**CDMA**

c1 = [1, 1, 1, 1]

c2 = [1, -1, 1, -1]

c3 = [1, 1, -1, -1]

c4 = [1, -1, -1, 1]

print('Enter the data bits:')

d1 = int(input('Enter D1: '))

d2 = int(input('Enter D2: '))

d3 = int(input('Enter D3: '))

d4 = int(input('Enter D4: '))

r1 = [x \* d1 for x in c1]

r2 = [x \* d2 for x in c2]

r3 = [x \* d3 for x in c3]

r4 = [x \* d4 for x in c4]

resultant\_channel = [r1[i] + r2[i] + r3[i] + r4[i] for i in range(len(c1))]

print('Resultant Channel:')

print(resultant\_channel)

channel = int(input('Enter the station to listen for C1=1, C2=2, C3=3, C4=4: '))

if channel == 1:

rc = c1

elif channel == 2:

rc = c2

elif channel == 3:

rc = c3

elif channel == 4:

rc = c4

inner\_product = sum([a\*b for a, b in zip(resultant\_channel, rc)])

print('Inner Product:')

print(inner\_product)

data = inner\_product / len(rc)

print('Data bit that was sent:')

print(data)

**BER**

clc;

clear;

N = 10^6; % number of bits or symbols

% Transmitter

ip = rand(1, N) > 0.5; % generating 0,1 with equal probability

s = 2\*ip - 1; % BPSK modulation 0 -> -1; 1 -> 0

Eb\_N0\_dB = (-3:35); % multiple Eb/N0 values

for ii = 1:length(Eb\_N0\_dB)

n = (1/sqrt(2)) \* (randn(1, N) + 1i\*randn(1, N)); % white Gaussian noise, 0dB variance

h = (1/sqrt(2)) \* (randn(1, N) + 1i\*randn(1, N)); % Rayleigh channel

% Channel and noise Noise addition

y = h.\*s + 10^(-Eb\_N0\_dB(ii)/20)\*n;

% equalization

yHat = y./h;

% receiver - hard decision decoding

ipHat = real(yHat) > 0;

% counting the errors

nErr(ii) = sum(ip ~= ipHat);

end

simBer = nErr/N; % simulated ber

theoryBerAWGN = 0.5\*erfc(sqrt(10.^(Eb\_N0\_dB/10))); % theoretical ber

EbN0Lin = 10.^(Eb\_N0\_dB/10);

theoryBer = 0.5.\*(1-sqrt(EbN0Lin./(EbN0Lin+1)));

% plot

figure; % open a new figure window

plot(Eb\_N0\_dB, theoryBerAWGN, 'cd-', 'LineWidth', 2);

hold on;

plot(Eb\_N0\_dB, theoryBer, 'r--', 'LineWidth', 2);

plot(Eb\_N0\_dB, simBer, 'bs-', 'LineWidth', 2);

hold off;

grid on;

title('BER for BPSK modulation in Rayleigh channel');

xlabel('Eb/No, dB');

ylabel('Bit Error Rate');

legend('AWGN-Theory', 'Rayleigh-Theory', 'Rayleigh-Simulation');

% Save figure to the PC

saveas(gcf, 'BER\_plot.png');

**TCP**

**Server.py**

import socket

IP = socket.gethostbyname(socket.gethostname())

PORT = 4455

ADDR = (IP, PORT)

SIZE = 1024

FORMAT = "utf-8"

def main():

print("[STARTING] Server is starting.")

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(ADDR)

server.listen()

print("[LISTENING] Server is listening.")

while True:

conn, addr = server.accept()

print(f"[NEW CONNECTION] {addr} connected.")

filename = conn.recv(SIZE).decode(FORMAT)

print(f"[RECV] Receiving the filename: {filename}")

conn.send("Filename received.".encode(FORMAT))

with open(filename, "wb") as file: # Open the file in binary write mode

while True:

data = conn.recv(SIZE)

if not data:

break

file.write(data)

conn.send("Data received.".encode(FORMAT))

print(f"[FILE RECEIVED] File {filename} received from {addr}.")

conn.close()

print(f"[DISCONNECTED] {addr} disconnected.")

if \_name\_ == "\_main\_":

main()

**Client.py**

import socket

IP = socket.gethostbyname(socket.gethostname())

PORT = 4455

ADDR = (IP, PORT)

FORMAT = "utf-8"

SIZE = 1024

def main():

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(ADDR)

file = open("Data/yt.txt", "rb") # Open the file in binary mode for correct transmission

data = file.read()

client.send("yt.txt".encode(FORMAT))

msg = client.recv(SIZE).decode(FORMAT)

print(f"[SERVER]: {msg}")

client.sendall(data) # Use sendall() to ensure complete data transmission

msg = client.recv(SIZE).decode(FORMAT)

print(f"[SERVER]: {msg}")

file.close()

client.close()

if \_name\_ == "\_main\_":

main()