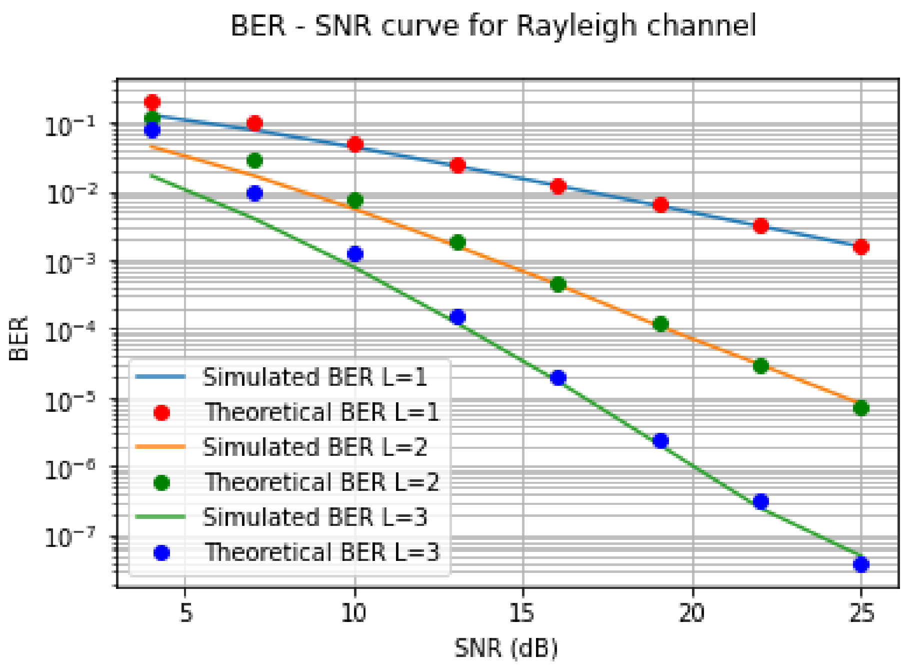
EE670A
PYTHON ASSIGNMENT #2
M. Aravind, 21104403

```python
"""
EE 670A
Python Assignment #2
Submitted by: M. Aravind
20-10-2021
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.random as nr
from math import comb


rblockLength = 10000 # Samples per block
nBlocksr = 1000 # Number of blocks. Multipltication of this with above gives total number of symbols. Here it is taken 10^7, So that the empirical curve follows analytical atleast till 10^-7 BER.
No=1
EbdBr = np.arange(1,35,3)
Ebr=10**(EbdBr/10)
SNRr = 2*Ebr/No;
SNRdBr= 10*np.log10(SNRr);
BERr = np.zeros(len(EbdBr))
BERrt = np.zeros(len(EbdBr))

#Start Plotting
plt.figure()
plt.yscale('log');

# L=1 antenna
L=1
for blk in range (nBlocksr):
    BitsIr = nr.randint(2,size=rblockLength)
    BitsQr = nr.randint(2,size=rblockLength)
    Sym_r = (2*BitsIr-1)+1j*(2*BitsQr-1)
    noise_r = nr.normal(0,np.sqrt(No/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(No/2),size=(L,rblockLength)) #of L x blocklength dimension correspoinding to L paths
    h = nr.normal(0,np.sqrt(1/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(1/2),size=(L,rblockLength)) #Rayleigh Fading Channel (Similarly of L x blocklength dimension)
    for k in range(len(EbdBr)):
        TxSym_r = np.sqrt(Ebr[k])*Sym_r #Transmited symbol through Rayleigh Channel
        RxSym_r = h*TxSym_r + noise_r
        EqSym_r_h = RxSym_r*(np.conj(h))/(np.linalg.norm(h,1)) #MRC
        EqSym_r = np.sum(EqSym_r_h,0) #Converting from SIMO to SISO for easy MRC simplification (Adding up columns of the Lxblocklength matrix to create a single vector) Here h is an Lxblocklength vector (representing L blocklength duration vectors or L different channel taps)
        DecBitsIr = (np.real(EqSym_r)>0)
        DecBitsQr = (np.imag(EqSym_r)>0)
        BERr[k] = BERr[k]+ np.sum(DecBitsIr != BitsIr) + np.sum(DecBitsQr != BitsQr)
BERr = BERr/rblockLength/2/nBlocksr
BERrt = comb(2*L-1,L)/((2*SNRr)**L)
plt.plot(SNRdBr,BERr)
plt.plot(SNRdBr,BERrt,'ro')

# L=2 antenna
L=2
for blk in range (nBlocksr):
    BitsIr = nr.randint(2,size=rblockLength)
    BitsQr = nr.randint(2,size=rblockLength)
    Sym_r = (2*BitsIr-1)+1j*(2*BitsQr-1)
    noise_r = nr.normal(0,np.sqrt(No/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(No/2),size=(L,rblockLength))
    h = nr.normal(0,np.sqrt(1/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(1/2),size=(L,rblockLength))
```

