Design of MPSK Receivers and SER Analysis

Experiment - 4

1 Aim

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

2 Theory

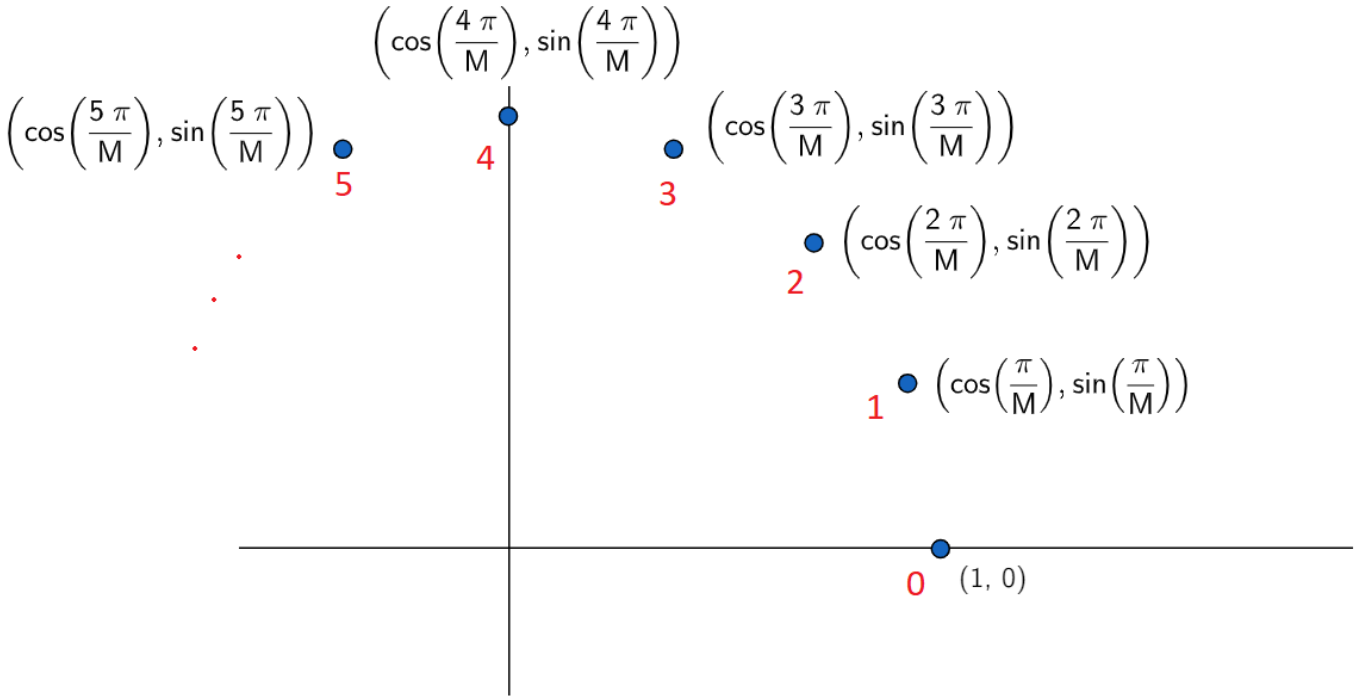


Fig.1 Constellation Map

2.1 ML Rule

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

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

2.2 SER

Let E be the error bit and $s_0, s_1, \ldots s_{M-1}$ 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,

$$P(E = 1 \mid s_0) = P(E = 1 \mid s_1) = \cdots = P(E = 1 \mid s_{M-1}) \tag{2}$$

From (1) and (2),

$$P(E = 1) = P(E = 1 \mid s_0) = \cdots = P(E = 1 \mid s_{M-1}) \tag{3}$$

Let us assume $s_0 = [\sqrt{E_s}, 0]$ was transmitted. Then,

$$Y_k = [\sqrt{E_s} + N_{1k}, N_{2k}]$$

It can be seen that Y_{1k}, Y_{2k} are independent gaussian random variables with means $\sqrt{E_s}, 0$ respectively and variance σ^2. Therefore:

$$p(Y) = \frac{1}{\pi N_0} e^{-\frac{(Y_{1k}-\sqrt{E_s})^2+Y_{2k}^2}{N_0}} \tag{4}$$

