Transmission of Information based on Shannon's model of Communication

Priyanka Raavi

Student No: 200393260

Email: priyankaravi03@gmail.com

According to Shannon's model of communication, the information or the message would be transfer from the sender to the transmitter, which acts as a modulator (converts the message in to signal). That signal is transferred through the noise and received by the receiver, which acts as a demodulator(changes the signal into a message) and sends it to the destination.

In this Notebook, we are going to see how a message can be transferred from the Information source to the destination by following Shannon's model of communication without considering the noise as of now.

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

If we consider the model of communication, we can see that there are two parts of the information transmission. One is the modulation of the information and sends to the receiver, and the other is demodulation of information and send to the destination.

So, In this notebook, we are implementing both parts using two methods. One is encode() for the modulation of message and decode() to demodulate the message.

To convert the message into a Signal, we need to modulate the message using PCM(Pulse Code Modulation).

Before that, let's get the knowledge of basic terminology and concepts related to modulation and types of modulation and sampling rate.

Modulation: It is a process of encoding information into the transmissible signal. However, how do we do it?. We can do it by combining the ***carrier signal*** and the ***information signal***(whether it is in analog or digital) provide the transmissible signal.

1. **Information Signal:** It is just a amplitudes list of a particular message.
2. **Carrier Signal:** It is just a sequence of an oscillating signal that vary about central value over time(merely a waveform.eg: sine wave).This carrier signal is which we modulate based on the information signal.

PCM: It is a process of breaking the analog signal into discrete values based on sampling rate(no. of samples per second). The standard sampling rate is 44100 samples for a second.

Types of Modulation: We have 2 ways to modulate the signal.

1. **Amplitude Modulation:** It is a way of modulating the signal by varying the amplitude and by taking frequency as constant.
2. **Frequency Modulation:** In FM, we take amplitude as constant and vary the frequency of the signal.

In this Notebook, we are using Amplitude Modulation to encode and decode the message because it is a bit easy to implement the decoding of the signal to message.

Here, we are initializing sbp(Samples per bit) and cpb(Cycles per bit) globally to get access to both encode() and decode() methods.

```
spb=30    #Samples per bit
cpb=2     #Cycles per bit
T=spb/cpb #period of Samples
```

Part1: Converting the Message into a signal using Amplitude Modulation

To convert the message into a signal, we need to generate the message signal and carrier signal. The initial step is to generate a carrier signal is to find out the number of samples and period of samples, which are used in the calculation of the formula f(x)=Asin(2pi*x).

To generate the message signal, we are using an encoding scheme by making the frequency constant and vary in Amplitude. If the bit is 0, then we encode those set of samples with an amplitude 2times of the carrier signal amplitude or else if the bit is 1, then we encode that set of samples with 4 times of the carrier signal amplitude.

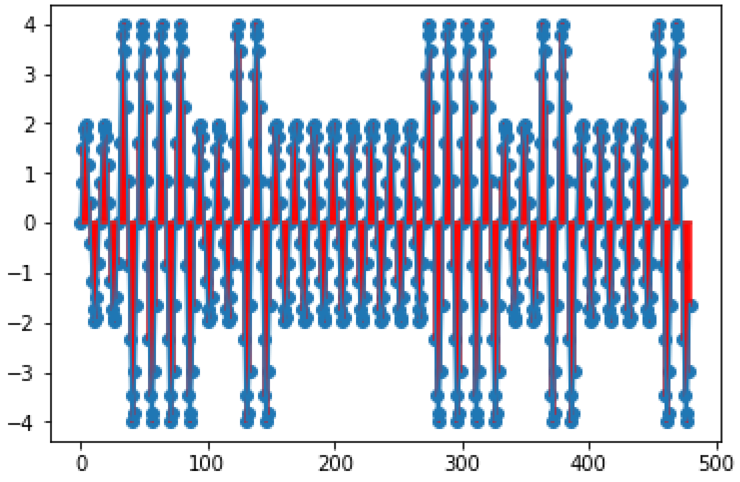
By multiplying both the carrier signal and the message signal, give the result as the transmission signal, that generated in the transmitter. That transmitter signal will be sent to the receiver.

```
#convert the message into binary
def text_to_bits(text, encoding='utf-8', errors='surrogatepass'):
    bits = bin(int.from_bytes(text.encode(encoding, errors), 'big'))[2:]
    return bits.zfill(8 * ((len(bits) + 7) // 8))
```

```
def encode(asciimsg):
  binmsg= text_to_bits(asciimsg)
  message = []
  for bit in binmsg:
    message.append(int(bit))
  Totalsamples = spb*len(message)                            #total 450 samples
  x = np.arange(Totalsamples) # the points on the x axis for plotting
  carrier = np.sin(2*np.pi*(1/T)*(x))                        #carrier signal list
  messageList = []
  #encoding Scheme
  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                                      #message signal
  transsignal= carrier*message                               #generated trnasmission signal
  #here we are creating a nested list of both time and amplitude to send that list as input t
  nestedList=[x,transsignal]
  #this instruction can only be used with IPython Notbook.
  % matplotlib inline
  # showing the exact location of the smaples
  plt.stem(x,transsignal, 'r' )
  plt.plot(x,transsignal)
  #nestedList
  return nestedList
```

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

```
'0110100001101001'
```



Part2: Converting the signal into a message using Amplitude Modulation

To decode the signal, we are taking the output of the encode() as an input to the decode() and decoding the signal by the following steps:

Step1: Take the amplitude list from the nestedList and calculating the count of bits in a message because that amplitude list is the message signal amplitude.

Step2: By creating a list of amplitude lists for every bit in a message, we can easily decode by comparing each amplitude point with the threshold.

Step3: Set a threshold value

Step4: Create a binary list and compare the every amplitude value of every bit in a message with a threshold value. If the amplitude value is higher than the threshold then we take it as 1 or else we take it as 0. store all the binary values into that list.

Step5: converting the binary into a text

```
def text_from_bits(bits, encoding='utf-8', errors='surrogatepass'):
  text = int(bits, 2)
  return text.to_bytes((n.bit_length() + 7) // 8, 'big').decode(encoding, errors) or '\0'
```

```
def decode(nestedList):
  transsignal= nestedList[1]                          #transsignal contain only the amplitu

  amptitudeList=[]
  #a//b- floor division operator
  messageinbits=len(transsignal)//spb           #In this case, output of messageinbits (450/

  #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 o
    tempList.clear()
    for j in transsignal[(i*spb):(i+1)*spb]:
      tempList.append(j)
      amptitudeList.append(tempList)

    #Decoding Scheme
  threshold = 3
  binaryList = []
  for i in range(0,len(amptitudeList)):
    crossing = 0
    for x in amptitudeList[i]:
      if x >= threshold:
        crossing = crossing + 1
      if crossing >=7:
        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
  text_from_bits(binarymsg)
  return binarymsg
```

```
decode(encodeList)
```

short-commings

1. In decode(), problem with storing the correct binarylist values.
2. In decode(), I couldn't able to store the binary list into a variable.

References

1.Tomesh, T. (2019). generatingSignals.ipynb. [online] Gist. Available at: https://gist.github.com/trevortomesh/7df84a9e049ebeeed09f765ba22598f0 [Accessed 20 Jun. 2019].

2.isidore.john.r at gmail dot com.[online] Available at: https://stackoverflow.com/questions/7396849/convert-binary-to-ascii-and-vice-versa