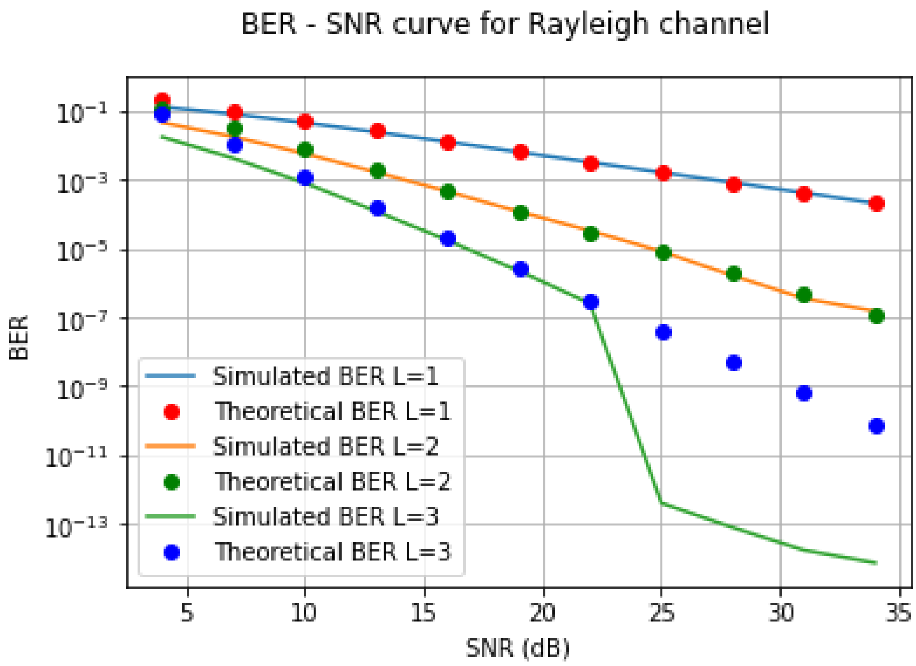
channel is assummed to be constant over one block. This is known as Block Fading channel

```
    for k in range(len(EbdBr)):
        TxSym_r = np.sqrt(Ebr[k])*Sym_r
        RxSym_r = h*TxSym_r + noise_r
        EqSym_r_h = RxSym_r*(np.conj(h))/(np.linalg.norm(h,1)) #MRC
        EqSym_r = np.sum(EqSym_r_h,0)
        DecBitsIr = (np.real(EqSym_r)>0)
        DecBitsQr = (np.imag(EqSym_r)>0)
        BERr[k] = BERr[k]+ np.sum(DecBitsIr != BitsIr) + np.sum(DecBitsQr != BitsQr)
BERr = BERr/rblockLength/2/nBlocksr
BERrt = comb(2*L-1,L)/((2*SNRr)**L)
plt.plot(SNRdBr,BERr)
plt.plot(SNRdBr,BERrt, 'go')

# L=3 antenna
L=3
for blk in range (nBlocksr):
    BitsIr = nr.randint(2,size=rblockLength)
    BitsQr = nr.randint(2,size=rblockLength)
    Sym_r = (2*BitsIr-1)+1j*(2*BitsQr-1)
    noise_r =
nr.normal(0,np.sqrt(No/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(No/2),size=(L,rblockLength))
    h =
nr.normal(0,np.sqrt(1/2),size=(L,rblockLength))+1j*nr.normal(0,np.sqrt(1/2),size=(L,rblockLength))
    for k in range(len(EbdBr)):
        TxSym_r = np.sqrt(Ebr[k])*Sym_r
        RxSym_r = h*TxSym_r + noise_r
        EqSym_r_h = RxSym_r*(np.conj(h))/(np.linalg.norm(h,1))
        EqSym_r = np.sum(EqSym_r_h,0)
        DecBitsIr = (np.real(EqSym_r)>0)
        DecBitsQr = (np.imag(EqSym_r)>0)
        BERr[k] = BERr[k]+ np.sum(DecBitsIr != BitsIr) + np.sum(DecBitsQr != BitsQr)
BERr = BERr/rblockLength/2/nBlocksr
BERrt = comb(2*L-1,L)/((2*SNRr)**L)
plt.plot(SNRdBr,BERr)
plt.plot(SNRdBr,BERrt, 'bo')

plt.grid(1,which='both')
plt.suptitle('BER - SNR curve for Rayleigh channel')
plt.xlabel('SNR (dB)')
plt.ylabel('BER')
plt.legend(["Simulated BER L=1", "Theoretical BER L=1", "Simulated BER L=2", "Theoretical BER L=2", "Simulated BER L=3", "Theoretical BER L=3" ])
```

<u>Output</u>:
(Truncated till BER = 10^{-7} for L=3)



(Plotted till SNR = 32 dB; corresponding to BER 10^{-7} for L=2)
(Here, BER crosses 10^{-6} for both L=2 and L=3)



Why the sharp drop in BER after 10^{-7} for L=3? It is because, in the code, I took only 10^7 samples. Had I taken 10^{13} samples (blocklength x number of blocks), then the simulated (or empirical) plot would have followed the theoretical (or analytical) plot till BER 10^{-13}. It is that I stopped at 10^7 samples in the code, for computational simplicity. (Calculations for 10^{13} samples take a very long time) Hence the sharp dip indicates just the deficiency of the number of symbols I took.

Observation:

Bit Error Rate, BER for an L antenna system with Maximal Ratio Combining at the receiver end (high SNR approximation) is:

$$BER = C_L^{2L-1} \frac{1}{(2SNR)^L}$$

For L=2, BER $\geq 10^{-6} \rightarrow$ SNR $\leq$ 30.88 dB
For L=3, BER $\geq 10^{-6} \rightarrow$ SNR $\leq$ 20.32 dB

Hence I plotted till around 35 dB.

From the above BER vs SNR plots, it can be easily observed that as we increase the number of receive antennas (L), BER of the system will decrease.

or $BER_{L=3} < BER_{L=2} < BER_{L=1}$

or $BER \propto \frac{1}{SNR^L}$

Also, the empirical (or simulated) plot was found to follow the theoretical/formula based (analytical) plot.

30/30