Dept. of Electronics and Electrical Communications Engineering Indian Institute of Technology, Kharagpur

DIGITAL COMMUNICATION LABORATORY [EC39002]

EXPERIMENT – 6

Huffman Encoding and Decoding

Samineni Rahul(21EC30045)
Group 23

Objective: To implement Channel Encoding and Decoding on a random sequence using BPSK channel modulation, evaluate the Error bitrate for the given BPSK carrier wave.

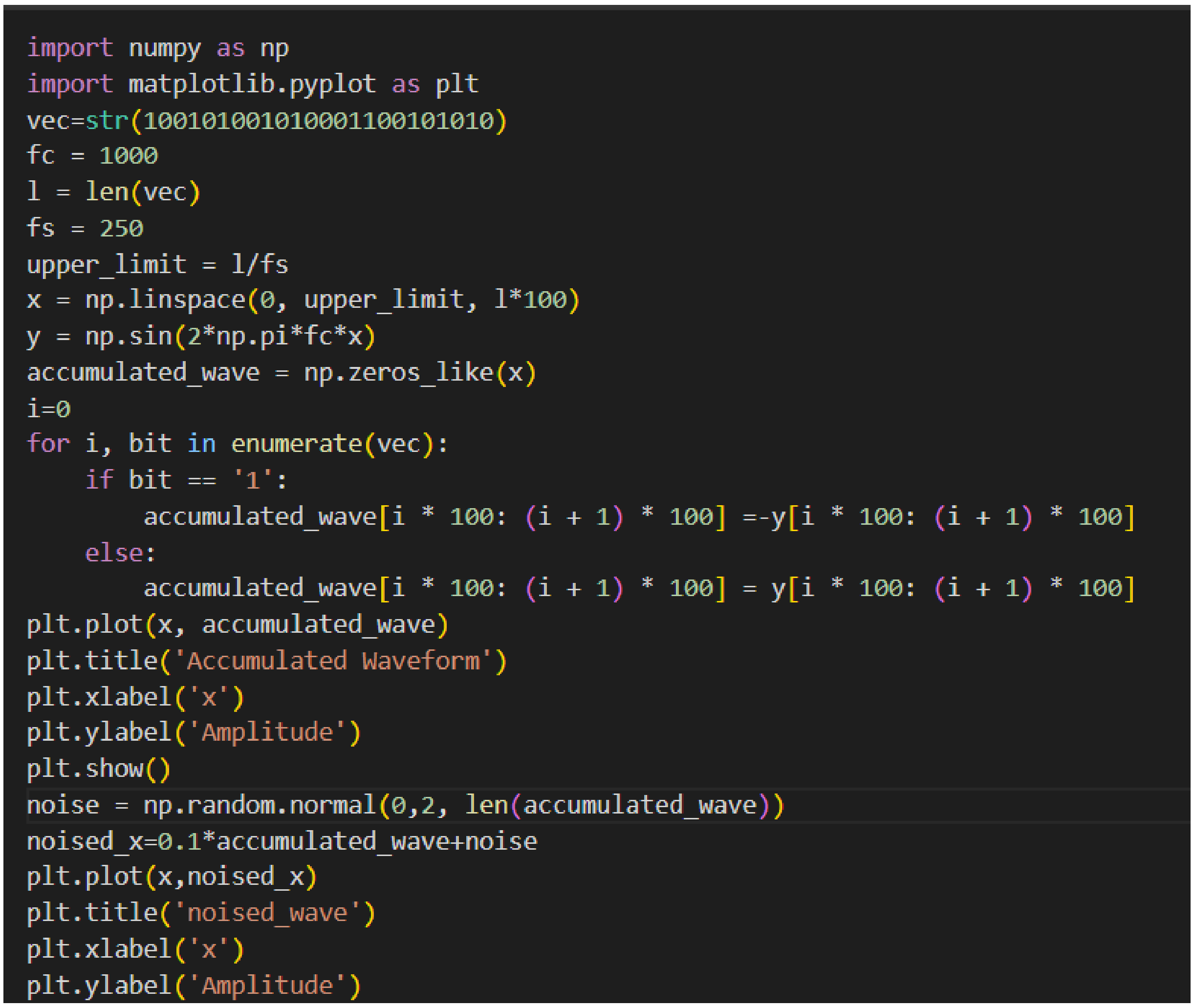
Theory:

Binary Phase Shift Keying (BPSK) modulation is a digital modulation scheme used in telecommunications and digital communication systems. It modulates binary data onto a sine wave carrier signal by shifting the phase of the carrier to represent the binary symbols (0 and 1). In BPSK, the phase of the carrier is shifted by 180 degrees (π radians) for each binary symbol.

