A project Report on

Performance Evaluation of Bayesian MMSE Channel Estimation on 16QAM Over Rayleigh Channel

Communication Signal Processing and Algorithms (TE61003)

Submitted by

Satyansh
Roll no: 25GS61R22

G.S. Sanyal School of Telecommunication

Indian Institute of Technology Kharagpur

Contents:

1.Project Objective:

The aim of this project is to analyze the Performance of 16QAM modulation over Rayleigh channel. We have used Minimum Mean Square Error Estimator to estimate the channel. The performance of 16QAM is studied on the basis of Symbol Error Rate(SER), Bit Error Rate (BER) and Normalized Mean Square Error (MSE) of the channel estimator. The project results are simulated using PYTHON.

1.1. Project details

The system model considered for this project is shown in Fig 1.1. It consists of 1 transmit antenna and 1 receive antenna. We send random data that has equal probability of 0's and 1's generated through Linear Congruential Generator. Data is modulated with 16-Quadrature Amplitude Modulation(QAM) with each bit having unit energy. These modulated symbols are arranged into packets along with pilots and are transmitted over a Rayleigh Channel(By taking 2 normally distributed samples on complex plane). MMSE estimator is used to estimate the channel "h" with the help of known sequences called as pilots(preamble). Then received symbols are compensated with "h" using Zero Forcing equalization technique. QAM demodulation is performed and then BER vs SNR, SER vs SNR and MSE vs SNR is plotted.

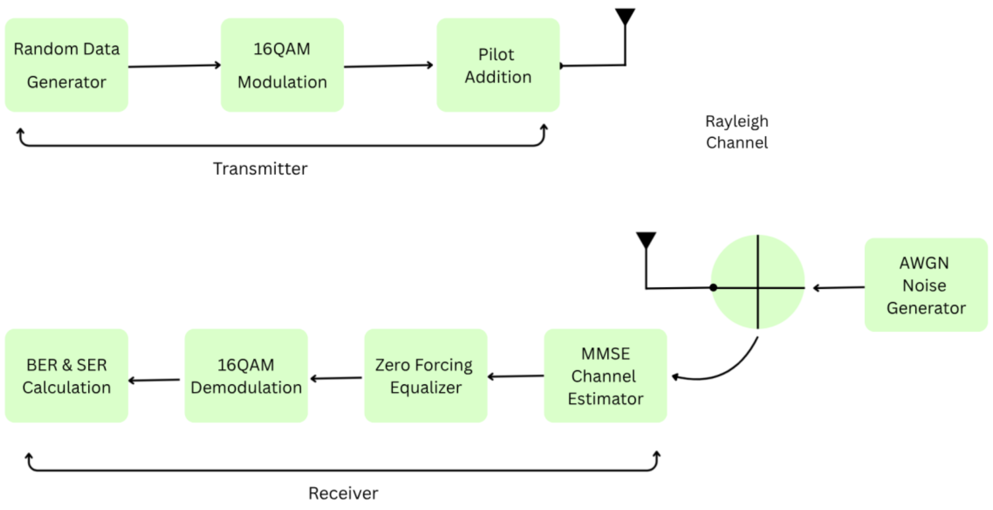

Fig.1.1) System Model

2. Introduction

2.1 IEEE Standard 802.11n pilots for system

The wireless system is a packet-switched system with random access protocol. This essentially means that the receiver has no prior knowledge about the arrival time of the packet. To perform synchronization of the system, a packet is preceded with a known sequence, i.e., a preamble.

The preamble is carefully designed to provide sufficient information for the receiver to perform packet detection, frequency offset estimation, symbol timing acquisition, and channel estimation.

We have considered the below sequence which is a wifi training Sequence, it is repeated 4 times.

L = {1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,0,1,-1,
-1,1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,1}

2.2 Minimum Mean Square Error (MMSE) Estimator:

In modern wireless communication systems, multipath propagation causes signal fading, leading to significant degradation in performance. Among various fading models, the **Rayleigh fading channel** is commonly used to model urban environments where no line-of-sight component exists between the transmitter and receiver.
To mitigate the effects of channel fading, accurate channel estimation is essential. This study implements a **Bayesian Minimum Mean Square Error (MMSE)** estimator to estimate the channel response using known **pilot symbols**.

