists of ax1,ax2
#ax1 will plot nrzl+noise
ax1.plot(x,nrzl,label=bit_data)
ax1.plot(x,rand_noise,color='lightcoral', linewidth=1, linestyle='dotted')
ax1.set_title("NRZ-L encoding + random noise")
ax1.set_xlabel("Time")
ax1.set_ylabel("Voltage level")
ax1.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
ax1.set_xticks(val_to_be_pl)
ax1.legend()
ax1.grid(axis='both')


#ax2 will plot the resultant nrzl due to noise i.e noisy signal
```

```
    ax2.plot(x,nrzl_with_noise)
    ax2.plot(x,nrzl,color='lightcoral', linewidth=1, linestyle='-')
    ax2.set_title("Noisy signal")
    ax2.set_xlabel("Time")
    ax2.set_ylabel("Voltage level")
    ax2.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax2.set_xticks(val_to_be_pl)
    ax2.grid(axis='both')



    #fig3 will consists of ax3,ax4
    #ax3 will plot nrzi+noise
    #ax4 will plot the resultant nrzi due to noise i.e noisy signal
    ax3.plot(x,nrzi,label=bit_data)
    ax3.plot(x,rand_noise,color='lightcoral', linewidth=1, linestyle='dotted')
    ax3.set_title("NRZ-I encoding + noise")
    ax3.set_xlabel("Time")
    ax3.set_ylabel("Voltage level")
    ax3.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax3.set_xticks(val_to_be_pl)
    ax3.legend(loc = 1)
    ax3.grid(axis='both')


    ax4.plot(x,nrzi_with_noise)
    ax4.plot(x,nrzi,color='lightcoral', linewidth=1, linestyle='-')
    ax4.set_title("Noisy signal")
    ax4.set_xlabel("Time")
    ax4.set_ylabel("Voltage level")
    ax4.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax4.set_xticks(val_to_be_pl)
    ax4.grid(axis='both')


    plt.tight_layout()
    plt.show()


bit_data = wd.IntText(value=110101001010,  description='Bit Data:
 ↪',disabled=False)


vol_pos = wd.FloatSlider(min=0, max=5, value=0.5, description="Maximum Positive␣
 ↪voltage of noise:")
```

```
vol_neg = wd.FloatSlider(min=-5, max=0, value=-0.5, description="Maximum␣
↪Negative voltage of noise:")

bit_d = str(bit_data)