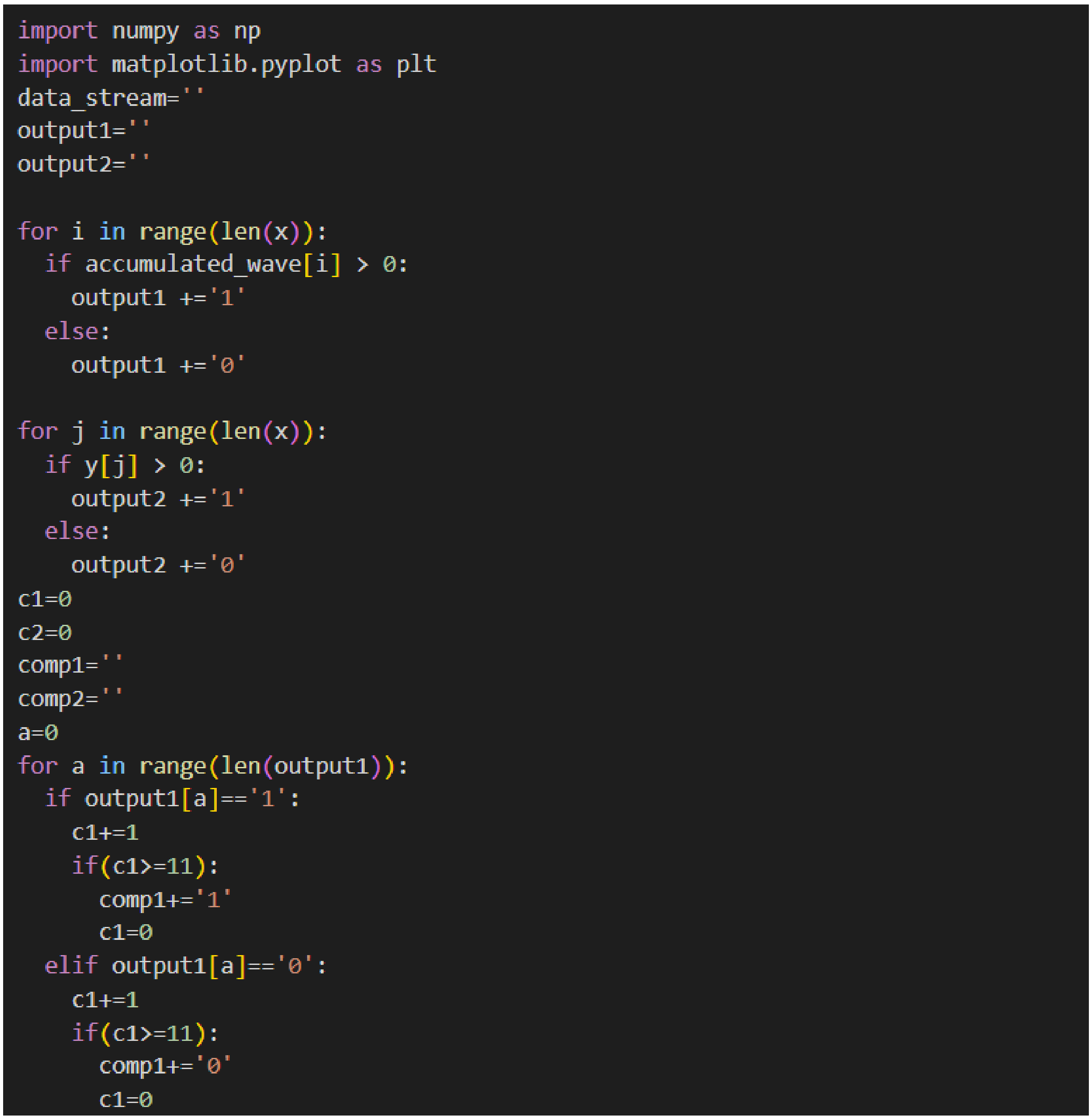
Working of BPSK modulation:

Binary Data Representation: Binary data is represented as a sequence of 0s and 1s.
Carrier Signal: A carrier signal, typically a sine wave, is used to transmit the binary data. The carrier signal is modulated to represent the binary symbols.
Phase Shift: Each binary symbol (0 or 1) is mapped to a specific phase shift of the carrier signal. For BPSK, the phase shift for the binary symbol 0 is 0 degrees, and the phase shift for the binary symbol 1 is 180 degrees (π radians).
Modulation: To modulate the binary data onto the carrier signal, the carrier's phase is shifted according to the binary symbols. For example, if the binary symbol is 0, the phase of the carrier remains unchanged. If the binary symbol is 1, the phase of the carrier is inverted.
Transmission: The modulated signal, which consists of the carrier signal with phase shifts representing the binary symbols, is transmitted over the communication channel.
Demodulation: At the receiver end, the received signal is demodulated to extract the original binary data. This is achieved by detecting the phase shifts of the carrier signal and mapping them back to the binary symbols.

