Design of BPSK Receivers and BER Analysis

Experiment - 1

1 Aim

To design a discrete time BPSK communication receiver, and analyze the BER performance.

2 Theory

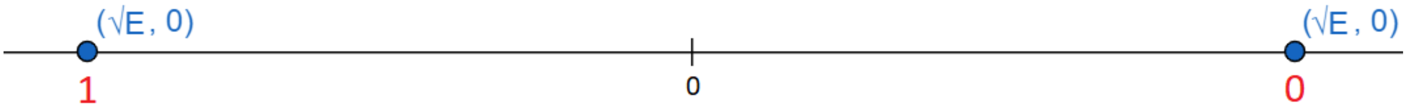


Fig.1 Constellation Map

2.1 ML Rule

Let Y be the output at the receiver and X be the message at the transmitter. Let N be the noise introduced by the AWGN channel. Let E_b be the bit energy. Then, for the k^{th} bit:

$$Y_k = X_k + N_k$$

Consider:

$$P(Y_k = y \mid X_k = \sqrt{E_b})$$

$$= \lim_{\epsilon \to 0} P(Y_k \in (y-\epsilon, y+\epsilon) \mid X_k = \sqrt{E_b})$$

$$= \lim_{\epsilon \to 0} P(X_k + N_k \in (y-\epsilon, y+\epsilon) \mid X_k = \sqrt{E_b})$$

$$= \lim_{\epsilon \to 0} P(N_k \in (y-\epsilon-\sqrt{E_b}, y+\epsilon-\sqrt{E_b}) \mid X_k = \sqrt{E_b})$$

$$= \lim_{\epsilon \to 0} 2\epsilon f_{Y_k}(y-\sqrt{E_b})$$

$$\boxed{P(Y_k = y \mid X_k = \sqrt{E_b}) = \lim_{\epsilon \to 0} 2\epsilon f_{Y_k}(y-\sqrt{E_b})} \tag{1}$$

Similarly:

$$\boxed{P(Y_k = y \mid X_k = -\sqrt{E_b}) = \lim_{\epsilon \to 0} 2\epsilon f_{Y_k}(y+\sqrt{E_b})} \tag{2}$$

Consider the ratio of (1) and (2):

$$\frac{P(Y_k = y \mid X_k = \sqrt{E_b})}{P(Y_k = y \mid X_k = -\sqrt{E_b})} = \lim_{\epsilon \to 0} \frac{2\epsilon f_{Y_k}(y-\sqrt{E_b})}{2\epsilon f_{Y_k}(y+\sqrt{E_b})}$$

$$= \frac{f_{Y_k}(y-\sqrt{E_b})}{f_{Y_k}(y+\sqrt{E_b})}$$

$$= \frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\sqrt{E_b})^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y+\sqrt{E_b})^2}{2\sigma^2}}}$$

$$= e^{\frac{(y+\sqrt{E_b})^2-(y-\sqrt{E_b})^2}{2\sigma^2}}$$

$$= e^{\frac{4y\sqrt{E_b}}{2\sigma^2}}$$

$$= e^{\frac{2y\sqrt{E_b}}{\sigma^2}}$$

$$\boxed{\frac{P(Y_k = y \mid X_k = \sqrt{E_b})}{P(Y_k = y \mid X_k = -\sqrt{E_b})} = e^{\frac{2y\sqrt{E_b}}{\sigma^2}}} \tag{3}$$

Let $B_k \in X$ be the transmitted bit and $\hat{B} \in Y$ be the the output at the receiver corresponding to B_k. Then the ML can be written as:

$$\hat{B} = \begin{cases} 1, & \frac{P(Y_k=y|B_k=1)}{P(Y_k=y|B_k=0)} > 1 \\ 0, & \frac{P(Y_k=y|B_k=1)}{P(Y_k=y|B_k=0)} < 1 \end{cases}$$

$$\hat{B} = \begin{cases} 1, & \ln\left(\frac{P(Y_k=y|B_k=1)}{P(Y_k=y|B_k=0)}\right) > 0 \\ 0, & \ln\left(\frac{P(Y_k=y|B_k=1)}{P(Y_k=y|B_k=0)}\right) < 0 \end{cases}$$

From the constellation diagram, we can write:

$$\hat{B} = \begin{cases} 1, & \ln\left(\frac{P(Y_k=y|X_k=-\sqrt{E_b})}{P(Y_k=y|X_k=\sqrt{E_b})}\right) > 0 \\ 0, & \ln\left(\frac{P(Y_k=y|X_k=-\sqrt{E_b})}{P(Y_k=y|X_k=\sqrt{E_b})}\right) < 0 \end{cases} \tag{4}$$

