Design of QPSK Receivers and SER Analysis

Experiment - 3

1 Aim

To design a discrete time QPSK communication receiver, and analyze the SER performance.

2 Theory

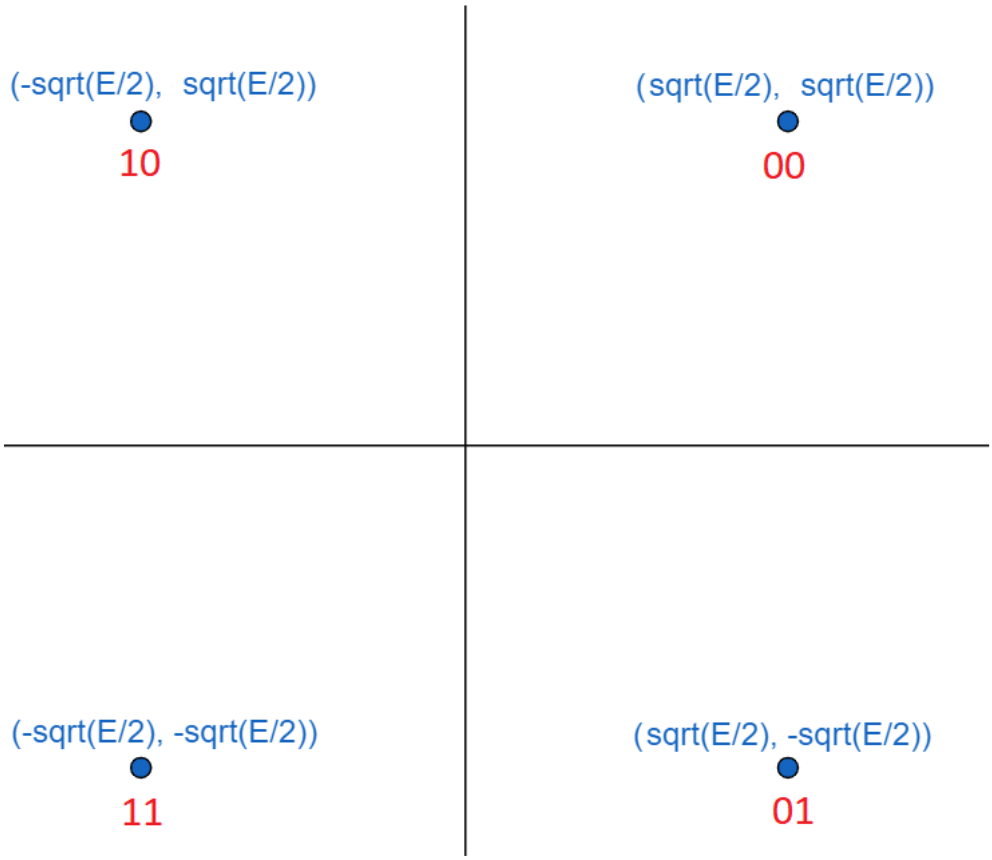


Fig.1 Constellation Map

2.1 ML Rule

Since every symbol is equally likely, the ML rule is based on the shortest euclidean distance:

$$arg\,min\,||y - s_i||^2,\ i = 0, 1, 2, 3$$

2.1 SER

Let E be the error bit and s_0, s_1, s_2, s_3 be the possible symbols. Let s_i be the i^{th} symbol and since all symbols are equally likely:

$$P(E = 1) = \frac{1}{4}\sum_{i=0}^{3} P(E = 1 \mid s_i) \tag{1}$$

Owing to the symmetry of the constellation,

$$P(E = 1 \mid s_0) = P(E = 1 \mid s_1) = P(E = 1 \mid s_2) = P(E = 1 \mid s_3) \tag{2}$$

From (1) and (2),

$$P(E = 1) = P(E = 1 \mid s_0) = P(E = 1 \mid s_1) = P(E = 1 \mid s_2) = P(E = 1 \mid s_3) \tag{3}$$