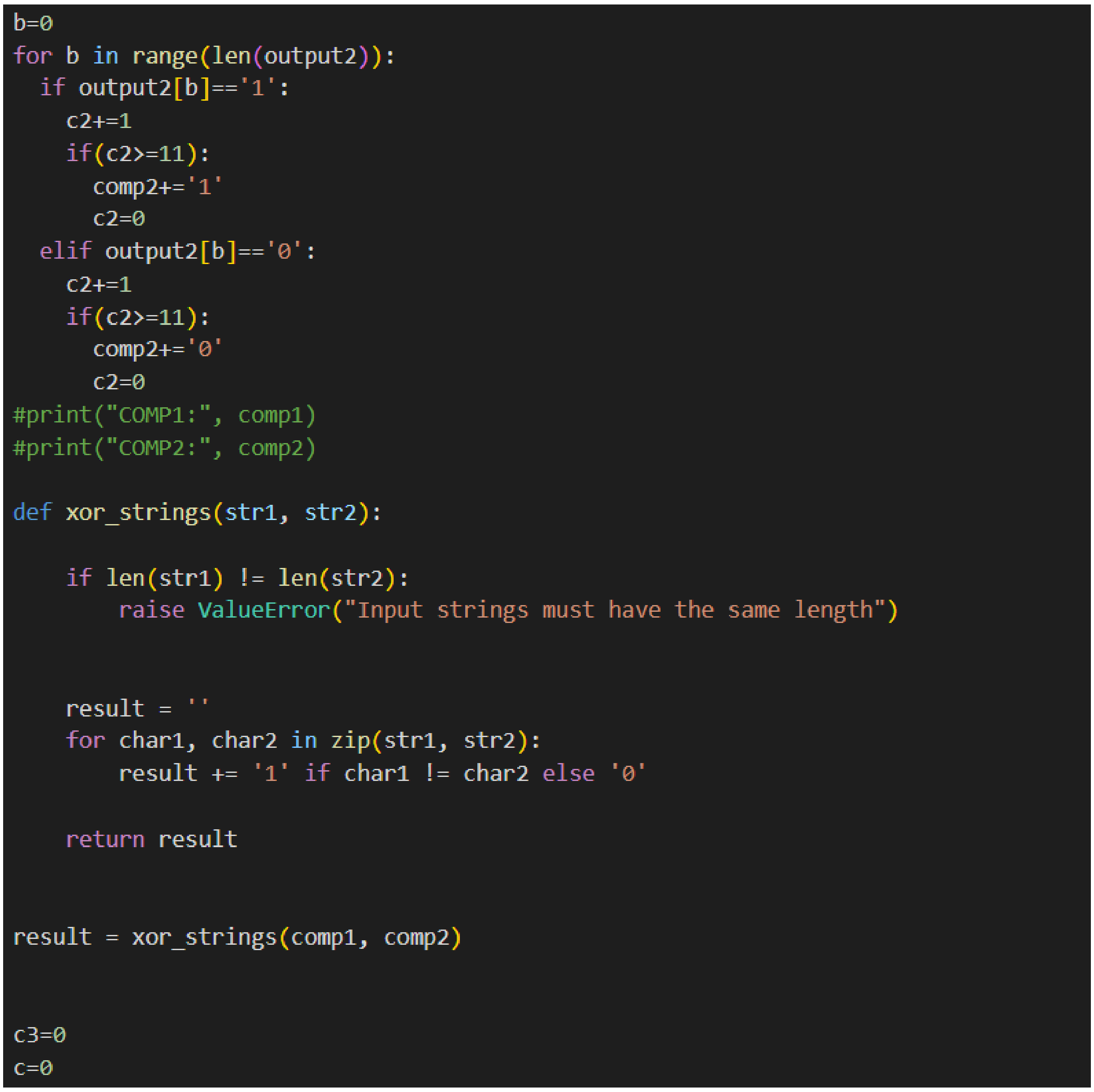
Code



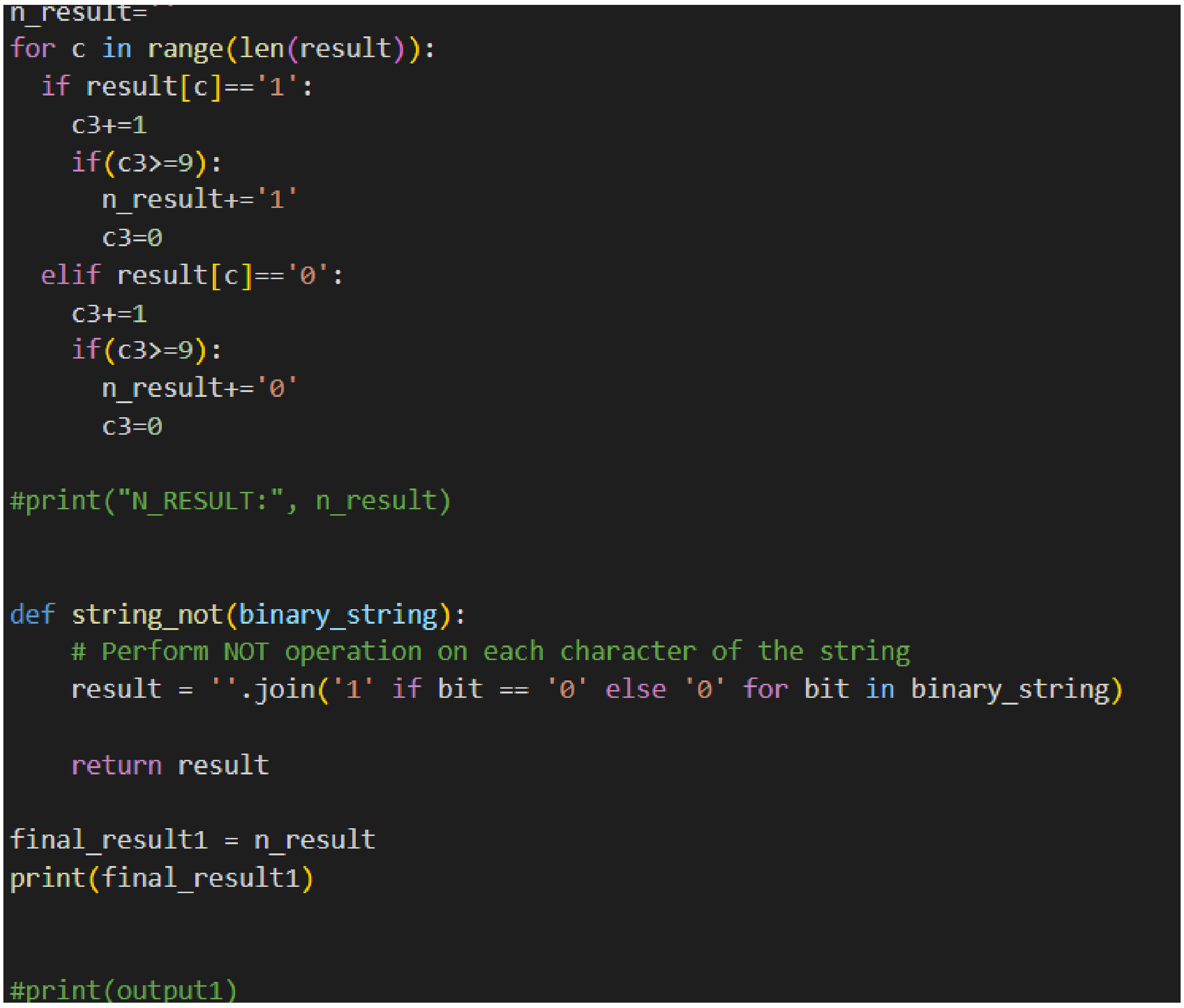
```python
import numpy as np
import matplotlib.pyplot as plt
vec=str(100101001010001100101010)
fc = 1000
l = len(vec)
fs = 250
upper_limit = l/fs
x = np.linspace(0, upper_limit, l*100)
y = np.sin(2*np.pi*fc*x)
accumulated_wave = np.zeros_like(x)
i=0
for i, bit in enumerate(vec):
    if bit == '1':
        accumulated_wave[i * 100: (i + 1) * 100] =-y[i * 100: (i + 1) * 100]
    else:
        accumulated_wave[i * 100: (i + 1) * 100] = y[i * 100: (i + 1) * 100]
plt.plot(x, accumulated_wave)
plt.title('Accumulated Waveform')
plt.xlabel('x')
plt.ylabel('Amplitude')
plt.show()
noise = np.random.normal(0,2, len(accumulated_wave))
noised_x=0.1*accumulated_wave+noise
plt.plot(x,noised_x)
plt.title('noised_wave')
plt.xlabel('x')
plt.ylabel('Amplitude')
```