Transforming (4) into polar coordinates using:

$$V = \sqrt{Y_{1k}^2 + Y_{2k}^2}$$

$$\Theta = \arctan\left(\frac{Y_{2k}}{Y_{1k}}\right)$$

we get:

$$p_{V,\Theta}(v,\theta) = \frac{v}{\pi N_0} e^{\frac{-v^2+E_s-2\sqrt{E_s}v\cos(\theta)}{N_0}}$$

Integrating over v, we get the marginal PDF of Θ as:

$$p_\Theta(\theta) = \int_0^\infty p_{V,\Theta}(v,\theta)dv$$

$$\boxed{p_\Theta(\theta) = \frac{1}{2\pi} e^{-\frac{E_s\sin^2(\theta)}{N_0}} \int_0^\infty v e^{-\frac{\left(v-\sqrt{\frac{2E_s}{N_0}}\cos(\theta)\right)^2}{2}} dv} \tag{5}$$

For $\frac{E_s}{N_0} \gg 1$ and $|\theta| \leq \frac{\pi}{2}$, (5) can be approximted as:

$$p_\Theta(\theta) \approx \frac{E_s}{\pi N_0}\cos(\theta) e^{-\frac{E_s\sin^2(\theta)}{N_0}} \tag{6}$$

The descision region of s_0 can be described as $\{\theta : -\frac{\pi}{m} < \theta < \frac{\pi}{m}\}$. Therefore,

$$P_e = 1 - \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} p_\Theta(\theta)d\theta \tag{7}$$

Substituting (7) in (6) and substituting $u = \sqrt{\frac{E_s}{N_0}}\sin(\theta)$:

$$P_e \approx 1 - \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} \frac{E_s}{\pi N_0}\cos(\theta) e^{-\frac{E_s\sin^2(\theta)}{N_0}} d\theta$$

$$\approx \frac{2}{\sqrt{\pi}} \int_{\sqrt{\frac{2E_s}{N_0}}\sin(\frac{\pi}{M})}^\infty e^{-u^2} du$$

$$= 2Q\left(\sqrt{2\frac{E_s}{N_0}}\sin\left(\frac{\pi}{M}\right)\right)$$

$$\therefore \boxed{P_e = 2Q\left(\sqrt{2\frac{E_s}{N_0}}\sin\left(\frac{\pi}{M}\right)\right)} \tag{8}$$

3 Design

```
Input: M, NO_OF_BITS, SB_N0_DB
Output: SER

L = log2(M)
phases = 2*i*PI/M, i = 0, 1, ..., M-1
constellation = [(cos(phases), sin(phases))]

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

Sk = map(Bk.group(L))

for SB_N0 in SB_N0_DB
    Nk = AWGN(SB_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

MPSK.js

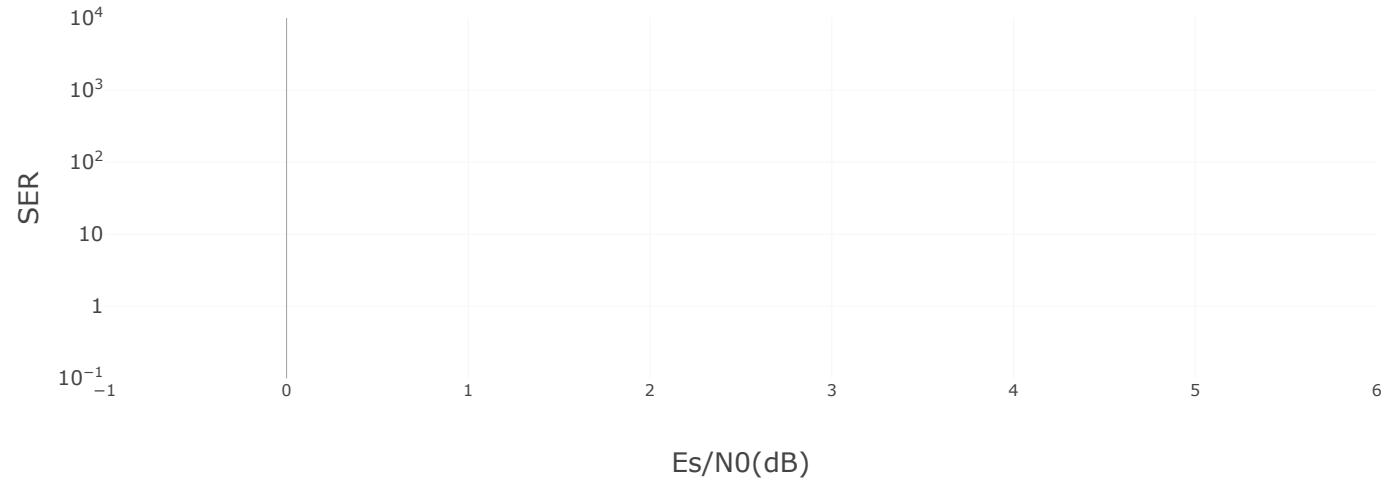
```
import DomHelpers from '../helpers/DomHelpers.js';
import SimulationHelpers from '../helpers/SimulationHelpers.js';

