Design of QAM Receivers and SER Analysis

Experiment - 5

1 Aim

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

2 Theory

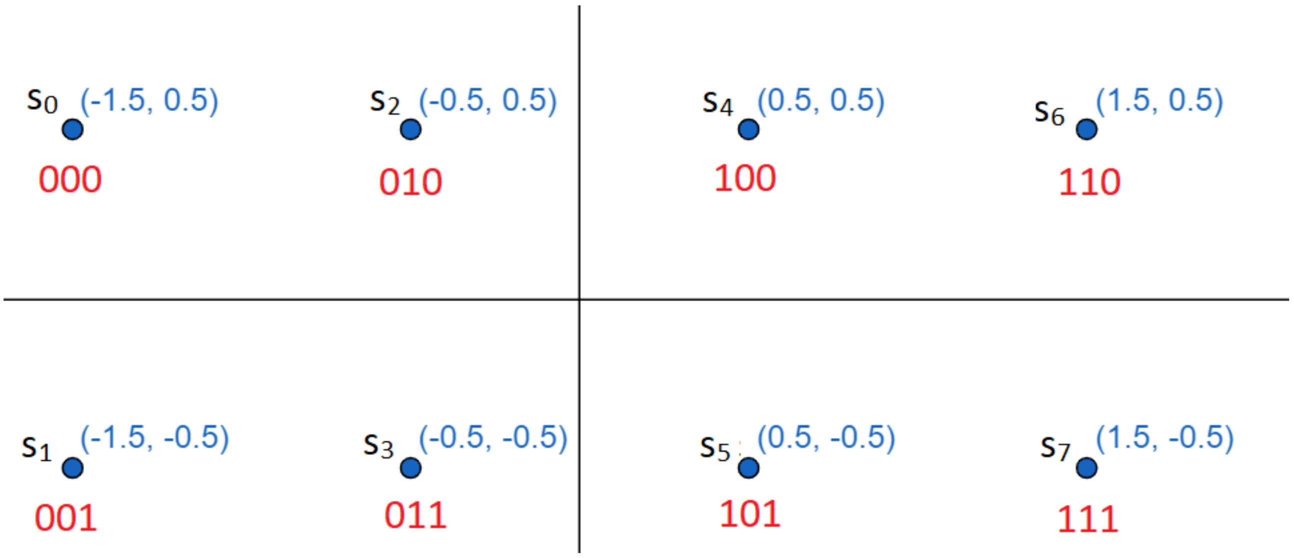


Fig.1 Constellation Map

2.1 ML Rule

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

$$arg\,min\,||y - s_i||^2,\ i = 0, 1, 2 \dots, 7$$

2.2 SER

Let d be the distance between the points and E be the error bit. Let s_i be the i^{th} symbol and since all symbols are equally likely:

$$P(E = 1) = \sum_{i=1}^{7} P(s_i)P(E \mid s_i) = \frac{1}{8}\sum_{i=1}^{7} P(E \mid s_i) \tag{1}$$

Owing to the symmetry of the constellation, we can write:

$$\begin{aligned} P(E \mid s_0) = P(E \mid s_1) = P(E \mid s_6) = P(E \mid s_7) \\ P(E \mid s_2) = P(E \mid s_3) = P(E \mid s_4) = P(E \mid s_5) \end{aligned} \tag{2}$$

Thus, (1) reduces to:

$$P(E = 1) = \frac{P(E \mid s_1) + P(E \mid s_3)}{2} \tag{3}$$

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,

$$\begin{aligned} P(E \mid s_1) &= P(\{Y_{1k} > -d\} \cap \{P(Y_{2k} > 0)\}) \\ &= 1 - P(\{Y_{1k} < -d\} \cap \{P(Y_{2k} < 0)\}) \\ &= 1 - P(Y_{1k} < -d)P(Y_{2k} < 0) \end{aligned}$$