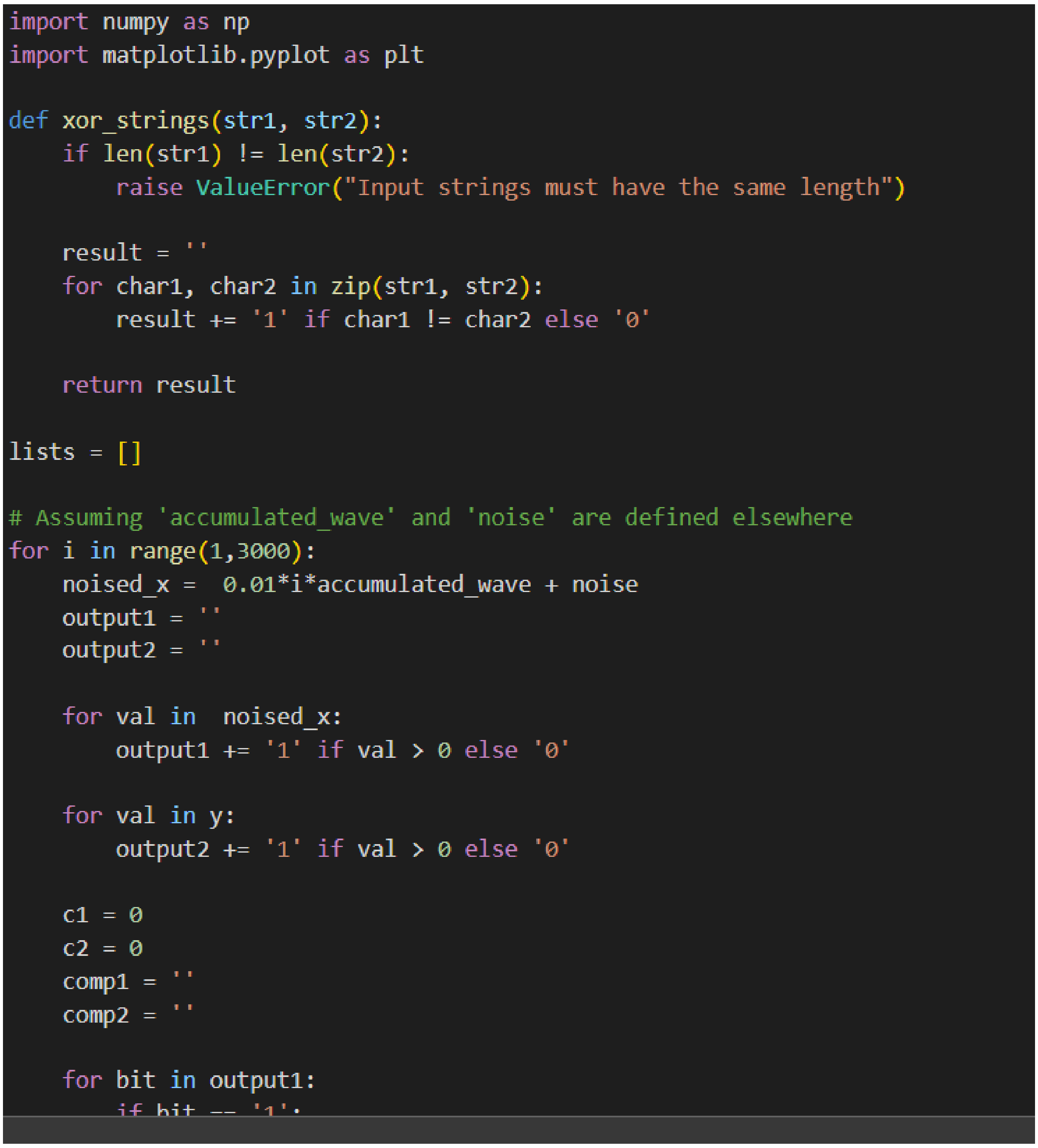
```python
import numpy as np
import matplotlib.pyplot as plt
data_stream=''
output1=''
output2=''

for i in range(len(x)):
  if accumulated_wave[i] > 0:
    output1 +='1'
  else:
    output1 +='0'

for j in range(len(x)):
  if y[j] > 0:
    output2 +='1'
  else:
    output2 +='0'
c1=0
c2=0
comp1=''
comp2=''
a=0
for a in range(len(output1)):
  if output1[a]=='1':
    c1+=1
    if(c1>=11):
      comp1+='1'
      c1=0
  elif output1[a]=='0':
    c1+=1
    if(c1>=11):
      comp1+='0'
      c1=0
```