Let $S_k = [S_{1k}, S_{2k}]$ represent the symbol transmitted and $N_k = [N_{1k}, N_{2k}]$ be the noise introudced by the AWGN channel such that $Y_k = [Y_{1k}, Y_{2k}] = N_k + S_k$. Then,

$$
\begin{aligned}
P(E=1) &= P(E=1 \mid s_0) \\
&= P(Y_{1k} < 0, Y_{2k} < 0) \\
&= 1 - P(Y_{1k} > 0, Y_{2k} > 0) \\
&= 1 - P(S_{1k} + N_{1k} > 0, S_{2k} + N_{2k} > 0) \\
&= 1 - P\left(\frac{\sqrt{E_s}}{2} + N_{1k} > 0, \frac{\sqrt{E_s}}{2} + N_{2k} > 0\right) \\
&= 1 - P\left(N_{1k} > -\sqrt{\frac{E_s}{2}}, N_{2k} > -\sqrt{\frac{E_s}{2}}\right) \\
&= 1 - P\left(N_{1k} > -\sqrt{\frac{E_s}{2}}\right) P\left(N_{2k} > -\sqrt{\frac{E_s}{2}}\right) \\
&= 1 - P\left(\frac{N_{1k}}{\sigma} > -\sqrt{\frac{E_s}{2\sigma^2}}\right) P\left(\frac{N_{2k}}{\sigma} > -\sqrt{\frac{E_s}{2\sigma^2}}\right) \\
&= 1 - Q^2\left(-\sqrt{\frac{E_s}{2\sigma^2}}\right) \\
&= 1 - \left(1 - Q\left(\sqrt{\frac{E_s}{2\sigma^2}}\right)\right)^2 \\
&= 2Q\left(\sqrt{\frac{E_s}{N_0}}\right) - Q^2\left(\sqrt{\frac{E_s}{N_0}}\right) \\
\therefore \boxed{P_e = P(E=1) = 2Q\left(\sqrt{\frac{E_s}{N_0}}\right) - Q^2\left(\sqrt{\frac{E_s}{N_0}}\right)} \quad (4)
\end{aligned}
$$

3 Design

```
Input: NO_OF_BITS, SB_N0_DB
Output: SER

map([bit1, bit2]) = {
    00: [1/sqrt(2), 1/sqrt(2)],
    01: [1/sqrt(2), -1/sqrt(2)],
    10: [-1/sqrt(2), 1/sqrt(2)],
    11: [-1/sqrt(2), -1/sqrt(2)]
}

ML([bit1, bit2]) = {
    00: bit1 > 0 and bit2 > 0
    01: bit1 > 0 and bit2 < 0
    10: bit1 < 0 abd bit2 > 0
    11: bit1 < 0 and bit2 < 0
}

Sk = map(Bk.group(2))

for SB_N0 in SB_N0_DB
    Nk = AWGN(SB_N0, 2D)
    Yk = Sk + Nk
    sHat = ML(Yk)
    unchangedSymbols = count(sHat === Sk)
    SER = 1 - (unchangedSymbols/NO_OF_BITS)

plot(SER, SER_THEORETICAL)
```

The time complexity of the algorithm is $O(N^2)$.