$$
\begin{aligned}
&= 1 - P(S_{1k} + N_{1k} < -d)P(S_{2k} + N_{2k} < 0) \\
&= 1 - P(N_{1k} - 3d/2 < -d)P(N_{2k} - d/2 < 0) \\
&= 1 - P(N_{1k} < d/2)P(N_{2k} < d/2) \\
&= 1 - (1 - P(N_{1k} > d/2))^2 \\
&= 1 - \left(1 - P\left(\frac{N_{1k}}{\sigma} > \frac{d}{2\sigma}\right)\right)^2 \\
&= 1 - \left(1 - Q\left(\frac{d}{2\sigma}\right)\right)^2 \\
&= 2Q\left(\frac{d}{2\sigma}\right) - Q^2\left(\frac{d}{2\sigma}\right) \qquad (4)
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
P(E \mid s_3) &= P(\{Y_{1k} < -d, Y_{1k} > 0\} \cap \{P(Y_{2k} > 0)\}) \\
&= 1 - P(\{-d < Y_{1k} < 0\} \cap \{P(Y_{2k} < 0)\}) \\
&= 1 - P(-d < Y_{1k} < 0)P(Y_{2k} < 0) \\
&= 1 - P(-d < S_{1k} + N_{1k} < 0)P(S_{2k} + N_{2k} < 0) \\
&= 1 - P(-d < N_{1k} - d/2 < 0)P(N_{2k} - d/2 < 0) \\
&= 1 - P(-d/2 < N_{1k} < d/2)P(N_{2k} < d/2) \\
&= 1 - P\left(-\frac{d}{2\sigma} < N_{1k} < \frac{d}{2\sigma}\right)P\left(\frac{N_{2k}}{\sigma} < \frac{d}{2\sigma}\right) \\
&= 1 - P\left(-\frac{d}{2\sigma} < N_{1k} < \frac{d}{2\sigma}\right)P\left(\frac{N_{2k}}{\sigma} < \frac{d}{2\sigma}\right) \\
&= 1 - \left(1 - P\left(\frac{d}{2\sigma} > N_{1k} > -\frac{d}{2\sigma}\right)\right)\left(1 - P\left(\frac{N_{2k}}{\sigma} > \frac{d}{2\sigma}\right)\right) \\
&= 1 - \left(1 - 2Q\left(\frac{d}{2\sigma}\right)\right)\left(1 - Q\left(\frac{d}{2\sigma}\right)\right) \\
&= 3Q\left(\frac{d}{2\sigma}\right) - 2Q^2\left(\frac{d}{2\sigma}\right) \qquad (5)
\end{aligned}
$$

Substituting (4) and (5) in (3):

$$\boxed{P(E = 1) = 2.5Q\left(\frac{d}{2\sigma}\right) - 1.5Q^2\left(\frac{d}{2\sigma}\right)} \qquad (6)$$

The average symbol energy can be found from the constellation map as:

$$E_s = \sum_k P(s_k)|s_k|^2 = \frac{4(\frac{d^2}{4} + \frac{d^2}{4}) + 4(\frac{9d^2}{4} + \frac{d^2}{4})}{8}$$

$$E_s = \frac{3d^2}{2} \qquad (7)$$

Substituting (7) in (6):

$$\boxed{P_e = P(E = 1) = 2.5Q\left(\sqrt{\frac{E_s}{3N_0}}\right) - 1.5Q^2\left(\sqrt{\frac{E_s}{3N_0}}\right)} \qquad (8)$$