The estimated channel is then used to **equalize** the received data before demodulation.

The simulation evaluates the system performance in terms of:

- **Bit Error Rate (BER)**

- **Symbol Error Rate (SER)**

- **Normalized Mean Square Error (nMMSE)**

Both **equalized** and **unequalized** data are compared to assess the effectiveness of MMSE-based channel estimation.

The baseband equivalent model of the communication system is expressed as:

$$\mathbf{Y = hx + n}$$

Where,

y : received symbol
h: complex channel gain (Rayleigh distributed)
x : transmitted symbol (pilot or data)
n : complex Additive White Gaussian Noise (AWGN)

The MMSE estimator minimizes the expected squared estimation error:

$$\hat{h}_{MMSE} = \arg\min_{\hat{h}} \mathbb{E}[|h - \hat{h}|^2 | y]$$

For a single-tap flat fading channel and known pilot symbols, the closed-form MMSE estimate is:

$$\hat{h}_{MMSE} = \frac{\sigma_h^2 X^H y}{|X|^2 \sigma_h^2 + \sigma_n^2}$$

Where,

X: pilot symbol matrix (diagonal or vector form)
y: received pilot vector
σ^2_h : variance of fading channel (usually 1)
σ^2_n: noise variance

This formula arises from the joint Gaussian nature of h and y.

2.3 Zero Forcing Equalizer:

Once h_{MMSE} is obtained, the received data symbols are equalized by:

$$\tilde{x} = \frac{y_{\text{data}}}{\hat{h}_{MMSE}}$$

This is a **1-tap complex equalizer** (scalar division) also known as a **zero-forcing (ZF) equalization** that uses the **MMSE channel estimate** rather than the true channel.

3. Simulation:

The simulation is done for transmission of 4 packets. Each packet has 1000 info bits and each trial is repeated 10 times to get better results. The bit error rate for different scenarios are plotted and the effect of no estimation is depicted and discussed.
BER, SER and Mean square error is plotted for different SNR values.

3.1 Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from scipy.special import erfc


# --------------------------
# Parameters
# --------------------------
num_bits = 4000
data_per_frame = 1000
num_trials = 10
a, c, m = 1664525, 1013904223, 2**32
seed = 42
snr_db_range = np.arange(0, 22, 1)
np.random.seed(0)

# Pilot sequence
pilot_symbols = np.array([
   1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-
1,1,1,1,1,
   1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-
1,1,1,1,1,
   1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-
1,1,1,1,1,
   1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-
1,1,1,1,1
], dtype=np.complex128)
Lp = len(pilot_symbols)
```

```
# --------------------------
# LCG Bitstream
# --------------------------
lcg = [seed]
for _ in range(num_bits - 1):
    lcg.append((a * lcg[-1] + c) % m)
lcg_normalized = np.array(lcg, dtype=np.float64) / m
bitstream = (lcg_normalized > 0.5).astype(int)

# --------------------------
# 16-QAM Mapping
# --------------------------
symbols_bits = [bitstream[i:i+4] for i in range(0, len(bitstream), 4)]
qam_symbols = []
valid_symbol_bits = []
for b in symbols_bits:
    if len(b) < 4:
        continue
    b0, b1, b2, b3 = b
    I = (1 - 2*b0) * (2 - (1 - 2*b2))
    Q = (1 - 2*b1) * (2 - (1 - 2*b3))
    q = (1 / np.sqrt(10)) * (I + 1j * Q)
    qam_symbols.append(q)
    valid_symbol_bits.append([int(b0), int(b1), int(b2), int(b3)])
qam_symbols = np.array(qam_symbols, dtype=np.complex128)
valid_symbol_bits = np.array(valid_symbol_bits, dtype=int)
num_symbols = len(qam_symbols)
signal_power = np.mean(np.abs(qam_symbols)**2)