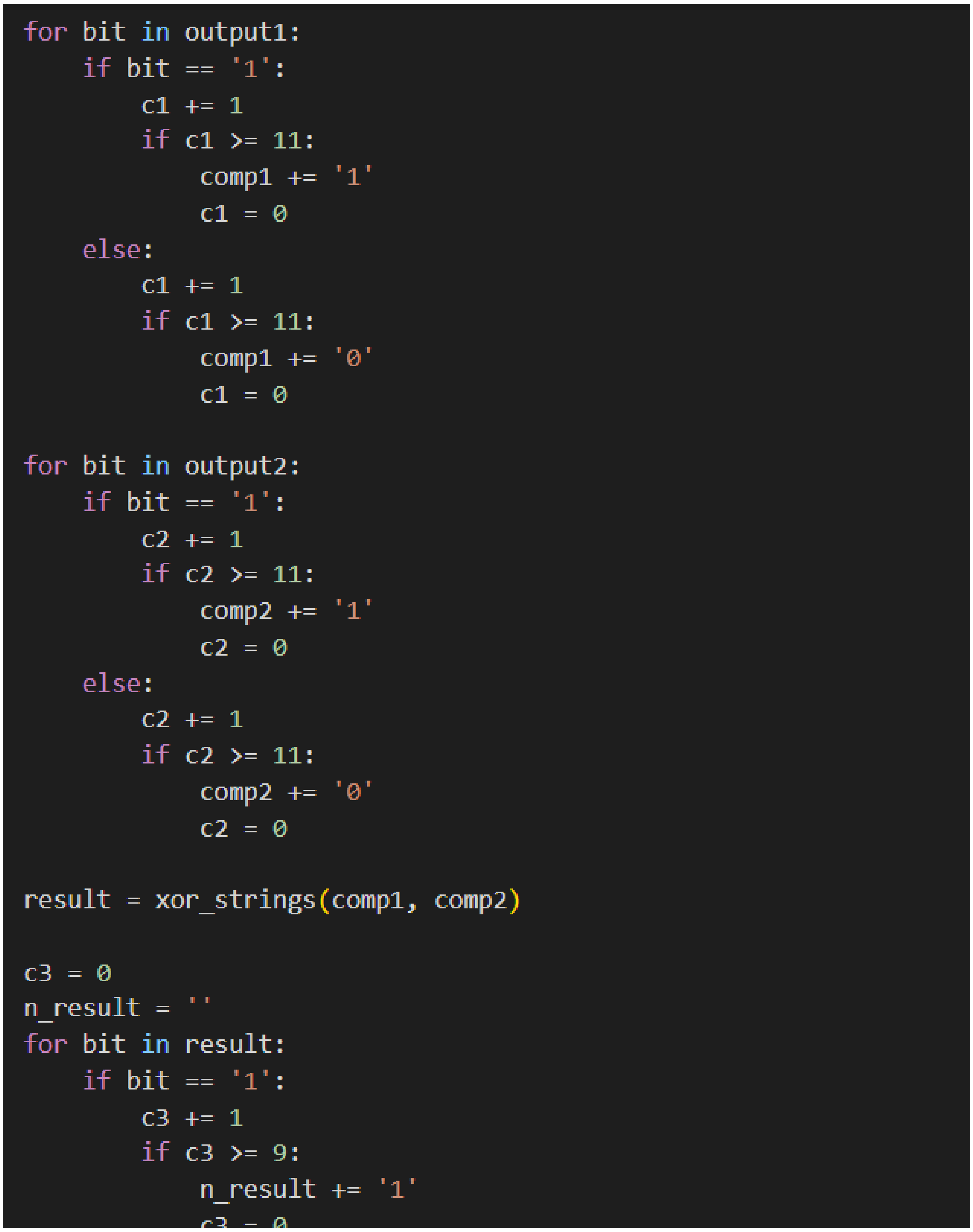
```python
b=0
for b in range(len(output2)):
  if output2[b]=='1':
    c2+=1
    if(c2>=11):
      comp2+='1'
      c2=0
  elif output2[b]=='0':
    c2+=1
    if(c2>=11):
      comp2+='0'
      c2=0
#print("COMP1:", comp1)
#print("COMP2:", comp2)

def xor_strings(str1, str2):

    if len(str1) != len(str2):
        raise ValueError("Input strings must have the same length")


    result = ''
    for char1, char2 in zip(str1, str2):
        result += '1' if char1 != char2 else '0'

    return result


result = xor_strings(comp1, comp2)


c3=0
c=0
```



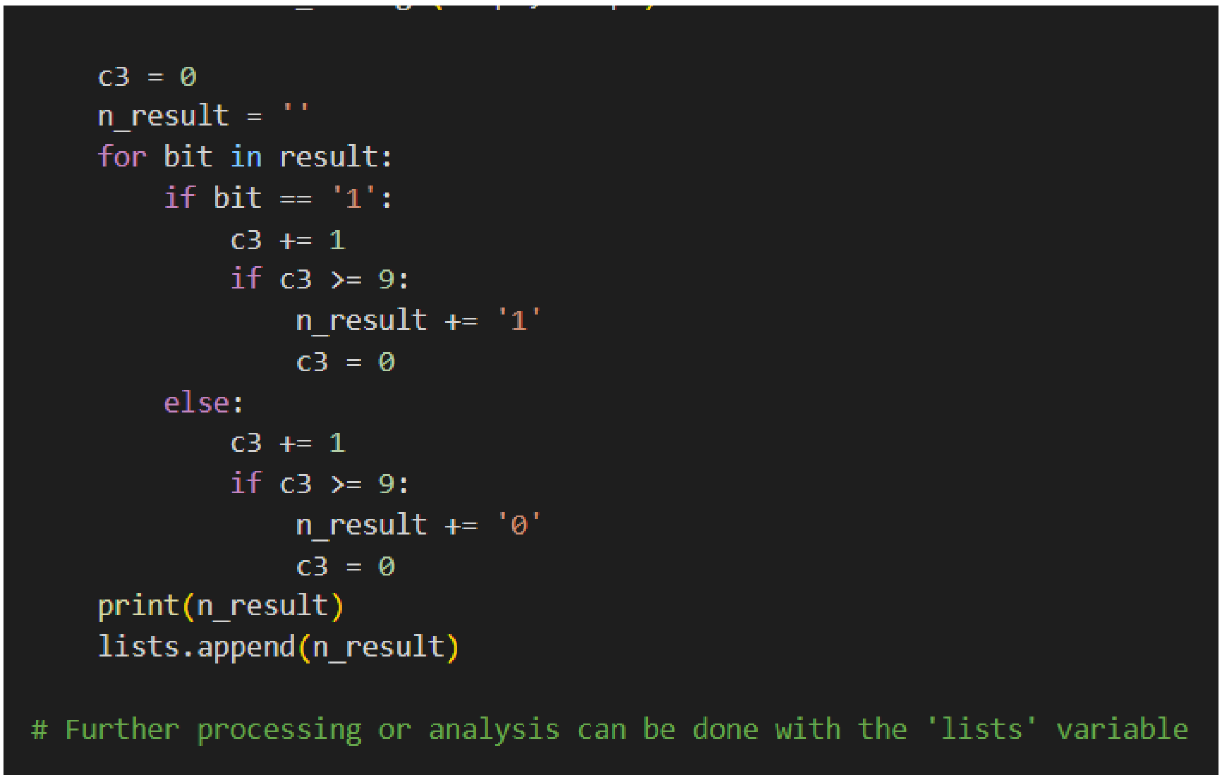
```python
n_result=''
for c in range(len(result)):
  if result[c]=='1':
    c3+=1
    if(c3>=9):
      n_result+='1'
      c3=0
  elif result[c]=='0':
    c3+=1
    if(c3>=9):
      n_result+='0'
      c3=0

#print("N_RESULT:", n_result)


def string_not(binary_string):
    # Perform NOT operation on each character of the string
    result = ''.join('1' if bit == '0' else '0' for bit in binary_string)

    return result

final_result1 = n_result
print(final_result1)


#print(output1)
```