3 Design

```
Input: NO_OF_BITS, EB_N0_DB, d
Output: SER

M = 8
RIGHT = d / 2 + (M / 4 - 1) * d
```

```
LEFT = -RIGHT
constellation = [(±d/2 ± i*d, ±d/2)], i = 0, 1

map(symbol) = constellation[bin2dec(symbol)]
ML(Y) = argmin euclidean_distance(Y, constellation)

Sk = map(Bk.group(3))

for EB_N0 in EB_N0_DB
    Nk = AWGN(EB_N0, 2D)
    Yk = Sk + Nk
    sHat = dec2bin(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

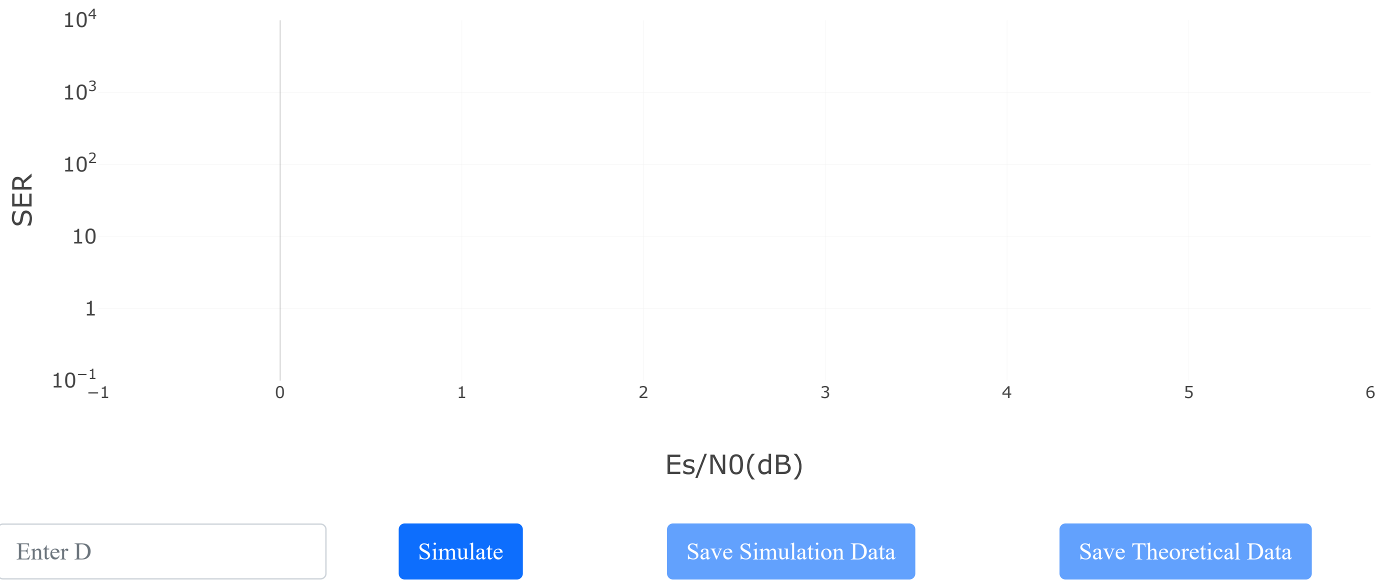
QAM.js

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

const S = new SimulationHelpers();
const D = new DomHelpers();

export default function QAM(d, SB_N0_DB, NO_OF_BITS) {
    const NO_OF_SYMBOLS = NO_OF_BITS / 3;
    const M = 8;
    const RIGHT = d / 2 + (M / 4 - 1) * d;
    const LEFT = -RIGHT;
    const Es = 1.5 * (d ** 2);
