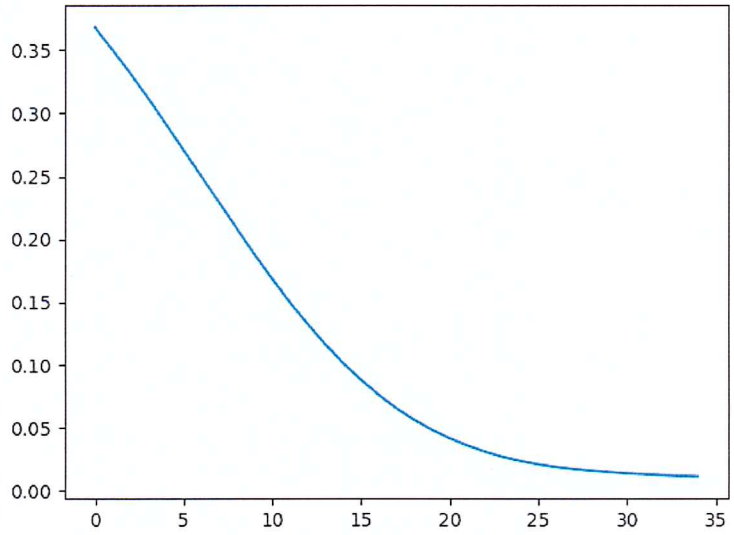
무선통신공학 과제

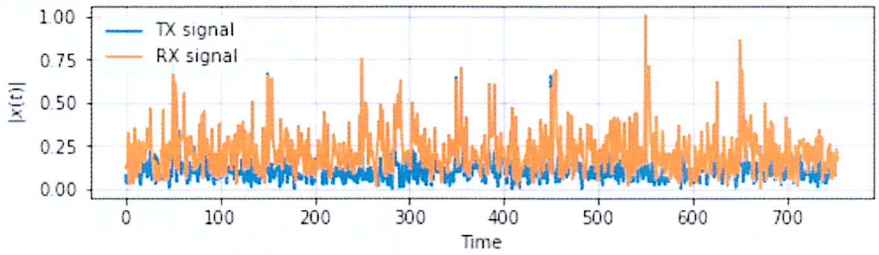
2021209854 홍성원

실행 결과화면

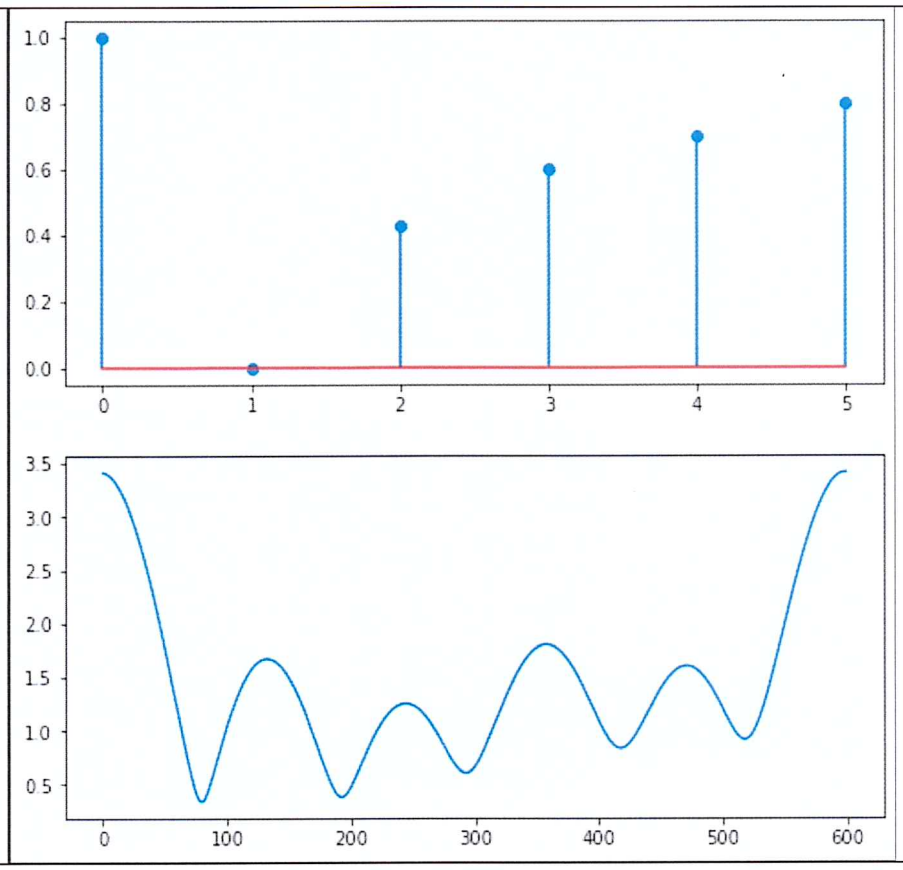
BER with respect to SNR



Signal (time)



Channel impulse response and frequency response (total carriers)