4 JavaScript Code

QPSK.js

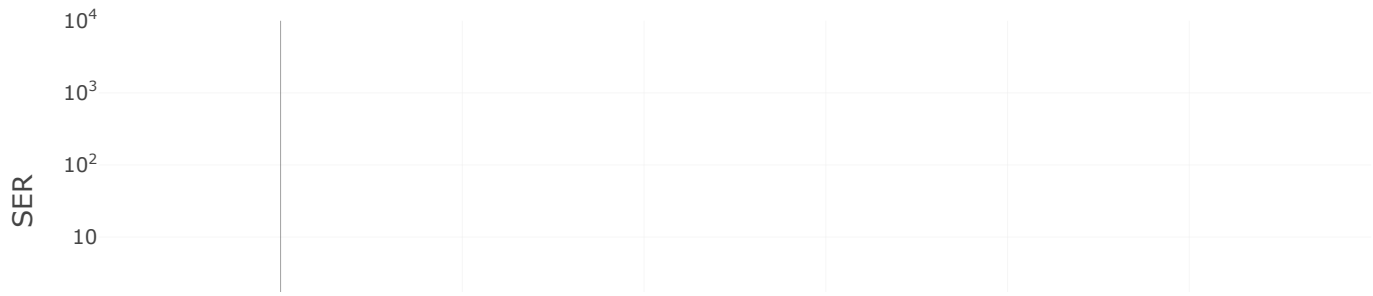
```
import SimulationHelpers from '../helpers/SimulationHelpers.js';

const S = new SimulationHelpers();

export default function QPSK(SB_N0_DB, NO_OF_BITS) {
    const NO_OF_SYMBOLS = NO_OF_BITS / 2;
    const SER = new Array(SB_N0_DB.length);
    const SER_THEORETICAL = S.getTheoreticalSerQpsk(SB_N0_DB);
    const Bk = S.randi([0, 1], NO_OF_BITS); // message
    const Sk = Bk.reduce((acc, bit, i) => { // modulation
        if (i % 2 === 0) {
            const dibit = [bit === 0 ? Math.SQRT1_2 : -Math.SQRT1_2];
            acc.push(dibit);
        } else {
            acc[acc.length - 1].push(bit === 0 ? Math.SQRT1_2 : -Math.SQRT1_2);
        }
        return acc;
    }, []);
    for (let i = 0; i < SB_N0_DB.length; i += 1) {
        const Nk = S.getAWGN(SB_N0_DB[i], [NO_OF_SYMBOLS, 2]); // AWGN noise
        const Yk = new Array(NO_OF_SYMBOLS).fill(0).map((_, j) => S.sum(Sk[j], Nk[j]));
        const sHat = Yk.map(([bit1, bit2]) => {
            if (bit1 >= 0 && bit2 >= 0) return [0, 0];
            if (bit1 <= 0 && bit2 >= 0) return [1, 0];
            if (bit1 <= 0 && bit2 <= 0) return [1, 1];
            return [0, 1];
        }); // ML decision rule
        const unchangedSymbols = sHat.reduce((acc, [bit1, bit2], j) => {
            // eslint-disable-next-line no-param-reassign
            if (bit1 === Bk[2 * j] && bit2 === Bk[2 * j + 1]) acc += 1;
            return acc;
        }, 0);
        SER[i] = 1 - (unchangedSymbols / NO_OF_SYMBOLS);
    }
    return [SER, SER_THEORETICAL];
}
```

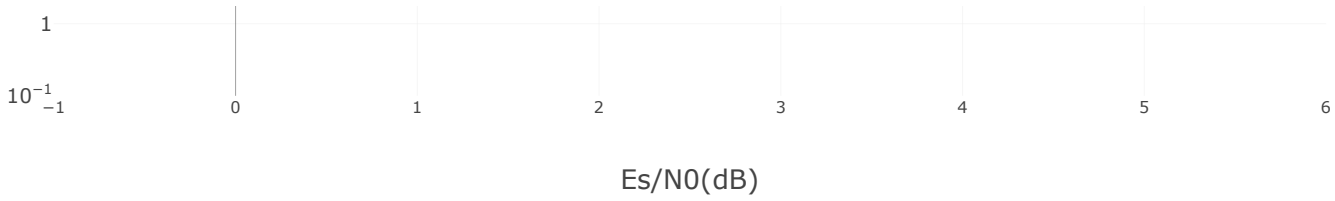
SimulationHelpers.js

```
const { jStat } = window;

export default class SimulationHelpers {
    constructor() {
        this.sum = (arr1, arr2) => arr1.map((num, i) => num + arr2[i]);

        this.linspace = (start, stop, diff) => {
            const entries = [];
            let entry = start;
            while (entry <= stop) {
                entries.push(entry);
                entry += diff;
            }
            return entries;
        };

        this.qfunc = (arg) => 0.5 * jStat.erfc(arg / Math.SQRT2);