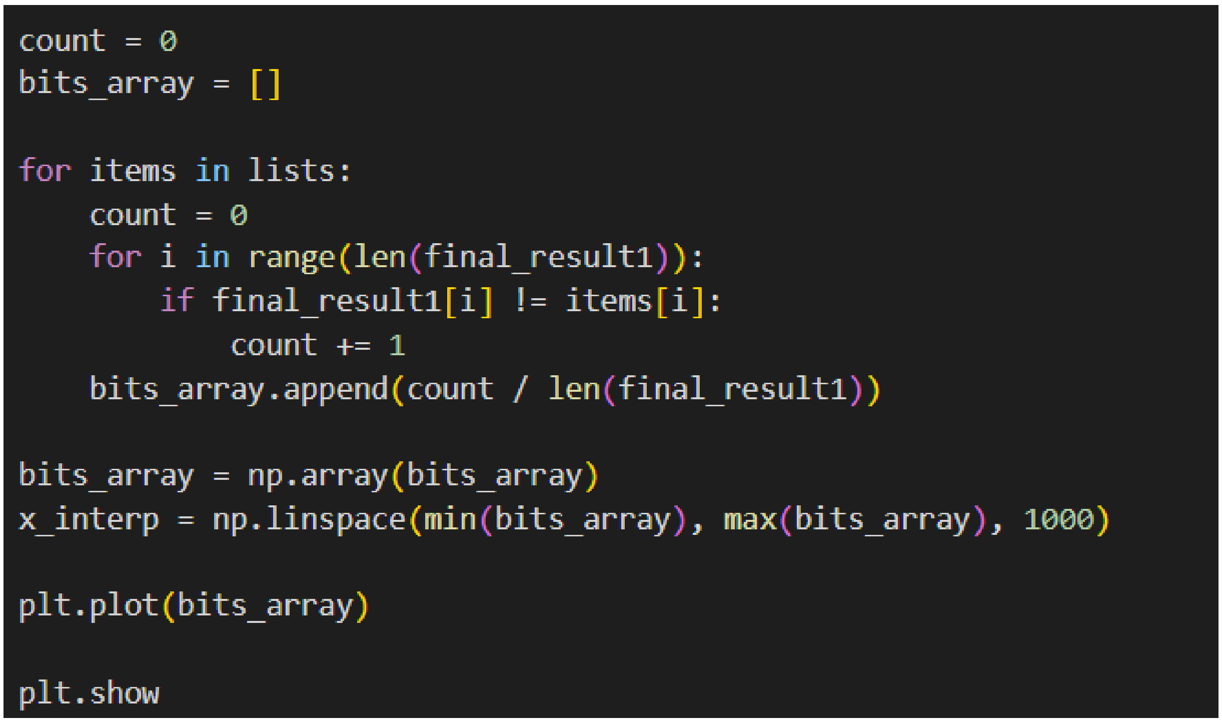
```python
import numpy as np
import matplotlib.pyplot as plt

def xor_strings(str1, str2):
    if len(str1) != len(str2):
        raise ValueError("Input strings must have the same length")

    result = ''
    for char1, char2 in zip(str1, str2):
        result += '1' if char1 != char2 else '0'

    return result

lists = []

# Assuming 'accumulated_wave' and 'noise' are defined elsewhere
for i in range(1,3000):
    noised_x =  0.01*i*accumulated_wave + noise
    output1 = ''
    output2 = ''

    for val in  noised_x:
        output1 += '1' if val > 0 else '0'

    for val in y:
        output2 += '1' if val > 0 else '0'

    c1 = 0
    c2 = 0
    comp1 = ''
    comp2 = ''

    for bit in output1:
        if bit == '1':
```



```python
for bit in output1:
    if bit == '1':
        c1 += 1
        if c1 >= 11:
            comp1 += '1'
            c1 = 0
    else:
        c1 += 1
        if c1 >= 11:
            comp1 += '0'
            c1 = 0

for bit in output2:
    if bit == '1':
        c2 += 1
        if c2 >= 11:
            comp2 += '1'
            c2 = 0
    else:
        c2 += 1
        if c2 >= 11:
            comp2 += '0'
            c2 = 0

result = xor_strings(comp1, comp2)

c3 = 0
n_result = ''
for bit in result:
    if bit == '1':
        c3 += 1
        if c3 >= 9:
            n_result += '1'
            c3 = 0
```