소스코드

```
import numpy as np
import matplotlib.pyplot as plt
import random
from scipy import interpolate


def SP(bits):
    return bits.reshape((len(dataCarriers), mu))

def mapping(bits):
    return np.array([mapping_table[tuple(b)] for b in bits])

def OFDM_symbol(QAM_payload):
    symbol = np.zeros(K, dtype=complex) # the overall K subcarriers
    symbol[pilotCarriers] = pilotvalue
    symbol[dataCarriers] = QAM_payload
    return symbol

def IFFT(OFDM_data):
    return np.fft.ifft(OFDM_data)

def addCP(OFDM_time):
```

```
    cp = OFDM_time[-CP:]
    return np.hstack([cp, OFDM_time])

def channel(signal):
    convolved = np.convolve(signal, channelResponse)
    signal_power = np.mean(abs(convolved**2))
    sigma2 = signal_power * 10**(-SNRdb/10)
    # print(f"RX signal power : {signal_power:.4f}. Noise power: {sigma2:.4f}")
    noise = np.sqrt(sigma2/2) *
(np.random.randn(*convolved.shape)+1j*np.random.randn(*convolved.shape))
    return convolved + noise

def removeCP(signal):
    return signal[CP:(CP+K)]

def FFT(OFDM_RX):
    return np.fft.fft(OFDM_RX)

def channelEstimate(OFDM_demod):
    pilots = OFDM_demod[pilotCarriers]
    Hest_at_pilots = pilots / pilotvalue

    Hest_abs = interpolate.interp1d(pilotCarriers, abs(Hest_at_pilots),
kind='linear')(allCarriers)
    Hest_phase = interpolate.interp1d(pilotCarriers, np.angle(Hest_at_pilots),
kind='linear')(allCarriers)
    Hest = Hest_abs * np.exp(1j * Hest_phase)

    # plt.plot(allCarriers, abs(H_exact), label="Correct Channel")
    # # plt.stem(pilotCarriers, abs(Hest_at_pilots), label='Pilot estimates')
    # plt.plot(allCarriers, abs(Hest), label="Estimated channel via interpolation")
    # plt.grid(True); plt.xlabel("Carrier index"); plt.ylabel("$|H(f)|$");
plt.legend(fontsize=10)
    # plt.ylim(0, 2)

    return Hest

def equalize(OFDM_demod, Hest):
    return OFDM_demod / Hest

def get_payload(equalized):
    return equalized[dataCarriers]

def Demapping(QAM):
    constellation = np.array([x for x in demapping_table.keys()])
    dists = abs(QAM.reshape((-1, 1)) - constellation.reshape((1, -1)))
    const_index = dists.argmin(axis = 1)
```

```
    hardDecision = constellation[const_index]
    return np.vstack([demapping_table[C] for C in hardDecision]), hardDecision

def PtoS(bits):
    return bits.reshape((-1, ))


K = 600 # number of OFDM subcarriers
CP = K // 4 # lenth of the cyclic prefix : 25 % of the block

P = 100 # number of pilot carriers per OFDM block
pilotvalue = 3 + 3j # The known value each pilot transmits


allCarriers = np.arange(K)
pilotCarriers = allCarriers[::K//100]
pilotCarriers = np.hstack([pilotCarriers, np.array([allCarriers[-1]])])
pilotCarriers
dataCarriers = np.delete(allCarriers, pilotCarriers)

allCarriers = np.arange(K) # indices of all subcarriers

pilotCarriers = allCarriers[::K//P]