        this.getTheoreticalBerBpsk = (EB_N0_DB) => EB_N0_DB
            .map((EB_N0) => this.qfunc(Math.sqrt(2 * (10 ** (EB_N0 / 10)))));
```

```
        this.getTheoreticalSerQpsk = (SB_N0_DB) => SB_N0_DB
            .map((SB_N0) => 2 * this.qfunc(Math.sqrt(10 ** (SB_N0 / 10)))
                - this.qfunc(Math.sqrt(10 ** (SB_N0 / 10))) ** 2);

        this.getTheoreticalBerBfsk = (EB_N0_DB) => EB_N0_DB
            .map((EB_N0) => this.qfunc(Math.sqrt(10 ** (EB_N0 / 10))));

        this.getTheoreticalSerQam8 = (SB_N0_DB, Es) => SB_N0_DB
            .map((SB_N0) => 2.5 * this.qfunc(Math.sqrt(((10 ** (SB_N0 / 10)) * Es) / 3))
                - 1.5 * (this.qfunc(Math.sqrt(((10 ** (SB_N0 / 10)) * Es) / 3))) ** 2);

        this.getTheoreticalSerMpsk = (SB_N0_DB, M) => SB_N0_DB
            .map((SB_N0) => 2 * this
                .qfunc(Math.sqrt(2 * 10 ** (SB_N0 / 10)) * Math.sin(Math.PI / M)));

        this.randi = ([min, max], count) => {
            const integers = [];
            // eslint-disable-next-line no-plusplus, no-param-reassign
            while (count--) {
                integers.push(Math.floor(min + ((max - min + 1) * Math.random())));
            }
            return integers;
        };

        this.randn = (count) => new Array(count).fill(0).map(() => jStat.normal.sample(0, 1));

        this.getAWGN = (SB_N0_DB, [rows, cols]) => {
            const awgn = this.randn(rows * cols)
                .map((N) => N / Math.sqrt(2 * (10 ** (SB_N0_DB / 10))));
            if (cols === 1) return awgn;
            return awgn.reduce((acc, N, j) => {
                if (j % cols === 0) {
                    const noise = [N];
                    acc.push(noise);
                } else {
                    acc[acc.length - 1].push(N);
                }
                return acc;
            }, []);
        };

        this.dec2bin = (number, length) => {
            let binaryString = '';
            for (let i = 0; i < length - 1; i += 1) binaryString += '0';
            binaryString += number.toString(2);
            return binaryString.slice(-length);
        };
    }
}
```

