Priyanka Raavi

Student No: 200393260

Email: priyankaravi03@gmail.com

Part1: Transmission of Information without noise based on Shannon's model of Communication(Assign1)

The Encoding and Decoding of information using Shannon's model of communication without the interference of noise has been the same as the previous Assignment. But, the slight modifications had been done to the encoding and decoding methods to get the features to work correctly. I had mentioned below, where the changes had made and why is it needed to be done.

```
import math
import struct
import numpy as np
from scipy import signal as sg
import matplotlib
import matplotlib.pyplot as plt
import binascii
import random
```

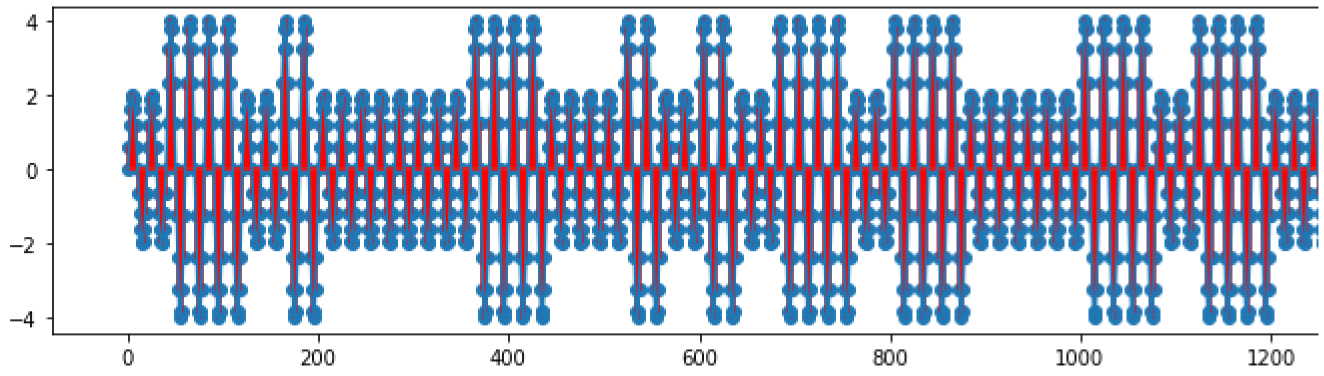
I had taken spb(samples per bit) value as 30 in the Assignment1. But, I have changed the value of spb to 40 now because, as in my encoding scheme I have taken the amplitudes 2 and 4 for 0 and 1 respectively. So, the number of samples per bit should be the multiple of the amplitudes to get the right calculation of the bits. In the decoding Sc

```
spb=40    #Samples per bit
cpb=2     #Cycles per bit
T=spb/cpb #period of Samples
```

```
#converting text to bits
def text_to_bits(text, encoding='ISO-8859-1', errors='surrogatepass'):
    bits = bin(int(binascii.hexlify(text.encode(encoding, errors)), 16))[2:]
    return bits.zfill(8 * ((len(bits) + 7) // 8))
```

```
#encode function
def encode(asciimsg):
  binmsg=text_to_bits(asciimsg)
  message=[]
  for bit in binmsg:
    if bit.isspace():
      pass
    else:
      message.append(int(bit))
  Totalsamples=spb*len(message)
  x=np.arange(Totalsamples)
  carrier=np.sin(2*np.pi*(1/T)*(x))
  messageList=[]
  for bit in message:
    if bit is 0:
      for i in range(0,spb):
        messageList.append(2)
    else:
      for i in range(0,spb):
        messageList.append(4)
  message=messageList
  transsignal=carrier*message
  nestedList=[x,transsignal]
  %matplotlib inline
  fig=plt.figure(figsize=(15,3))
  plt.stem(x,transsignal,'r')
  plt.plot(x,transsignal)
  #print(nestedList)
  return nestedList
```

```
encodeList=encode("hello")       #list contain time and amplitude list
```



```
#convert bits to text
def text_from_bits(bits, encoding='ISO-8859-1', errors='surrogatepass'):
    n = int(bits, 2)
    return int2bytes(n).decode(encoding, errors)

def int2bytes(i):
    hex_string = '%x' % i
    n = len(hex_string)
    return binascii.unhexlify(hex_string.zfill(n + (n & 1)))
```