# --------------------------
# Demodulation
# --------------------------
def demodulate(symbol):
    I = np.real(symbol) * np.sqrt(10)
    Q = np.imag(symbol) * np.sqrt(10)
    b0 = 0 if I > 0 else 1
    b2 = 0 if abs(I) < 2 else 1
    b1 = 0 if Q > 0 else 1
    b3 = 0 if abs(Q) < 2 else 1
    return [b0, b1, b2, b3]

# --------------------------
# Frame Setup
# --------------------------
num_frames = int(np.ceil(num_symbols / data_per_frame))
sigma_h2 = 1.0

# --------------------------
# Storage
# --------------------------
ber_uneq, ber_eq = [], []
ser_uneq, ser_eq = [], []
norm_mmse_list = []
```

```
for snr_db in snr_db_range:
    snr_linear = 10**(snr_db / 10)
    noise_power = signal_power / snr_linear
    sigma_n2 = noise_power
    noise_std = np.sqrt(noise_power / 2)

    ber_uneq_trials, ber_eq_trials = [], []
    ser_uneq_trials, ser_eq_trials = [], []
    mmse_trials = []

    for _ in range(num_trials):
        bit_errors_uneq = 0
        bit_errors_eq = 0
        sym_errors_uneq = 0
        sym_errors_eq = 0
        total_data_symbols = 0
        total_mse = 0.0
        symbol_idx = 0

        for frame in range(num_frames):
            Nd = min(data_per_frame, num_symbols - symbol_idx)
            if Nd <= 0:
                break
            s_frame = qam_symbols[symbol_idx : symbol_idx + Nd]
            bits_frame = valid_symbol_bits[symbol_idx : symbol_idx + Nd]

            h = (np.random.randn() + 1j*np.random.randn()) * np.sqrt(sigma_h2/2)

            n_p = noise_std * (np.random.randn(Lp) + 1j*np.random.randn(Lp))
            y_p = h * pilot_symbols + n_p

            p = pilot_symbols
            denom = sigma_h2 * np.vdot(p, p) + sigma_n2
            numer = sigma_h2 * np.vdot(p, y_p)
            h_hat = numer / denom

            mse_frame = np.abs(h - h_hat)**2
            total_mse += mse_frame
            total_data_symbols += Nd

            n_data = noise_std * (np.random.randn(Nd) + 1j*np.random.randn(Nd))
            y_data = h * s_frame + n_data

            est_bits_uneq = []
            sym_err_uneq = 0
            for i in range(Nd):
                est_b = demodulate(y_data[i])
                est_bits_uneq.extend(est_b)
                if est_b != list(bits_frame[i]):
                    sym_err_uneq += 1

            est_bits_eq = []
            sym_err_eq = 0
            s_hat_eq = y_data / h_hat
            for i in range(Nd):
                est_b = demodulate(s_hat_eq[i])
                est_bits_eq.extend(est_b)
                if est_b != list(bits_frame[i]):
                    sym_err_eq += 1

            bits_frame_flat = bits_frame.flatten().tolist()
            est_bits_uneq = np.array(est_bits_uneq, dtype=int)
            est_bits_eq = np.array(est_bits_eq, dtype=int)
            bits_frame_arr = np.array(bits_frame_flat, dtype=int)

            bit_errors_uneq += np.sum(bits_frame_arr != est_bits_uneq)
            bit_errors_eq += np.sum(bits_frame_arr != est_bits_eq)
            sym_errors_uneq += sym_err_uneq
            sym_errors_eq += sym_err_eq
            symbol_idx += Nd

        ber_uneq_trials.append(bit_errors_uneq / (4 * num_symbols))
        ber_eq_trials.append(bit_errors_eq / (4 * num_symbols))
        ser_uneq_trials.append(sym_errors_uneq / num_symbols)
        ser_eq_trials.append(sym_errors_eq / num_symbols)
        mmse_trials.append(total_mse / (sigma_h2 * num_frames))
    ber_uneq.append(np.mean(ber_uneq_trials))
    ber_eq.append(np.mean(ber_eq_trials))
    ser_uneq.append(np.mean(ser_uneq_trials))
    ser_eq.append(np.mean(ser_eq_trials))
    norm_mmse_list.append(np.mean(mmse_trials))