wd.interactive(draw_plot,bit_data=bit_data,vol_pos=vol_pos,vol_neg=vol_neg)
```

Enter the bit sequence

Bit Data: 110101001010

Maximum ... 0.50

Maximum ... -0.50

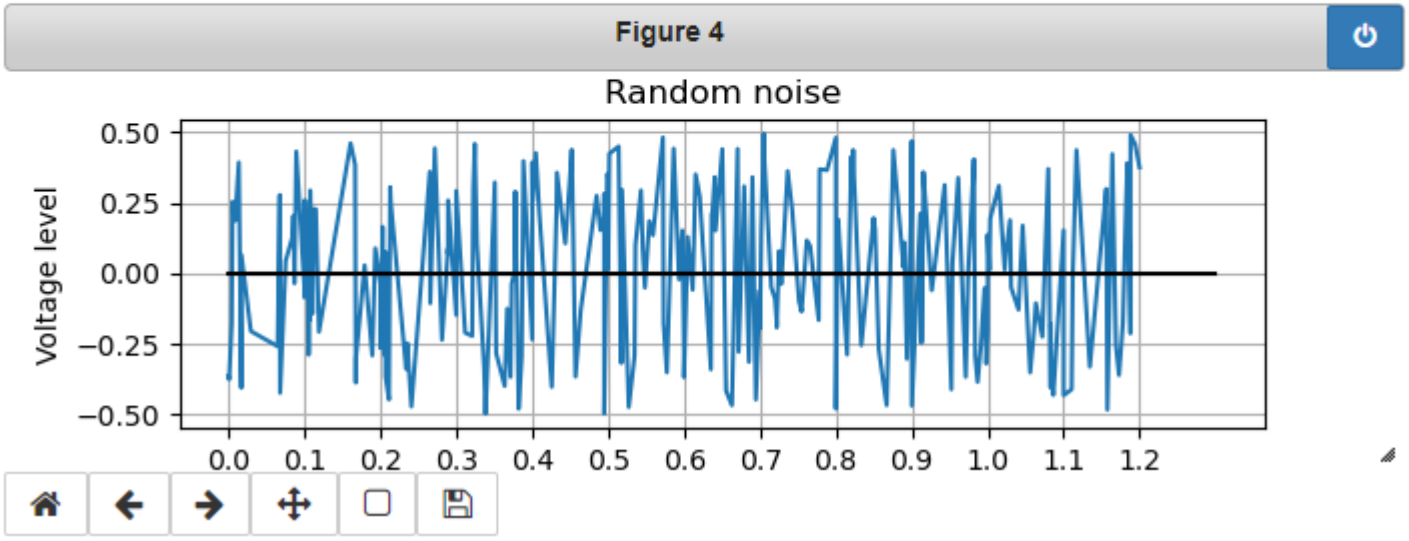


Figure 5

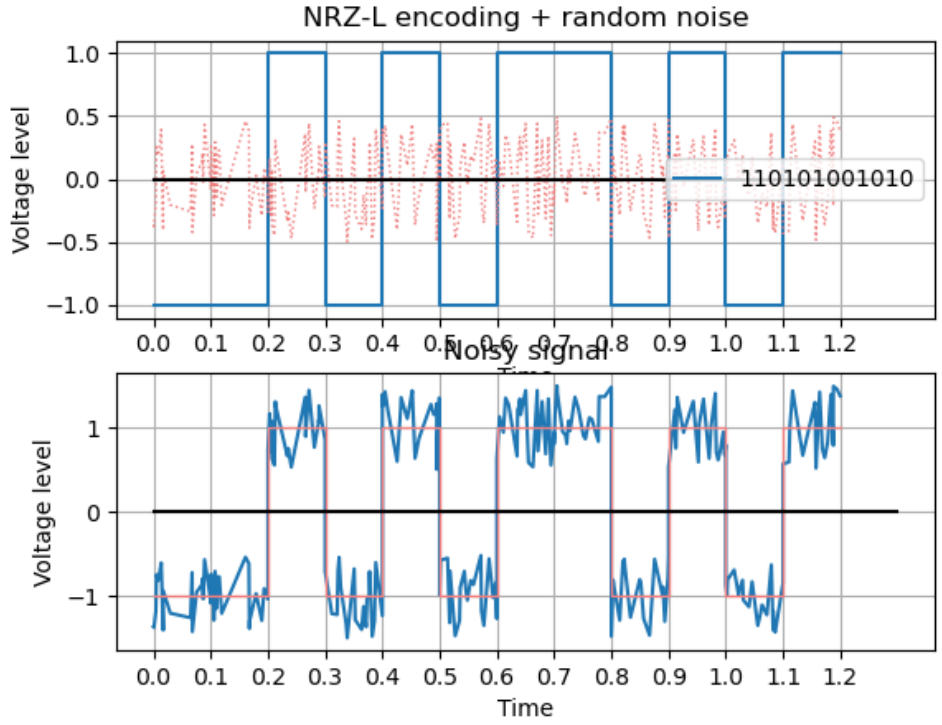
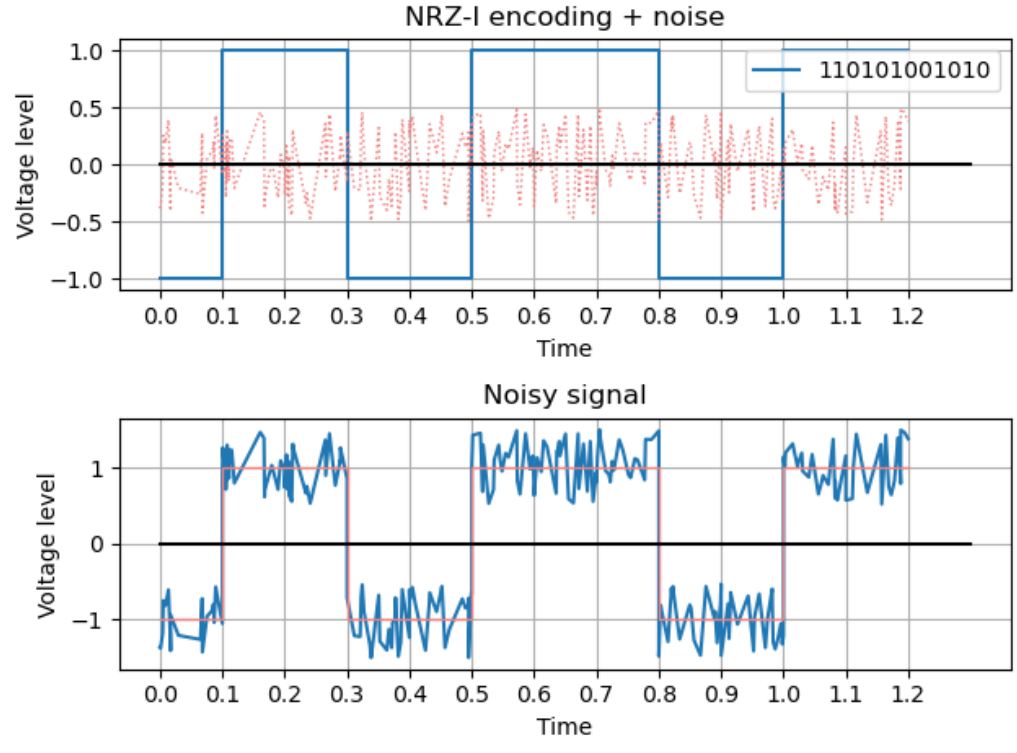


Figure 6



x=0.505845 y=0.592789

1.0.4 A-3: Decide threshold for detection of bit sequence back and calculate number of bits in error

[17]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
import math
import ipywidgets as wd
from IPython.display import display

%matplotlib nbagg
```