In the decoding Scheme, after getting the number of crossings when each value in the amplitude list compared with the threshold value. I have changed the number of crossings should be greater than equal to 6 rather than 7 to get append the value to the binaryList. So, I can get the calculation of the binary value of the text correct as each cycle has 3 and 3 points at each side of the cycle.

```
def decode(nestedList):
  transsignal= nestedList[1] #transsignal contain only the amplitud
  #x1=newlist[0]
  amplitudeList=[]
  messageinbits=len(transsignal)//spb #In this case, output of messageinbits (450/3
#creating a list of amplitudes for every bit in a message into a amplitudes.(i.e; amptitude
  for i in range(0,messageinbits):
    tempList=[] #tempList contains the list of amplitudes of
    tempList.clear()
    for j in transsignal[(i*spb):(i+1)*spb]:
      tempList.append(j)
    amplitudeList.append(tempList)
#Decoding Scheme
  threshold =2.5
  binaryList = []
  for i in range(0,len(amplitudeList)):
    crossing = 0
    for x in amplitudeList[i]:
      if x >= threshold:
        crossing = crossing + 1
    if crossing >=6:
      binaryList.append(1)
    else:
      binaryList.append(0)
#storing the list of binary values into a variable
  for i in range(0,len(binaryList)):
    binary= ''+str(binaryList[i])
  binarymsg =''.join(str(i) for i in binaryList) #final binary form of a message
  textmessage=text_from_bits(binarymsg)
  #print(binarymsg)
  return textmessage

print(decode(encodeList))
```

```
hello
```

PartII: The Discrete Channel with Noise- Assignment3

In the previous assignment, we had spoken about the broadcasting symbols over the channel without the presence of the noise.

Noise: The modern definition of noise is that "Any unwanted modification to a signal that is not part of an original message." which also mean that, the difference between the original signal and the resultant signal is noise.

This noise in the channel can be any kind, including amplitude modulation noise. A lot of different sources of noise that are not random had been discovered now. But, Shannon definition is a very narrow idea of noise. According to him, the noise had no reason or rhyme. It was just a random number of generations that added to the original signal. Based on his idea of noise, the definition of noise can be assumed as, the received signal (E) to be a function of the transmitted signal (S) and a second variable, Noise(N)[1].

i.e., E=f(S, N) where Eis the resultant signal, S is the original signal and N is the noise

So, In the case of no Noise, which is an ideal situation, E=S, which means, the resultant signal is equal to the original signal.

But such cases won't happen in the real world in any situation. Usually, the resultant signal(E) is a function that depends more on the noise present in it. Let us take the radio as an example. When we were turning through the channels of the radio, we will hear a lot of interference from different other channels or when the channels overlap, etc.. This interference or the disturbance is not random. It might be because of the electromagnetic waves or cosmic waves in the air.

There are many kinds of noise. They are:

1. Additive Noise
2. Multiplicative Noise
3. Phase Noise
4. Quantization Noise and so on.

As of now, we will discuss Additive Noise. To an original signal sine wave, if we have noise in the form of an unexpected burst in amplitude. Then that kind of noise is called additive noise. There are many types of additive noise. They are named after colors. In those, the most common being used is the white noise. White Noise: It is called white noise because all of the different frequencies are equally possible. And, It is random in such a way that it is equally random.

Other additive noises are Pink Noise, Brown Noise, Black Noise, Cauchy Noise, and so on

Now, To test the encoding-decoding scheme, we will create a noise and add that noise to the encoding scheme and see how the encoding scheme is getting effected and how accuracy that the encoding scheme can be with the noise. The noise that we are generating in this assignment is the Additive white Gaussian noise(AWGN), which is the only kind of white noise. I'm using generating this noise as it is the basic noise used in Information Theory.

PartA: Make Some Noise!

To generate the white noise, I have defined a function that takes the noise Amplitude and the Signal Length as arguments. We can generate the noise by taking the list of random values based on the noise amplitude. And, returns a list of amplitude values that is of the same length as the original signal(i.e., the signal length will be the length of the original signal).

The generation of random values will be in between -100 to 100 where it follows the normal distribution that provides the mean average as 0 and as the carrier signal range is in between -1 to 1 in the previous assignment.