# --------------------------
# Theoretical AWGN Curves (16-QAM)
# --------------------------
snr_linear = 10**(snr_db_range / 10)
M = 16
k = np.log2(M)
EsN0 = snr_linear
EbN0 = EsN0 / k

# Theoretical Symbol Error Rate for square 16-QAM in
AWGN
ser_theoretical = 3/2 * erfc(np.sqrt(0.1 * snr_linear))
ser_theoretical = np.minimum(ser_theoretical, 1)

# Theoretical BER (approximate)
ber_theoretical = ser_theoretical / k


# --------------------------
# Plot Results
# --------------------------

plt.style.use('ggplot')

# Create a 2x2 grid: top row for BER and SER, bottom row
for MSE spanning both columns
fig = plt.figure(figsize=(12, 7))
gs = gridspec.GridSpec(2, 2, height_ratios=[1, 1.2])

# Top row: BER and SER
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])

# Bottom row: MSE spans both columns
ax3 = fig.add_subplot(gs[1, :])

# BER Plot
x1.semilogy(snr_db_range, ber_uneq, 'r-o',
label='Unequalized BER')
ax1.semilogy(snr_db_range, ber_eq, 'b-s', label='Equalized
BER')
ax1.semilogy(snr_db_range, ber_theoretical, 'k--',
label='AWGN Theoretical BER')
ax1.set_xlabel('SNR (dB)')
ax1.set_ylabel('Bit Error Rate')
ax1.set_title('BER vs SNR (16-QAM)')
ax1.grid(True, which='both', linestyle=':')
ax1.legend()

# SER Plot
ax2.semilogy(snr_db_range, ser_uneq, 'r-o',
label='Unequalized SER')
ax2.semilogy(snr_db_range, ser_eq, 'b-s', label='Equalized
SER')
ax2.semilogy(snr_db_range, ser_theoretical, 'k--',
label='AWGN Theoretical SER')
ax2.set_xlabel('SNR (dB)')
ax2.set_ylabel('Symbol Error Rate')
ax2.set_title('SER vs SNR (16-QAM)')
ax2.grid(True, which='both', linestyle=':')
ax2.legend()

# MSE Plot
ax3.semilogy(snr_db_range, norm_mmse_list, 'm-o',
label='Normalized MMSE (E[|h-h_hat|^2]/E[|h|^2])')
ax3.set_xlabel('SNR (dB)')
ax3.set_ylabel('Normalized MMSE')
ax3.set_title('MSE vs SNR (MMSE Estimation)')
ax3.grid(True, which='both', linestyle=':')
ax3.legend()