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

export default function MPSK(M, SB_N0_DB, NO_OF_SYMBOLS) {
    const L = Math.log2(M);
    const NO_OF_BITS = L * NO_OF_SYMBOLS;
    const SER = new Array(SB_N0_DB.length);
    const SER_THEORETICAL = S.getTheoreticalSerMpsk(SB_N0_DB, M);

    const phases = new Array(M).fill(0).map((_, i) => (2 * i * Math.PI) / M);
    const constellation = phases.map((phase) => [Math.cos(phase), Math.sin(phase)]);
    const Bk = S.randi([0, 1], NO_OF_BITS); // message
    const Sk = Bk.reduce((acc, bit, i) => { // modulation
        if (i % L === 0) {
            const lbit = [bit];
            acc.push(lbit);
        } else {
            acc[acc.length - 1].push(bit);
        }
        return acc;
    }, []).map((s) => constellation[parseInt(s.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), L).split('').map((char) => Number(char));
        }); // ML decision rule
        const unchangedSymbols = sHat.reduce((acc, symbol, j) => {
            const s = Bk.slice(L * j, L * j + L);
            // eslint-disable-next-line no-param-reassign
            if (symbol.toString() === s.toString()) acc += 1;
            return acc;
        }, 0);
        // eslint-disable-next-line no-param-reassign
        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 Inference