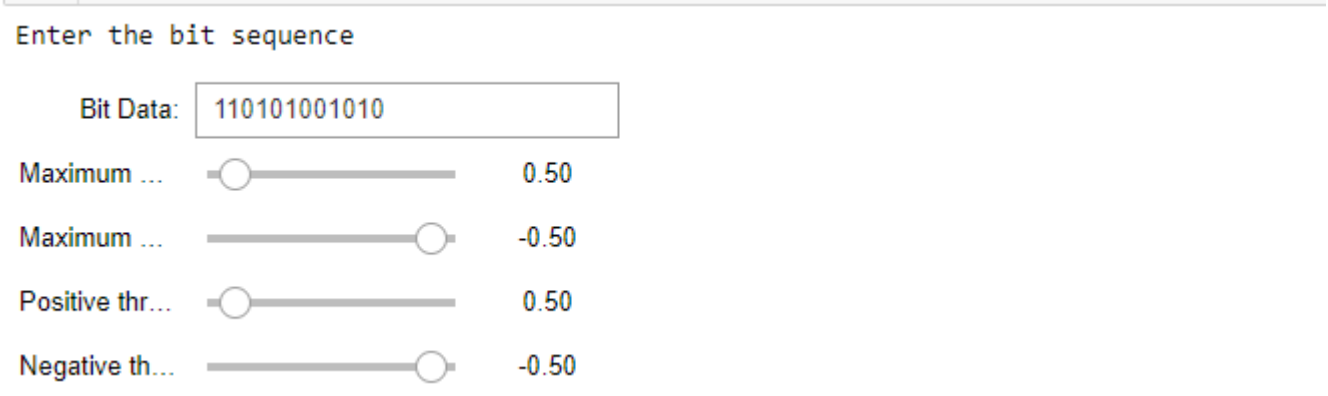
[18]:
```
print("Enter the bit sequence")

def draw_plot(bit_data,vol_pos,vol_neg,threshold_pos,threshold_neg):

    plt.clf()

    bit_data = str(bit_data)



    # Creating Two Plots

    fig1,(ax1,ax3)=plt.subplots(nrows=2,ncols=1)
    fig2,(ax2,ax4)=plt.subplots(nrows=2,ncols=1)

    x=[]
    nrzl=[]
    nrzi=[]
    x_val=0

    # Generating values for nrzl and nrzi
    for it in range(len(bit_data)):
        if int(bit_data[it])==0:
            if len(nrzi) == 0:
```

```
            val=1
        else:
            val=nrzi[-1]
        for _ in range(18):
            x+=[round(random.uniform(x_val,x_val+0.1),4)]
            nrzl+=[1]
            nrzi+=[val]
        x+=[x_val,x_val+0.1]
        nrzl+=[1,1]
        nrzi+=[val,val]
        x_val+=0.1
    else:
        if len(nrzi)==0:
            val=-1
        else:
            val=-nrzi[-1]
        for _ in range(18):
            x += [round(random.uniform(x_val, x_val + 0.1),4)]
            nrzl += [-1]
            nrzi+=[val]
        x+=[x_val,x_val+0.1]
        nrzl+=[-1,-1]
        nrzi+=[val,val]
        x_val+=0.1
x. sort()
nrzl_with_noise=[]
nrzi_with_noise=[]
rand_noise=[]


#Generating random noise signal
for j in range(len(x)):
    rand_noise+=[round(random.uniform(vol_neg,vol_pos),4)]
    nrzl_with_noise+=[rand_noise[j]+nrzl[j]]
    nrzi_with_noise+=[rand_noise[j]+nrzi[j]]


#bit sequence received
bit_seq_in_nrzi=""
bit_seq_in_nrzl=""
ambg_in_nrzl=0

#it is just any number, to calculate nrzi
prev_mean=100
i=0
while i<len(nrzl):
    mean1=(sum(nrzl_with_noise[i:i+20]))/10
```

```
        mean2=(sum(nrzi_with_noise[i:i+20]))/10
        if mean1>=threshold_pos:
            bit_seq_in_nrzl+="0"
        elif mean1<=threshold_neg:
            bit_seq_in_nrzl += "1"
        else:#if it does not fall in any limit then
             #we will consider the bit as 1 and we we
             #can have count of such bits
            bit_seq_in_nrzl+="1"
            ambg_in_nrzl+=1
        if prev_mean==100:
            if mean2>=threshold_pos:
                bit_seq_in_nrzi+="0"
            else:
                bit_seq_in_nrzi+="1"
        else:
            if abs(prev_mean+mean2)>abs(mean2) and␣
↪abs(prev_mean+mean2)>abs(mean1):#if they have same sign
                bit_seq_in_nrzi += "0"
            else:#therefore they have opp sign
                bit_seq_in_nrzi += "1"
        prev_mean = mean2
        i+=20

    bit_err_in_nrzi=0
    bit_err_in_nrzl=0
    for j in range(len(bit_data)):
        if int(bit_data[j])!=int(bit_seq_in_nrzi[j]):
            bit_err_in_nrzi+=1
        if int(bit_data[j])!=int(bit_seq_in_nrzl[j]):
            bit_err_in_nrzl+=1
    print("No. of bits which we were in ambiguous to indentify in␣