plt.tight_layout()
plt.show()
```

3.2 Simulation results:

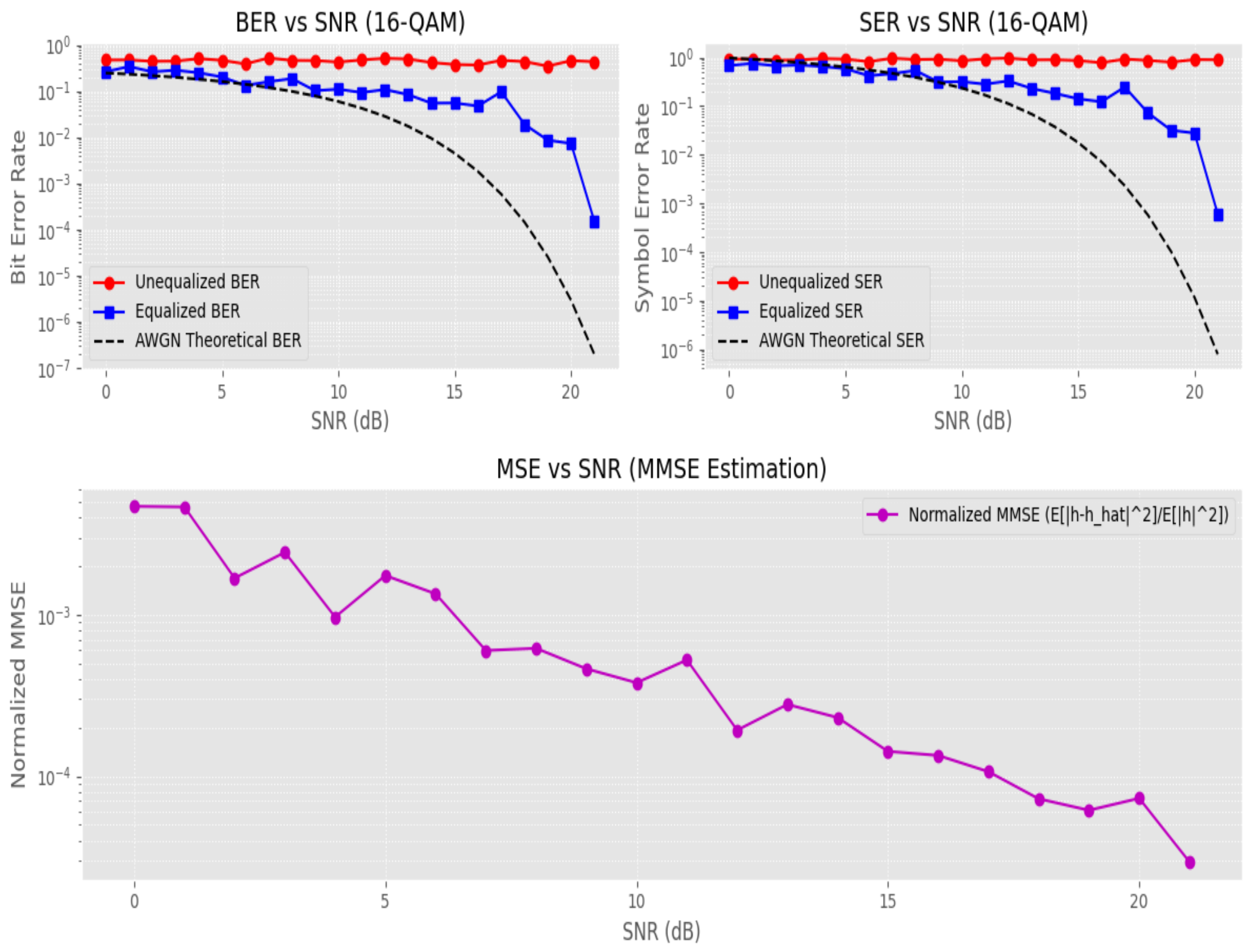

Fig.3.1) BER, SER and AWGN Theoretical v/s SNR

1.) BER v/s SNR plot (RED -> Unequalised, BLUE -> Equalised)

2.) SER v/s SNR plot (RED -> Unequalised, BLUE -> Equalised)

3.) MSE v/s SNR (Normalised MMSE)

4. Discussion :

• The **BER plot** shows the probability of bit errors as a function of the signal-to-noise ratio (SNR) for both **equalized** and **unequalized** cases.
As the **SNR increases**, the BER for the **equalized signal** decreases significantly faster than for the unequalized signal. This demonstrates the effectiveness of channel equalization in compensating for the Rayleigh fading distortion. The **unequalized BER** curve tends to flatten out at a higher error floor because fading causes deep amplitude nulls that cannot be corrected without equalization.

Beyond **~18–20 dB**, the BER for the equalized data typically reaches below 10^{-3} - 10^{-4} , indicating reliable symbol recovery.

• The **SER plot** follows a similar trend as BER but is generally higher in absolute value because one symbol error can cause multiple bit errors.

• The **normalized MMSE plot** displays the average relative mean-square error between the true channel coefficient “h” and its MMSE estimate. The linear decay of normalized MMSE with SNR confirms that the Bayesian MMSE estimator effectively exploits pilot energy and noise statistics. This improved channel knowledge directly explains the reduction in BER and SER at high SNR.

5. Conclusion:

The Bayesian MMSE estimator for estimating Rayleigh channel coefficients is designed and the effect of Rayleigh channel on the communication system is compared with the AWGN channel in the 16QAM modulation scheme.

The mean square error(MSE) is observed for various values of SNR. From the results, we can conclude that estimation and equalization of unknown parameters is must before the detection of data, and better the estimator performance better the system performance for successful communication.

6. References:

Fundamentals of Statistical Signal Processing: Estimation Theory, Steven M.kay, University