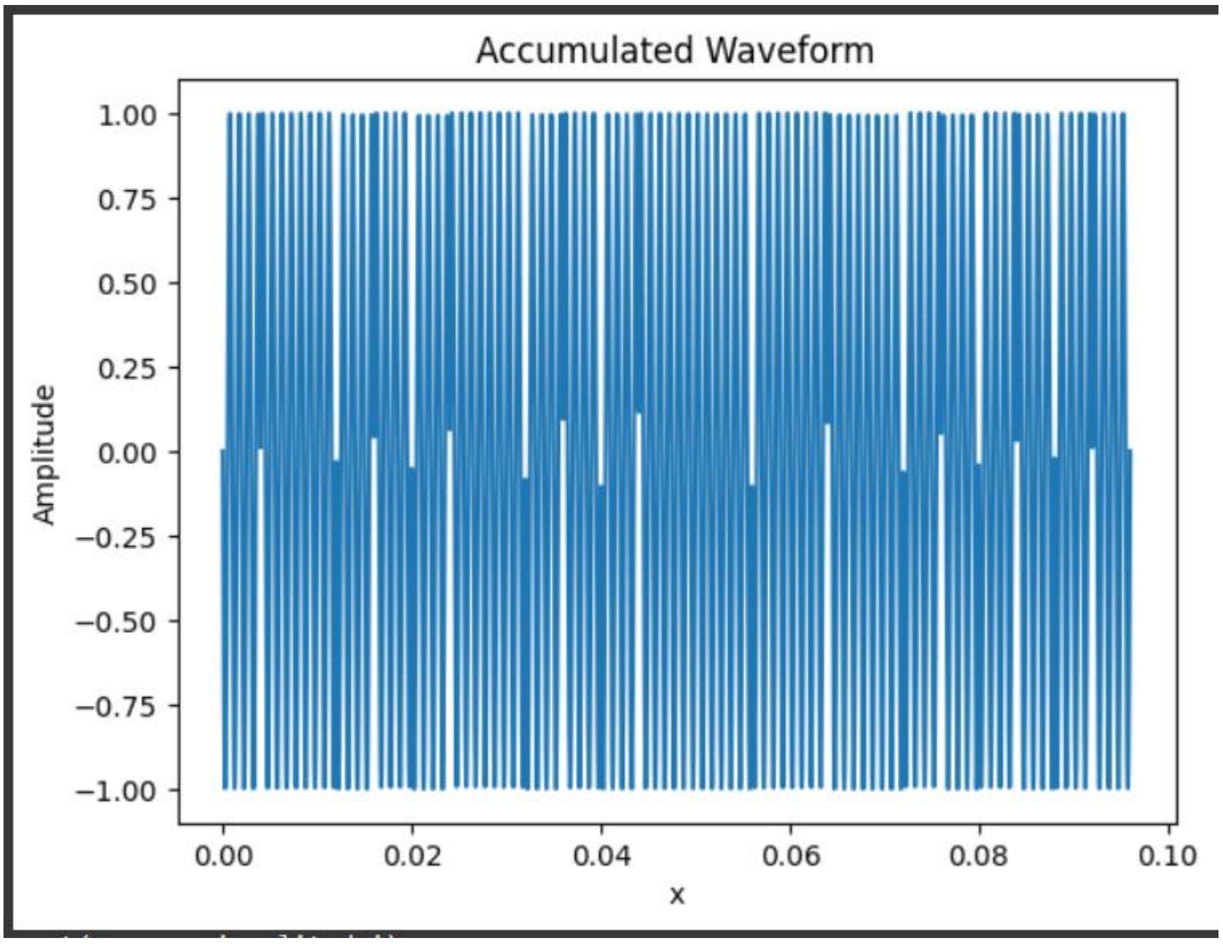


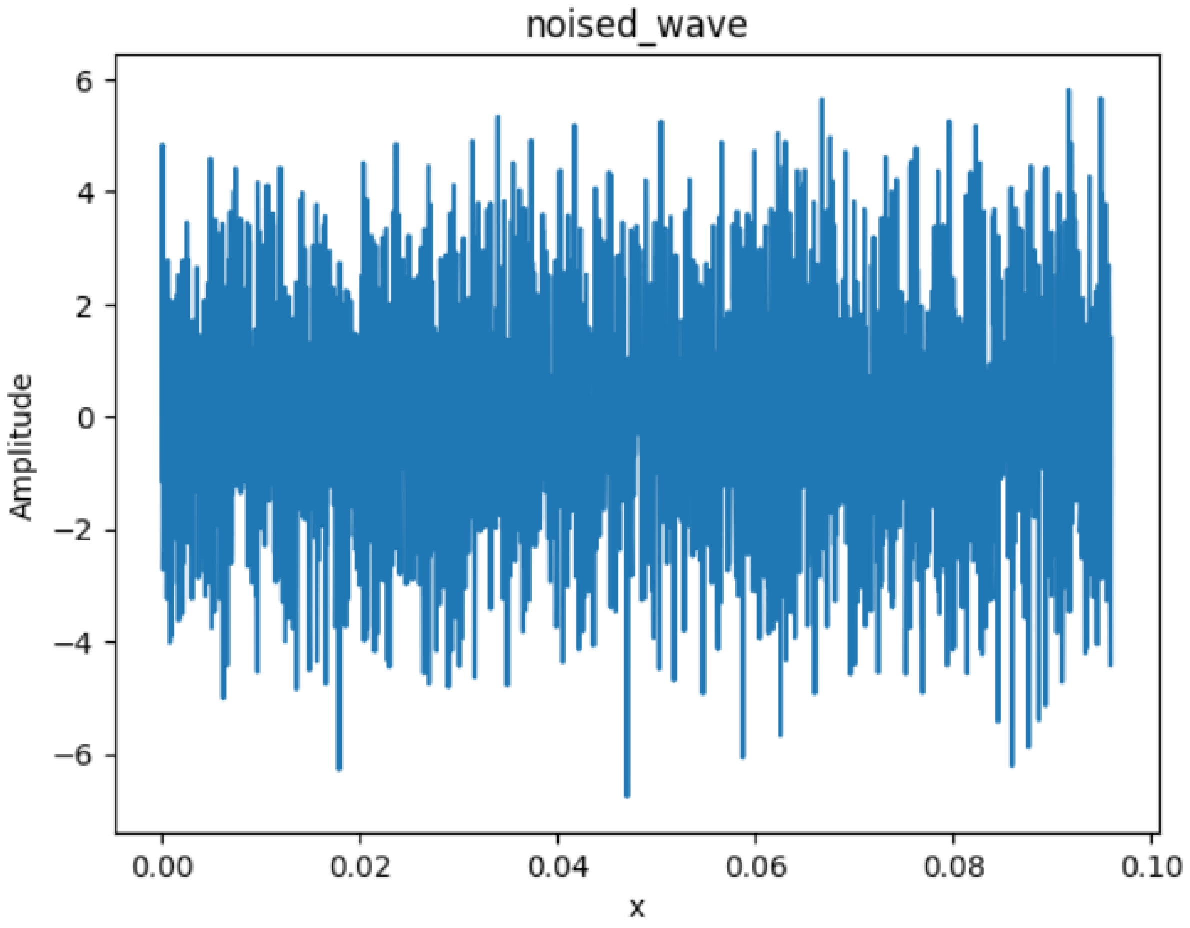
```
    c3 = 0
    n_result = ''
    for bit in result:
        if bit == '1':
            c3 += 1
            if c3 >= 9:
                n_result += '1'
                c3 = 0
        else:
            c3 += 1
            if c3 >= 9:
                n_result += '0'
                c3 = 0
    print(n_result)
    lists.append(n_result)

# Further processing or analysis can be done with the 'lists' variable
```