↪NRZ-L=",ambg_in_nrzl)
    print("No.of bits error in NRZ-L = ",bit_err_in_nrzl)
    print("No.of bits error in NRZ-I = ",bit_err_in_nrzi)

    # list of values to be plotted on x-axis
    val_to_be_pl=[0]
    x_val=0.1
    for _ in range(len(bit_data)):
         val_to_be_pl+=[x_val]
         x_val+=0.1


    #fig1 will consists of ax2,ax4
    #ax1 will plot nrzl+noise
    #ax3 will plot noisy signal
```

```
    ax1.plot(x,nrzl,label=bit_data)
#     ax1.plot(x,rand_noise)
    ax1.set_title("NRZ-L encoding + random noise")
    ax1.set_xlabel("Time")
    ax1.set_ylabel("Voltage level")
    ax1.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax1.set_xticks(val_to_be_pl)
    ax1.legend(loc=1)
#     ax1.tight_layout()
    ax1.grid(axis='both')


    ax3.plot(x,nrzl_with_noise,label="bit rec={}".format(bit_seq_in_nrzl))
    ax3.plot(x,nrzl,color='lightcoral', linewidth=1, linestyle='-')
    ax3.set_title("Noisy signal")
    ax3.set_xlabel("Time")
    ax3.set_ylabel("Voltage level")
    ax3.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax3.set_xticks(val_to_be_pl)
    ax3.legend(loc=1)
#     ax3.tight_layout()
    ax3.grid(axis='both')


    #fig2 will consists of ax2,ax4
    #ax2 will plot nrzi+noise
    #ax4 will plot noisy signal
    ax2.plot(x,nrzi,label=bit_data)
#     ax2.plot(x,rand_noise)
    ax2.set_title("NRZ-I encoding + noise")
    ax2.set_xlabel("Time")
    ax2.set_ylabel("Voltage level")
    ax2.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax2.set_xticks(val_to_be_pl)
    ax2.legend(loc=1)
#     ax2.tight_layout()
    ax2.grid(axis='both')


    ax4.plot(x,nrzi_with_noise,label="bit rec={}".format(bit_seq_in_nrzi))
    ax4.plot(x,nrzi,color='lightcoral', linewidth=1, linestyle='-')
    ax4.set_title("Noisy signal")
    ax4.set_xlabel("Time")
    ax4.set_ylabel("Voltage level")
    ax4.plot((0,(len(bit_data)+1)*0.1),(0,0),color='k')
    ax4.set_xticks(val_to_be_pl)
    ax4.legend(loc=1)
```

```
    ax4.grid(axis='both')

    fig1.tight_layout()
    fig2.tight_layout()
    plt.tight_layout()
    plt.show()


bit_data = wd.IntText(value=110101001010,  description='Bit Data:
 ↪',disabled=False)

vol_pos = wd.FloatSlider(min=0, max=5, value=0.5,description="Maximum Positive␣
 ↪voltage of noise:")
vol_neg = wd.FloatSlider(min=-5, max=0, value=-0.5,description="Maximum␣
 ↪Negative voltage of noise:")

threshold_pos = wd.FloatSlider(min=0, max=5, value=0.5,description="Positive␣
 ↪threshold amplitude")
threshold_neg = wd.FloatSlider(min=-5, max=0, value=-0.5,description="Negative␣
 ↪threshold amplitude")

wd.interactive(draw_plot, bit_data=bit_data, vol_pos=vol_pos, vol_neg=vol_neg,
                threshold_pos=threshold_pos,threshold_neg=threshold_neg)
```