5 Results and Inferences





Simulate Save Simulation Data Save Theoretical Data

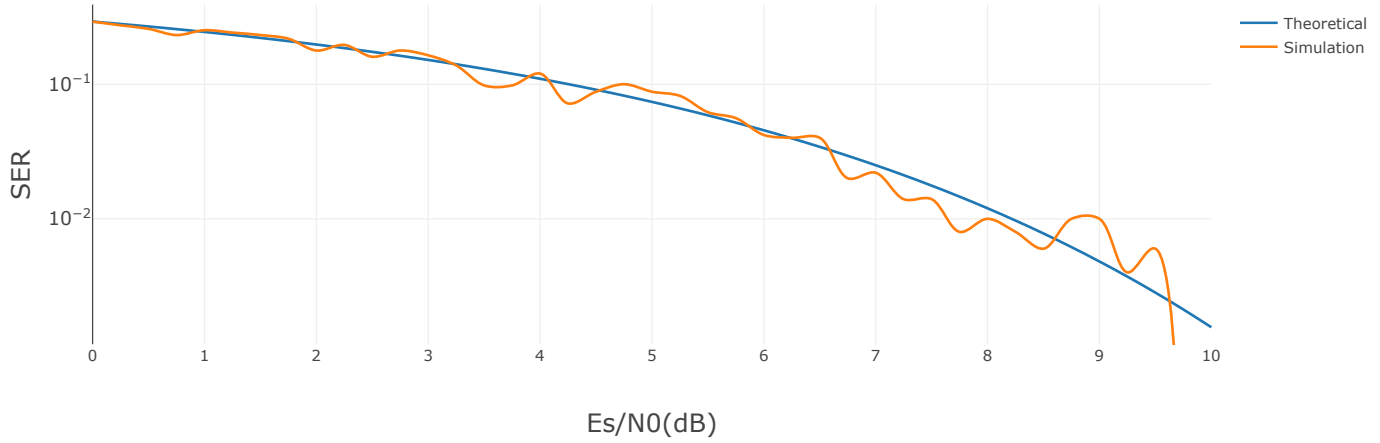


Fig.2 Simulation result for QPSK with 10^3 bits

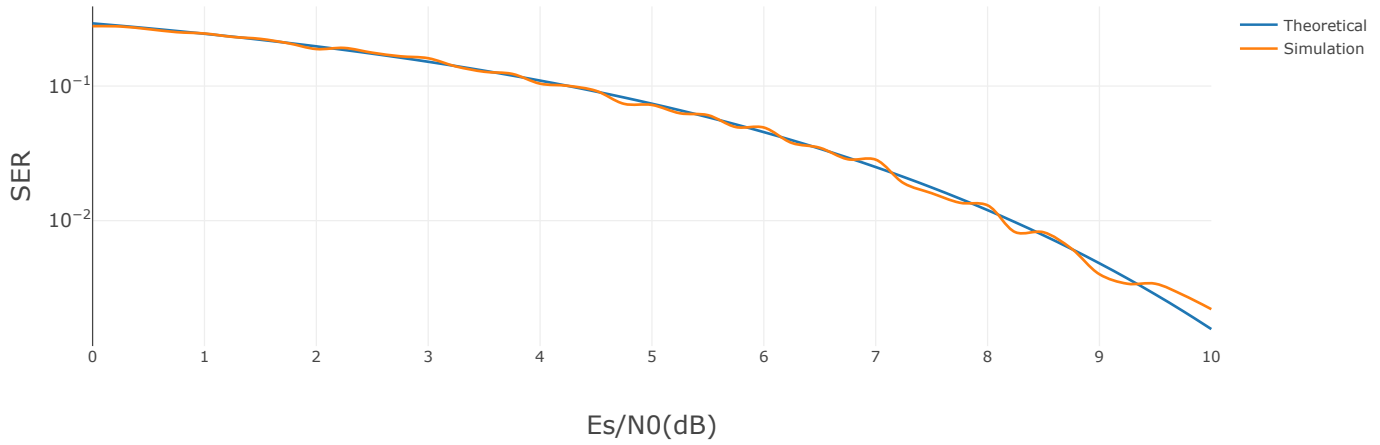
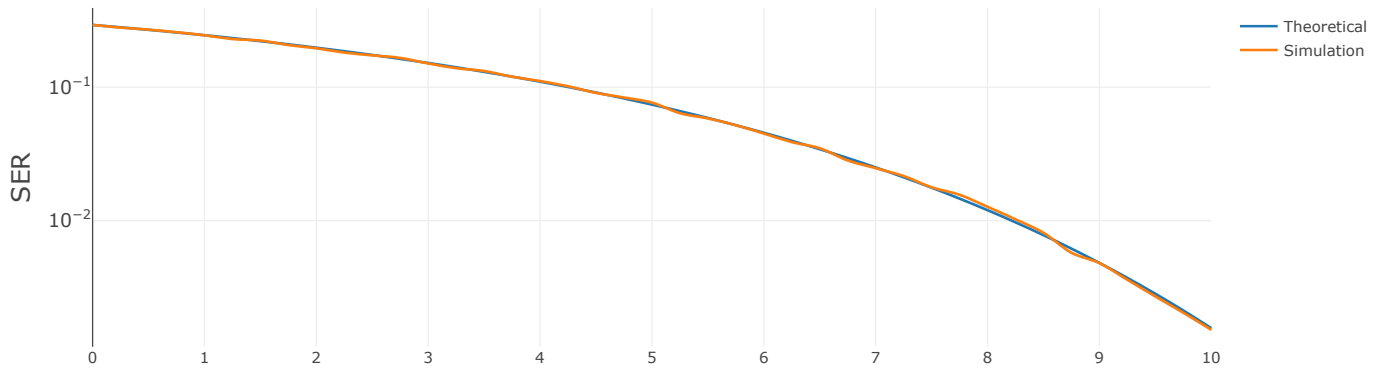