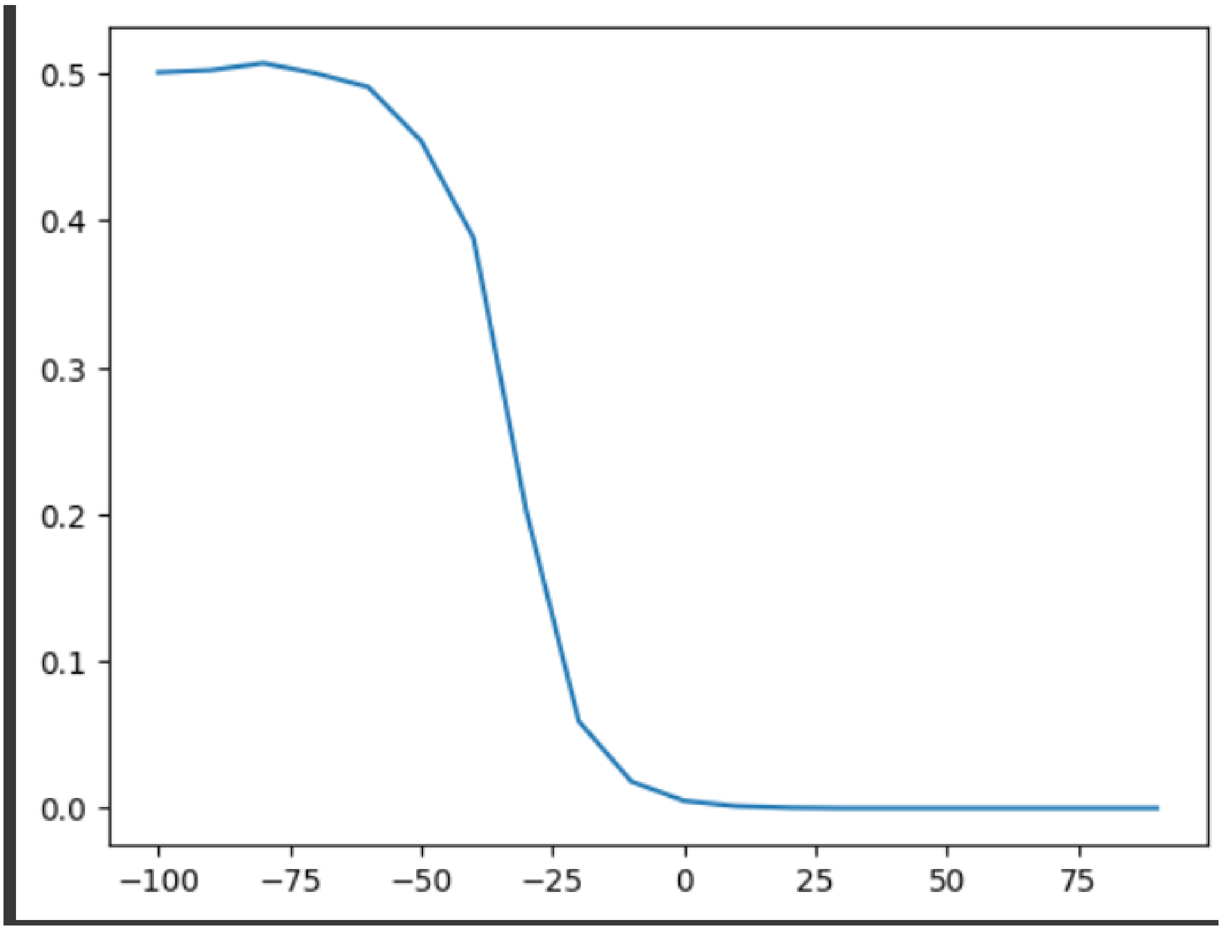
```
count = 0
bits_array = []

for items in lists:
    count = 0
    for i in range(len(final_result1)):
        if final_result1[i] != items[i]:
            count += 1
    bits_array.append(count / len(final_result1))

bits_array = np.array(bits_array)
x_interp = np.linspace(min(bits_array), max(bits_array), 1000)

plt.plot(bits_array)

plt.show
```

Results :


Accumulated Waveform
Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.02
0.04
0.06
0.08
0.10
x


noised_wave
Amplitude
x
6
4
2
0
−2
−4
−6
0.00
0.02
0.04
0.06
0.08
0.10

ERROR BITRATE V/S SNR



Discussion

Binary Phase Shift Keying (BPSK) is a digital modulation technique that encodes binary data onto a sine wave carrier signal by shifting the phase of the carrier to represent the binary symbols 0 and 1. Each binary symbol is mapped to a specific phase shift of the carrier signal, with 0 represented by 0 degrees and 1 represented by 180 degrees. This modulated signal is then transmitted over the communication channel.

- In wireless communication systems like Wi-Fi, Bluetooth, and satellite communication, BPSK ensures efficient transmission of digital data over radio frequency channels.
- Telecommunication systems use BPSK for transmitting digital data over long distances, offering robust performance in noisy environments for applications such as voice communication and data transmission.
- Digital modems rely on BPSK modulation to transmit data over telephone lines and other digital communication channels, enabling reliable and high-speed data transmission.
- Remote sensing applications, including radar systems and satellite communication, utilize BPSK for transmitting and receiving digital signals over long distances.
- Global navigation satellite systems (GNSS) like GPS employ BPSK modulation for transmitting navigation data accurately to receivers worldwide, facilitating precise positioning and timing information.

BPSK modulation provides a simple and efficient means of transmitting digital data over communication channels, making it indispensable in telecommunications, wireless communication, and digital signal processing applications.