# For convenience of channel estimation, let's make the last carriers also be a pilot
pilotCarriers = np.hstack([pilotCarriers, np.array([allCarriers[-1]])])
P += 1

# Data Carriers are all remaining carriers
dataCarriers = np.delete(allCarriers, pilotCarriers)

# print("allCarriers: ", allCarriers)

# plt.figure(figsize=(8, 2))
# plt.plot(pilotCarriers, np.zeros_like(pilotCarriers), 'bo', label='pilot')
# plt.plot(dataCarriers, np.zeros_like(dataCarriers), 'ro', label='data')
# plt.xlim([0, 30]); plt.ylim([-1, 2])
# plt.grid()
# plt.legend()
# plt.show()

mu = 4 # bits per symbol (i.e., 16QAM)
payloadBits_per_OFDM = len(dataCarriers) * mu # number of payload bits per OFDM
symbol

mapping_table = {
    (0, 0, 0, 0) : -3-3j,
```

```
    (0, 0, 0, 1) : -3-1j,
    (0, 0, 1, 0) : -3+3j,
    (0, 0, 1, 1) : -3+1j,
    (0, 1, 0, 0) : -1-3j,
    (0, 1, 0, 1) : -1-1j,
    (0, 1, 1, 0) : -1+3j,
    (0, 1, 1, 1) : -1+1j,
    (1, 0, 0, 0) :  3-3j,
    (1, 0, 0, 1) :  3-1j,
    (1, 0, 1, 0) :  3+3j,
    (1, 0, 1, 1) :  3+1j,
    (1, 1, 0, 0) :  1-3j,
    (1, 1, 0, 1) :  1-1j,
    (1, 1, 1, 0) :  1+3j,
    (1, 1, 1, 1) :  1+1j,
}
demapping_table = {v : k for k, v in mapping_table.items()}

# plt.figure()
# for b3 in [0, 1]:
#     for b2 in [0, 1]:
#         for b1 in [0, 1]:
#             for b0 in [0, 1]:
#                 B = (b3, b2, b1, b0)
#                 Q = mapping_table[B]
#                 plt.plot(Q.real, Q.imag, 'bo')
#                 plt.text(Q.real, Q.imag+0.2, "".join(str(x) for x in B), ha='center')
# plt.axis([-4, 4, -4, 4])
# plt.grid(True, linestyle="--")
# plt.show()

channelResponse = np.array([1, 0.3+0.3j, 0.7, 0.6, 0.1+0.6j])
H_exact = np.fft.fft(channelResponse, K)
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.stem(np.abs(channelResponse))
plt.subplot(2, 1, 2)
plt.plot(allCarriers, abs(H_exact))
plt.show()

SNR_dbs = np.arange(0, 35, 1)
BER_mc = np.zeros_like(SNR_dbs, dtype=np.float64)
for trial in range(10000):
    BER = []
    print("trial: ", trial+1, '\r')
    for SNRdb in SNR_dbs:
        bits = np.random.binomial(n=1, p=0.5, size=(payloadBits_per_OFDM, ))
```

```
        bits_SP = SP(bits)
        QAM = mapping(bits_SP)
        OFDM_data = OFDM_symbol(QAM)
        OFDM_time = IFFT(OFDM_data)
        OFDM_withCP = addCP(OFDM_time)
        OFDM_TX = OFDM_withCP
        OFDM_RX = channel(OFDM_TX)
        OFDM_RX_noCP = removeCP(OFDM_RX)
        OFDM_demod = FFT(OFDM_RX_noCP)
        Hest = channelEstimate(OFDM_demod=OFDM_demod)
        equalized_Hest = equalize(OFDM_demod, Hest)
        QAM_est = get_payload(equalized_Hest)
        PS_est, hardDecision = Demapping(QAM_est)
        bits_est = PtoS(PS_est)
        # print("obtained Bit error rate: ", np.sum(abs(bits-bits_est)) / len(bits))
        BER.append(np.sum(abs(bits-bits_est)) / len(bits))
    BER_mc += np.array(BER)

plt.figure()
plt.plot(BER_mc / 10000)
plt.show()
```