Fig.3 Simulation result for QPSK with 10^4 bits



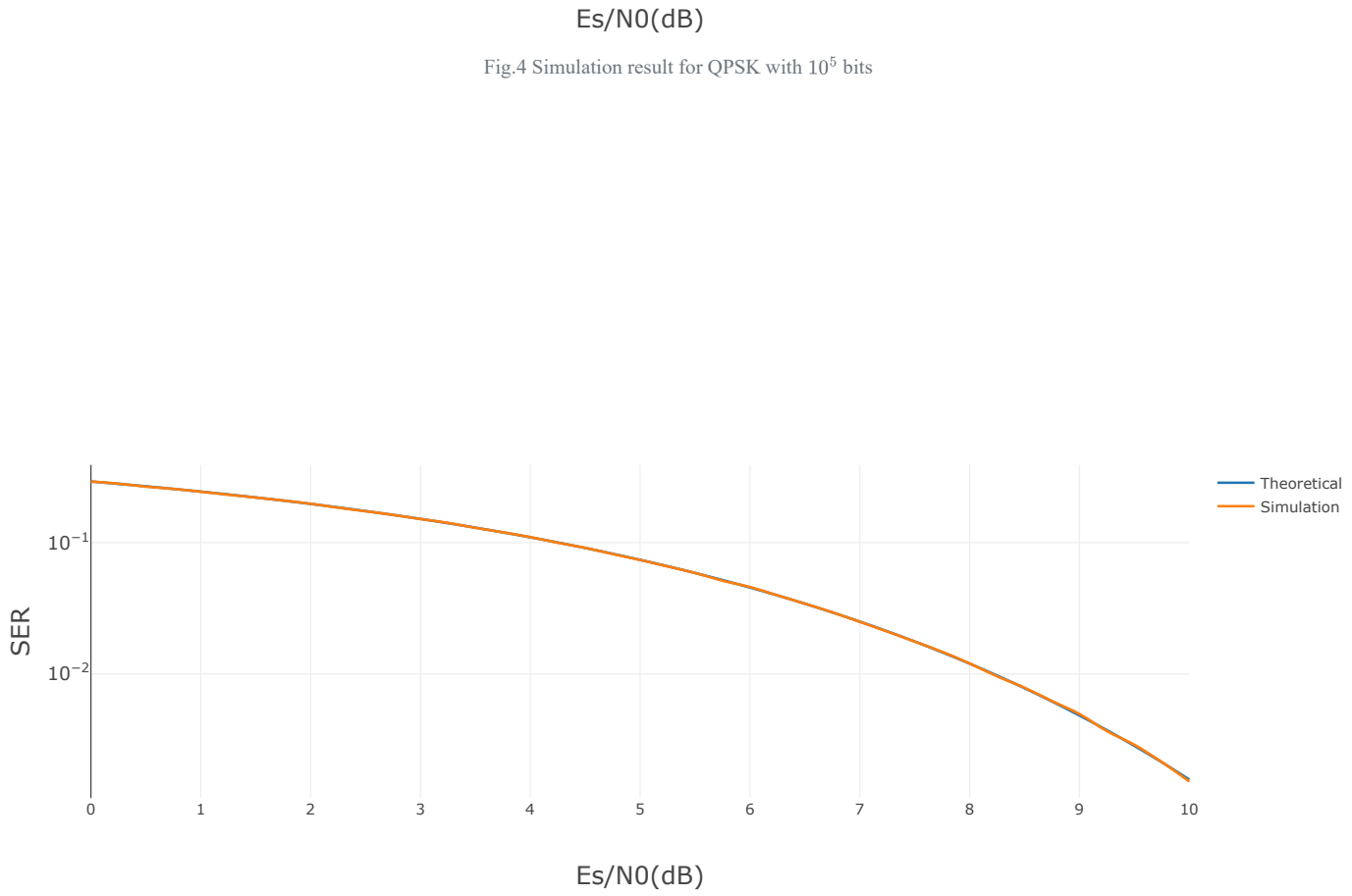


Fig.4 Simulation result for QPSK with 10^5 bits

Fig.5 Simulation result for QPSK with 10^6 bits

From the above figures, it can be seen that the simulated P_e and theoretical P_e match better with increase in the number of bits. This is because, to get confidence in the simulated results, there must be sufficient number of bit errors. For example, to get a bit error rate of 10^{-5}, one needs to send at least 10^6 bits. Also, it can be seen that the P_e reduces gradually with increase in $\frac{E_s}{N_0}$, the SNR per symbol. This is due to the fact that as the SNR per symbol increases, the signal becomes less affected by the presence of noise, thus reducing errors in ML detection. Finally, another important thing to be noted is that the simulated curve is not only affected by the symbol errors, but is also affected by floating point errors.