    const SER = new Array(SB_N0_DB.length);
    const SER_THEORETICAL = S.getTheoreticalSerQam8(SB_N0_DB, Es);
    const constellation = S
        .linspace(LEFT, RIGHT, d)
        .map((point) => [[point, d / 2], [point, -d / 2]])
        .flat();
    const Bk = S.randi([0, 1], NO_OF_BITS); // message
    const Sk = Bk.reduce((acc, bit, i) => { // modulation
        if (i % 3 === 0) {
            const tribit = [bit];
            acc.push(tribit);
        } else {
            acc[acc.length - 1].push(bit);
        }
        return acc;
    }, []).map((tribit) => constellation[parseInt(tribit.join(''), 2)]);
    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(([x1, y1]) => {
            let shortestDistance = Infinity;
            let closestPoint;
            constellation.forEach(([x2, y2]) => {
                const distance = Math.hypot(x2 - x1, y2 - y1);
                if (distance < shortestDistance) {
                    shortestDistance = distance;
                    closestPoint = [x2, y2];
                }
            });
            return S.dec2bin(D.indexOf(constellation, closestPoint), 3).split('').map((char) => Number(char));
        });
        const unchangedSymbols = sHat.reduce((acc, symbol, j) => {
            const s = Bk.slice(3 * j, 3 * j + 3);
            // eslint-disable-next-line no-param-reassign
            if (symbol.toString() === s.toString()) 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



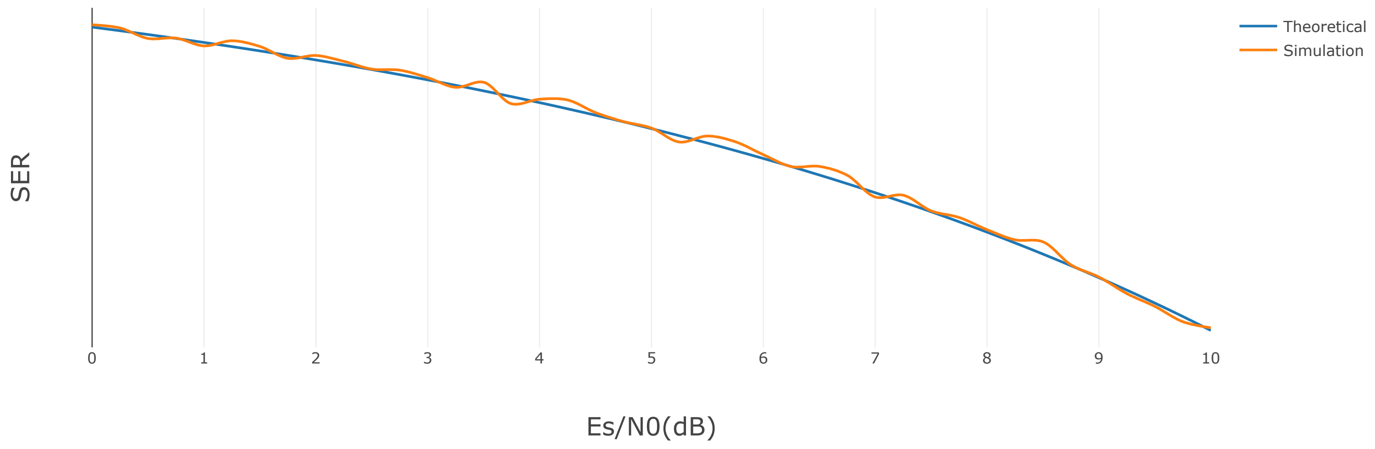


Fig.2 Simulation result for $d = 0.1$ with 3×10^5 bits

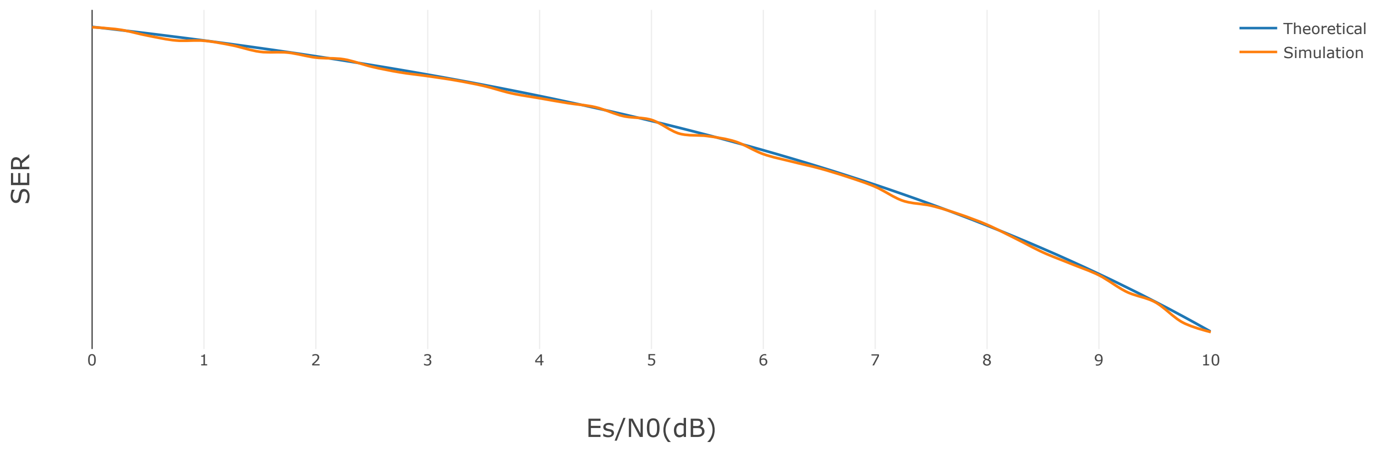