This noise() provide the equal distribution of random values within the range based on the noise amplitude. The value of the Noise amplitude I can give as an argument to noise will be in the range of 1-2 based on my encoding scheme used in the previous assignment.

```
#noise generation
def noise(Namplitude, signallength):
  noiseList = []
  for x in range(signallength):
    noiseList.append((random.randint(-100,100)*(Namplitude))/100)

  return noiseList
```

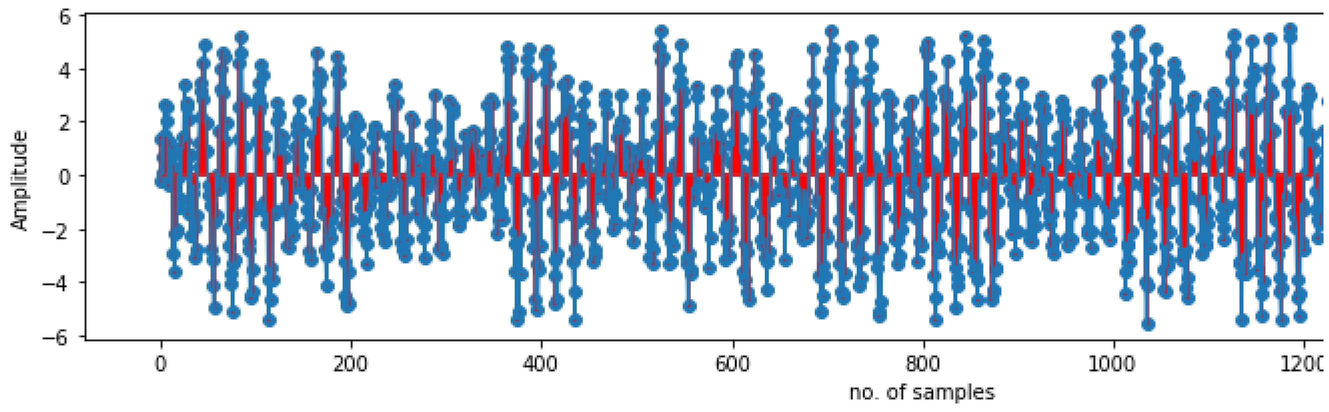
Part B: Cross the streams!

We will add this generated noise to the original signal then we will get the combined noiseaddedsignal. As we can see the below plotted figure, there is a lot of fluctuation in the amplitudes of the signal when compared to the encoded signal plot. Which means that, there is some noise added to the original signal and that made a change in the message that need to be transmitted.

Here, I have provided only one instance of noise amplitude and checked.

```
Namplitude = 1.6
asciimsg="hello"
slist = encode(asciimsg)
signal = slist[1]
signallength = len(signal)
noise(Namplitude, signallength)
signalwithnoise = []
signalwithnoise = signal + noise(Namplitude, signallength)   # adding signal with the noise
x=np.arange(signallength)
decodesignal=[x,signalwithnoise]
%matplotlib inline
fig=plt.figure(figsize=(15,3))
plt.stem(x,signalwithnoise,'r')
plt.plot(x,signalwithnoise)
plt.xlabel("no. of samples")
plt.ylabel("Amplitude")
```

```
Text(0, 0.5, 'Amplitude')
```



As we can see, there is an uneven increase in the amplitudes of signal at every point which effect the original message or signal.

Now, we got the noise added signal and test our encoding and decoding scheme efficiency, we will pass the noise added signal to the decode function as an argument.

```
DecodedMessage = decode(decodesignal)
print("received Msg:",DecodedMessage,"\n\n\n")
```

```
received Msg: hel}o
```

Part C: Repeat and Analyze

To repeat the partB and check for multiple noise amplitudes from 1.0 to 1.9 which are taken based on the scheme used. To calculate the accuracy and the SNR value, I have developed an SNR() to calculate the SNR value based on various noise amplitude and the average of the signal amplitudes.(which is around the threshold value). To calculate the accuracy, I have explicitly wrote a function which takes difference between the asciimessage and the noise value. The compareString method is been created to compare the actual message and the noised message to know how many characters are different from each other and keep a track using counter. Using this counter value, we can calculate the accuracy for each noise amplitude.

```
#SNR
def SNR(Namplitude):
  snr = 2.5/Namplitude
  return snr

#comparing 2 strings to get the character difference
def compareString(asciimsg,noisedmsg):
  counter = 0
  for i in range(len(asciimsg)):
    if asciimsg[i]!=noisedmsg[i]:
      counter = counter + 1
  return counter

#calculating accuracy
```

```
def getaccuracy(asciimsglength, Noise):
  differinchar = asciimsglength - Noise
  return (differinchar/asciimsglength)*100
```