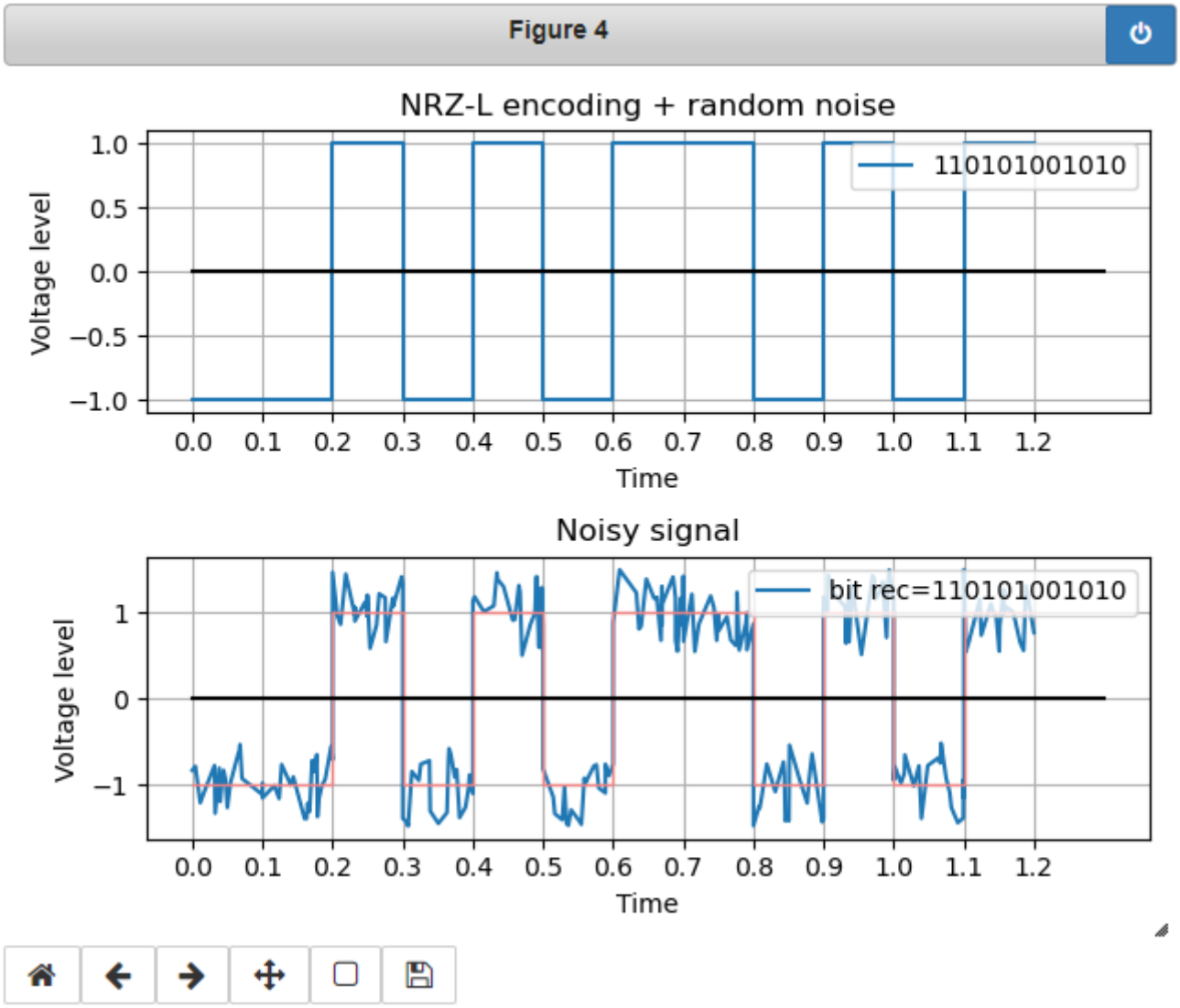
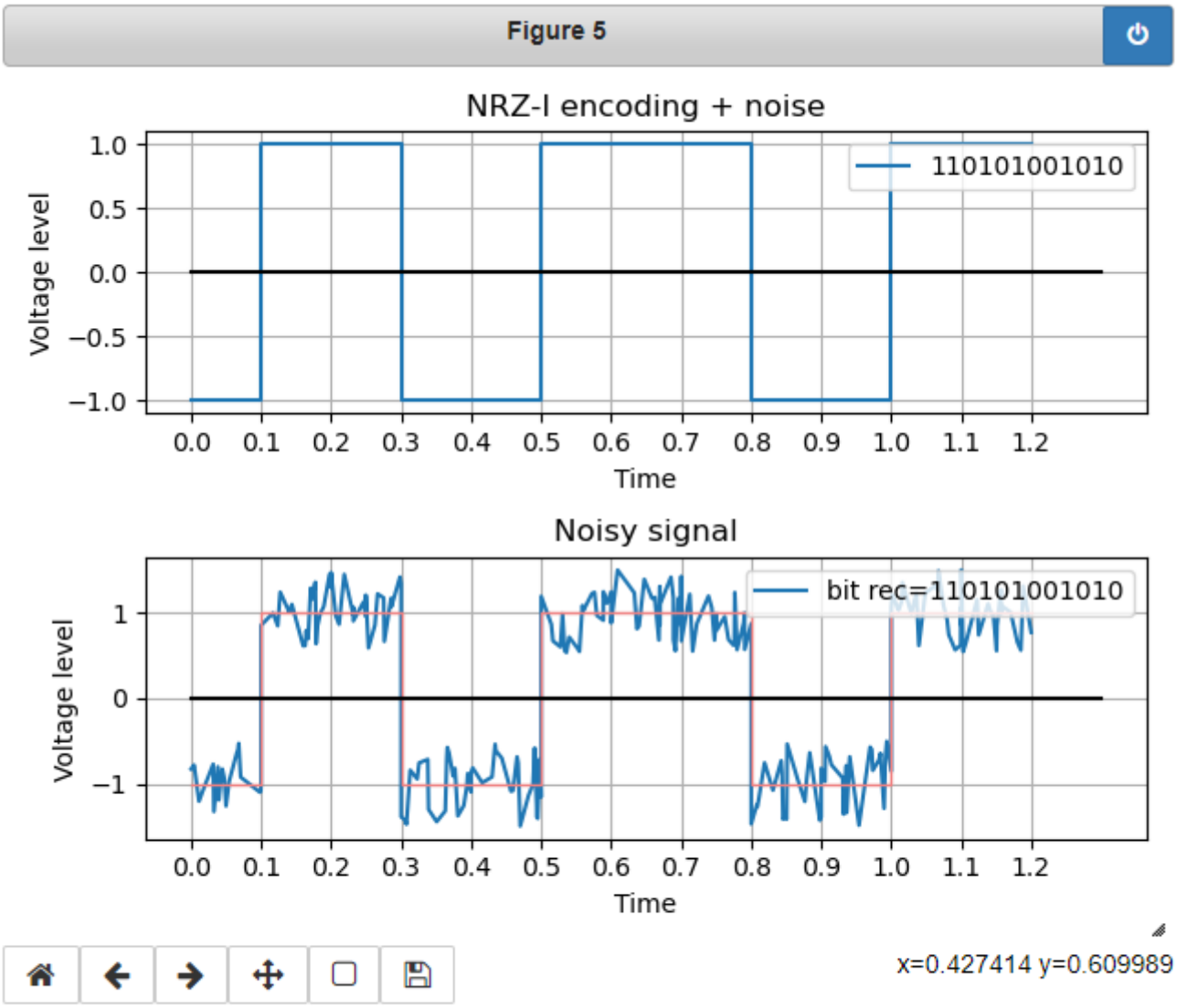


Figure 4
NRZ-L encoding + random noise
110101001010
Voltage level
1.0
0.5
0.0
−0.5
−1.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2
Time
Noisy signal
bit rec=110101001010
Voltage level
1
0
−1
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2
Time




```
No. of bits which we were in ambiguous to indentify in NRZ-L= 0
No.of bits error in NRZ-L =  0
No.of bits error in NRZ-I =  0
```