Fig.3 Simulation result for $d = 0.25$ with 3×10^5 bits

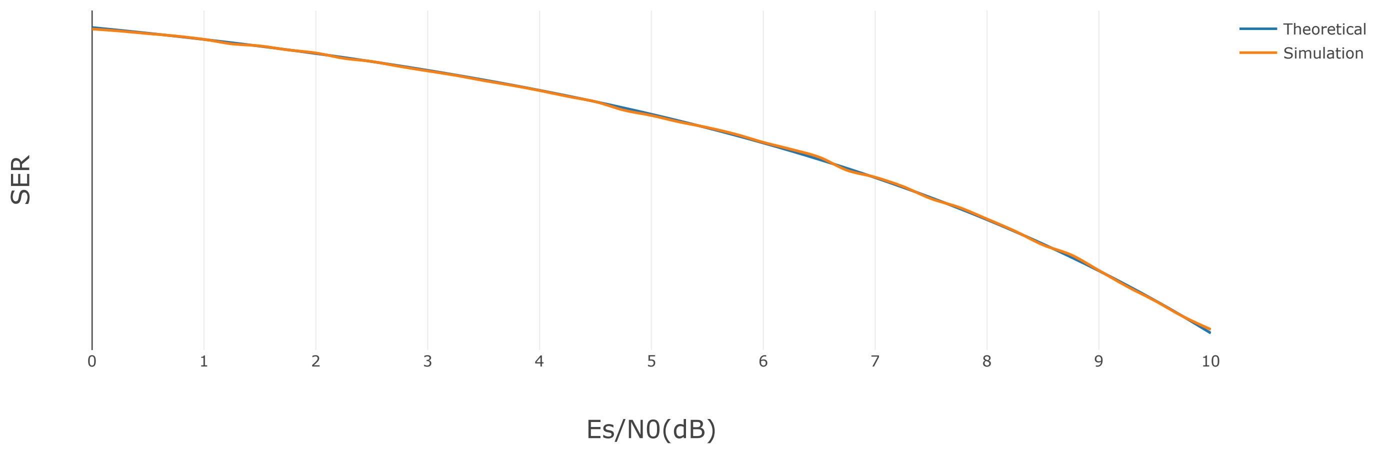


Fig.4 Simulation result for $d = 0.5$ with 3×10^5 bits

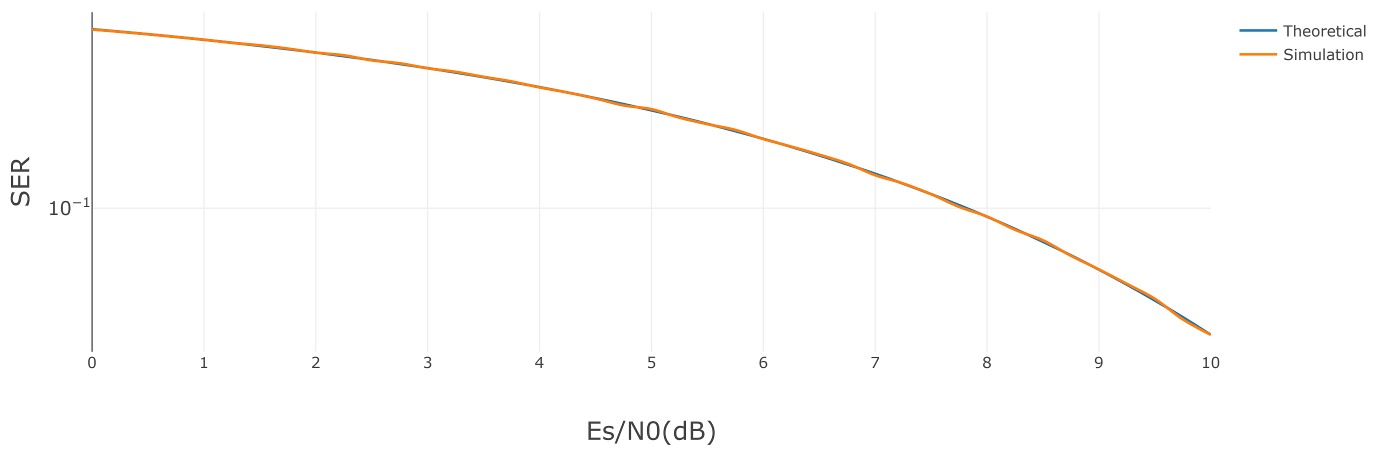


Fig.5 Simulation result for $d = 1$ with 3×10^5 bits

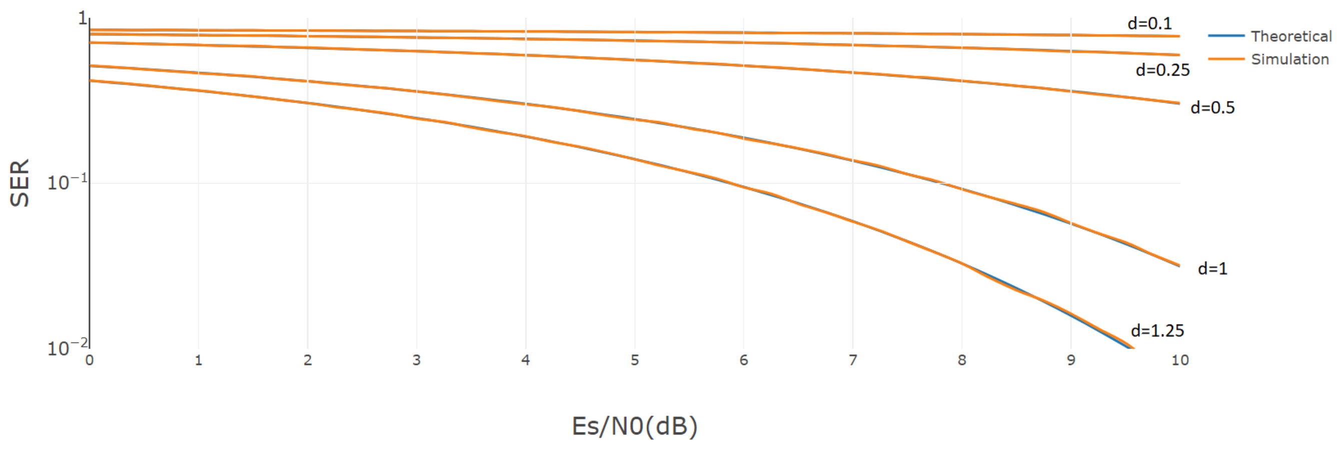


Fig.6 Comparison of simulation results for $d = 0.1, 0.25, 0.5, 1$

From the above figures, it can be seen that for a given d, 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.

From figure 6, we can see that the waterfall curve tends to flatten with decrease in d. This is because, with decrease in d the decision region becomes smaller and hence vastly increases the chances of symbol error. This also explains the fact that for larger d, the curves start from a lower SER closer to 0 and as $D \to \infty, SER \to 0$.