Calculated SNR and accuracy values for noise amplitude value 1.6. The accuracy is been calculated using the length of the ascii message and the counter.

```
counter = compareString(asciimsg,DecodedMessage)
snr = SNR(Namplitude)
accuracy = getaccuracy(len(asciimsg),counter)
print("snr:",snr)
print("accuracy:",accuracy,"%\n")
```

```
snr: 1.5625
accuracy: 80.0 %
```

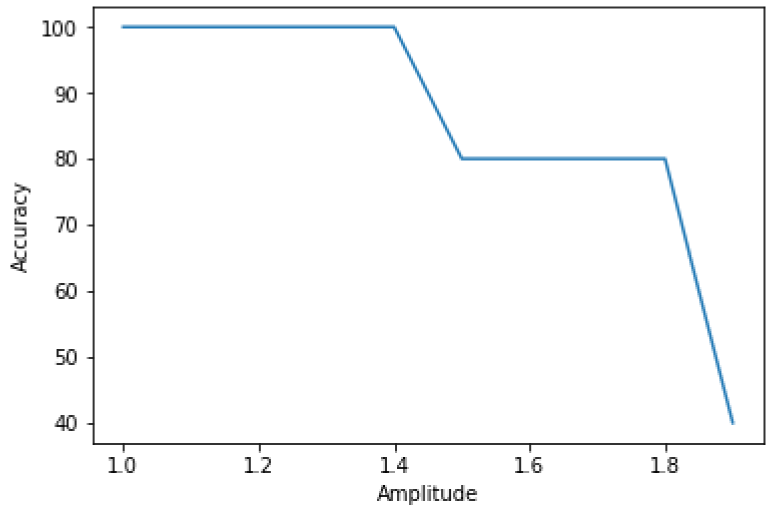
The noisedSignal() is the method that does complete set. This function first calls the encode function and then the noise() is been called and then will add both noise and signal so that we can get the noise added signal. Now, It will run the decode() and receive the message that may or maynot have an error. And then, It calculates the SNR and accuracy values for each noise amplitude from 1.0 to 1.9

```
#noised signal function to get calculate the accuracy at various points of SNR
def noisedSignal(asciimsg):

  List = encode(asciimsg)
  signal = List[1]
  signallength = len(signal)
  accuracylist = []
  amplitude = []
  accuracyAmplitude = []
  for i in range(10,20):
    Namplitude = i/10
    noiseList = noise
    noiseList = noise(Namplitude, signallength)
    signalwithnoise = []
    signalwithnoise = signal + noise(Namplitude, signallength)
    x=np.arange(signallength)
    decodesignal=[x,signalwithnoise]
    decodemsg = decode(decodesignal)
    counter = compareString(asciimsg,decodemsg)
    snr = SNR(Namplitude)
    accuracy = getaccuracy(len(asciimsg),counter)
    accuracylist.append(accuracy)
    amplitude.append(Namplitude)
    print("Amplitude:",Namplitude)
    print("accuracy:",accuracy,"%\n")
  accuracyAmplitude.append(accuracylist)
  accuracyAmplitude.append(amplitude)
  return accuracyAmplitude
```

To have better understanding of the accuracy values for each noise amplitude, we are plotting a graph over the amplitude in the x-axis and accuracy placing in the Y-axis.

```
accuracyAmplitude = noisedSignal(asciimsg)
accuracylist = accuracyAmplitude[0]
amplitude = accuracyAmplitude[1]
%matplotlib inline
plt.plot(amplitude,accuracylist)
plt.xlabel("Amplitude")
plt.ylabel("Accuracy")
```

```
Amplitude: 1.0
accuracy: 100.0 %

Amplitude: 1.1
accuracy: 100.0 %

Amplitude: 1.2
accuracy: 100.0 %

Amplitude: 1.3
accuracy: 100.0 %

Amplitude: 1.4
accuracy: 100.0 %

Amplitude: 1.5
accuracy: 80.0 %

Amplitude: 1.6
accuracy: 80.0 %

Amplitude: 1.7
accuracy: 80.0 %

Amplitude: 1.8
accuracy: 80.0 %

Amplitude: 1.9
accuracy: 40.0 %

Text(0, 0.5, 'Accuracy')
```



Part D: Discussion

As we can see the plot, my algorithm is good enough until the 1.4 amplitude constantly. But, It is getting effected after increasing in the noise amplitude, which means the increase in the noise over the signal in a channel. To

Increase the robustness to my algorithm, In my assignment4 I would like to use the Triple Redundancy Forward Error Correction algorithm to rectify few errors.

References

1. C. E. Shannon, "A mathematical theory of communication," in The Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, July 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6773024&isnumber=6773023