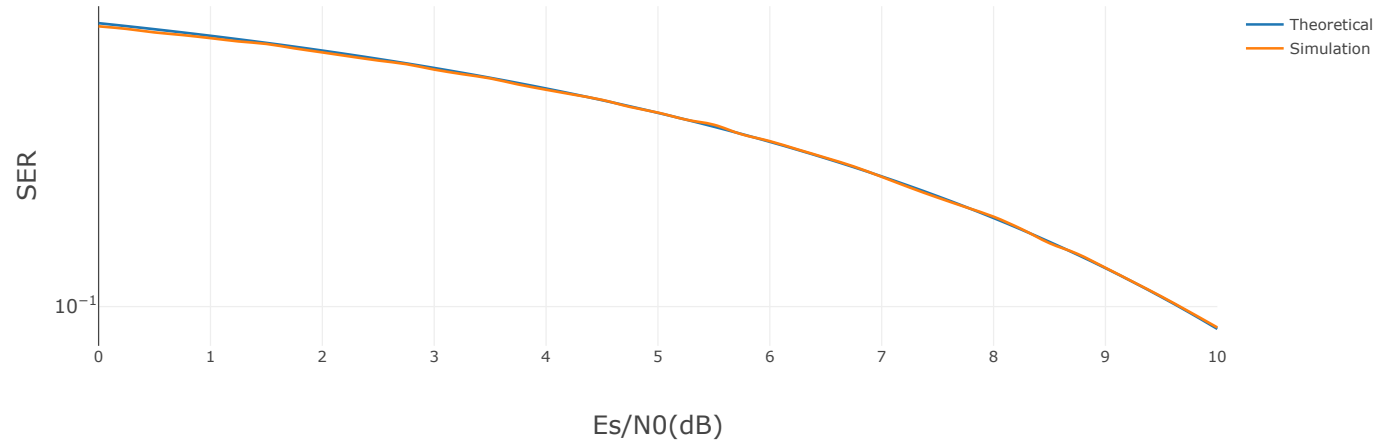


Fig.2 Simulation result for 8-MPSK with 3×10^5 bits

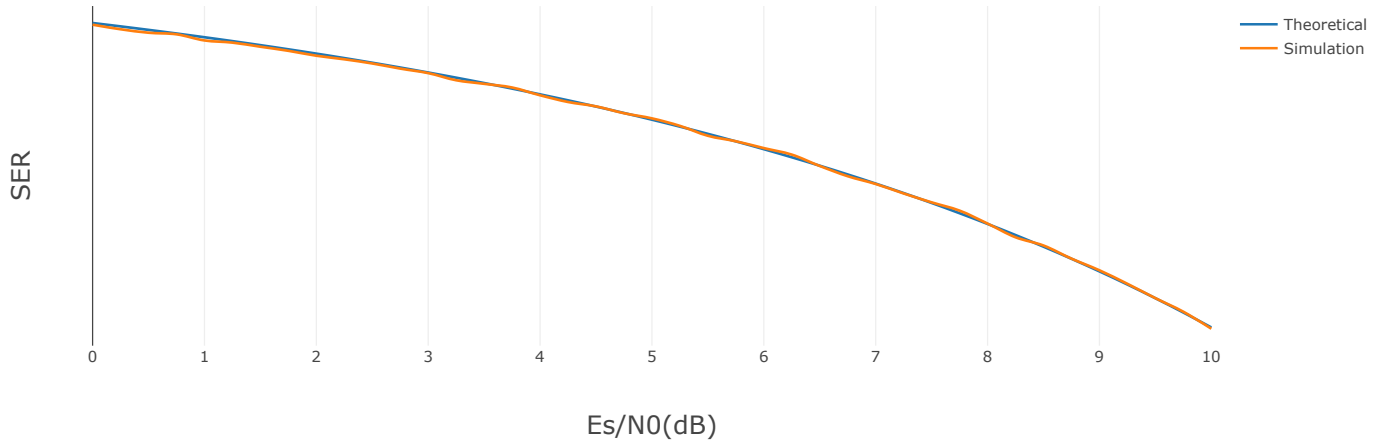


Fig.3 Simulation result for 16-MPSK with 4×10^5 bits

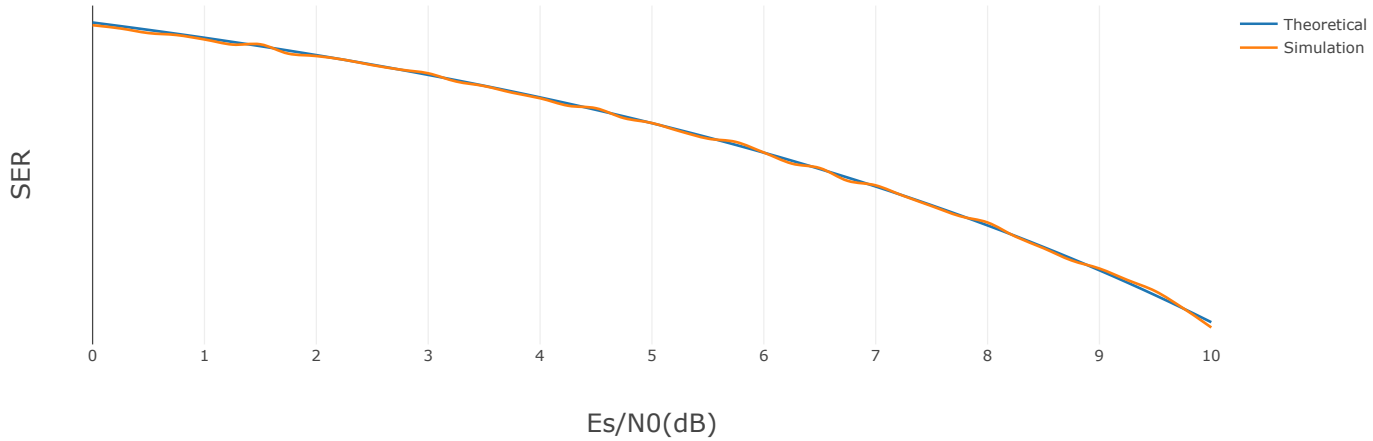


Fig.4 Simulation result for 32-MPSK with 5×10^5 bits

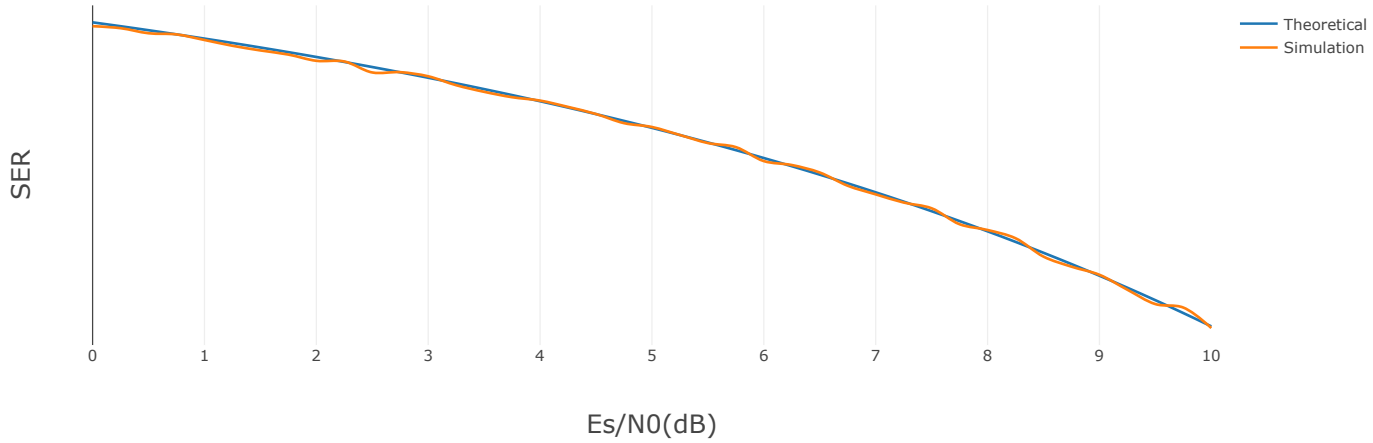


Fig.5 Simulation result for 64-MPSK with 6×10^5 bits

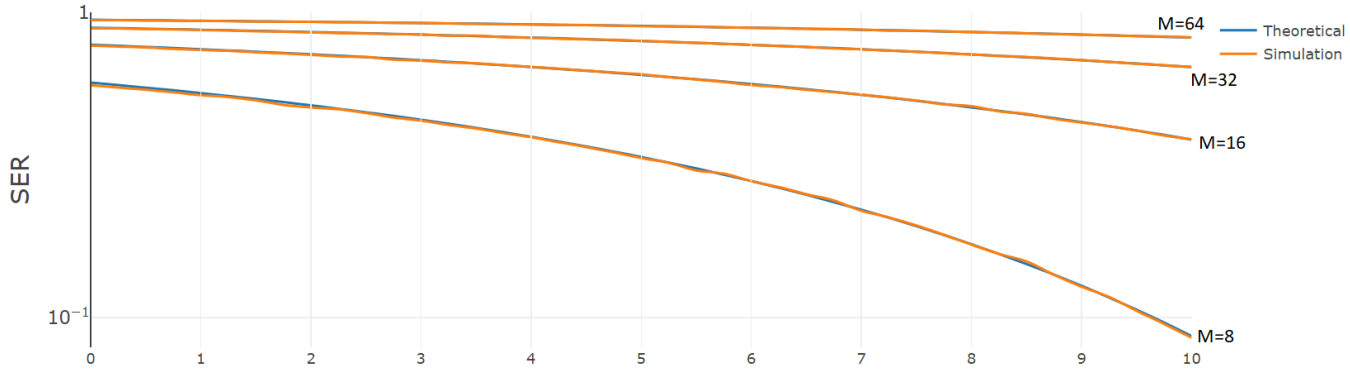


Fig.6 Comparison of simulation results for $M = 8, 16, 32, 64$

From the above figures, it can be seen that for a given M, 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 increase in M. This is because, with increase in M the decision region becomes smaller and hence vastly increases the chances of symbol error. This also explains the fact that for larger M, the curves start from a higher SER closer to 1 and as $M \to \infty, SER \to 1$.