Substituting (3) in (4):

$$\hat{B} = \begin{cases} 1, & \frac{2y\sqrt{E_b}}{\sigma^2} < 0 \\ 0, & \frac{2y\sqrt{E_b}}{\sigma^2} > 0 \end{cases}$$

$$\hat{B} = \begin{cases} 1, & y < 0 \\ 0, & y > 0 \end{cases}$$

2.2 BER

Let E be the error bit. Then,

$$P(E = 1) = P(E = 1, B = 1) + P(E = 1, B = 0)$$

$$P(E = 1) = P(E = 1 \mid B = 1)P(B = 1) + P(E = 1 \mid B = 0)P(B = 0) \tag{5}$$

$$P(E = 1 \mid B = 1) = P(\hat{B} = 0 \mid B = 1)$$

$$= P(Y > 0 \mid B = 1)$$

$$= P(N - \sqrt{E_b} > 0 \mid B = 1)$$

$$= P(N > \sqrt{E_b} \mid B = 1)$$

$$= P(N > \sqrt{E_b})$$

$$= P\left(\frac{N}{\sigma} > \sqrt{\frac{E_b}{\sigma^2}}\right)$$

$$= \int_{\sqrt{\frac{E_b}{\sigma^2}}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

$$= Q\left(\sqrt{\frac{E_b}{\sigma^2}}\right)$$

$$= Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{6}$$

Similarly,

$$P(E = 1 \mid B = 0) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{7}$$

Substituting, (6) and (7) in (5),

$$\boxed{\therefore P_e = P(E = 1) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)}$$

3 Design

```
Input: NO_OF_BITS, EB_N0_DB
Output: BER

Xk = Bk(0 --> 1, 1 --> -1)
ML(point) = (0 if point > 0; 1 if point < 0)

for EB_N0 in EB_N0_DB
    Nk = AWGN(EB_N0, 1D)
    Yk = Xk + Nk
    bHat = ML(Yk)
    unchangedBits = count(bHat === Bk)
    BER = 1 - (unchangedBits/NO_OF_BITS)

plot(BER, BER_THEORETICAL)
```

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

4 JavaScript Code

BPSK.js

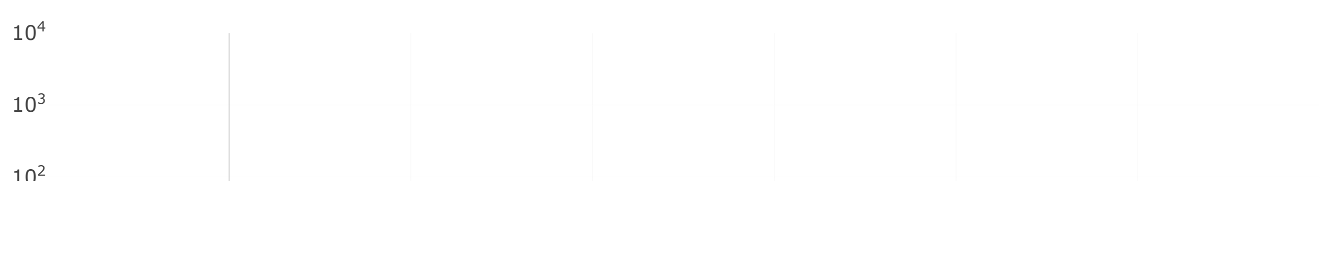
```
import SimulationHelpers from '../helpers/SimulationHelpers.js';

const S = new SimulationHelpers();

export default function BPSK(EB_N0_DB, NO_OF_BITS) {
    const BER_THEORETICAL = S.getTheoreticalBerBpsk(EB_N0_DB);
    const BER = new Array(EB_N0_DB.length);
    const Bk = S.randi([0, 1], NO_OF_BITS); // message
    const Xk = Bk.map((bit) => (bit === 0 ? 1 : -1)); // modulation
    for (let i = 0; i < EB_N0_DB.length; i += 1) {
        const Nk = S.getAWGN(EB_N0_DB[i], [NO_OF_BITS, 1]); // AWGN Noise
        const Yk = S.sum(Xk, Nk);
        const bHat = Yk.map((point) => Number(point < 0)); // ML decision rule
        const unchangedBits = Bk.reduce((acc, bit, j) => {
            // eslint-disable-next-line no-param-reassign
            if (bit === bHat[j]) acc += 1;
            return acc;
        }, 0);
        BER[i] = 1 - (unchangedBits / NO_OF_BITS);
    }
    return [BER, BER_THEORETICAL];
}
```

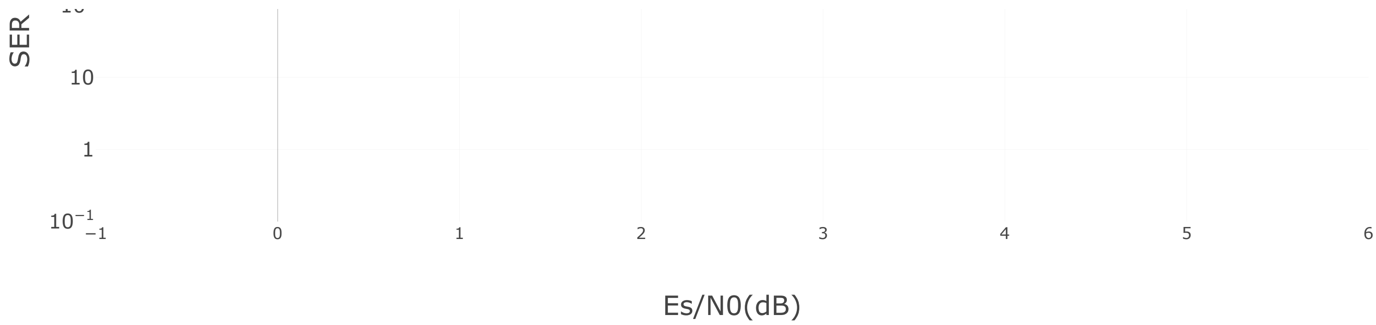
SimulationHelpers.js

```
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





Simulate   Save Simulation Data   Save Theoretical Data

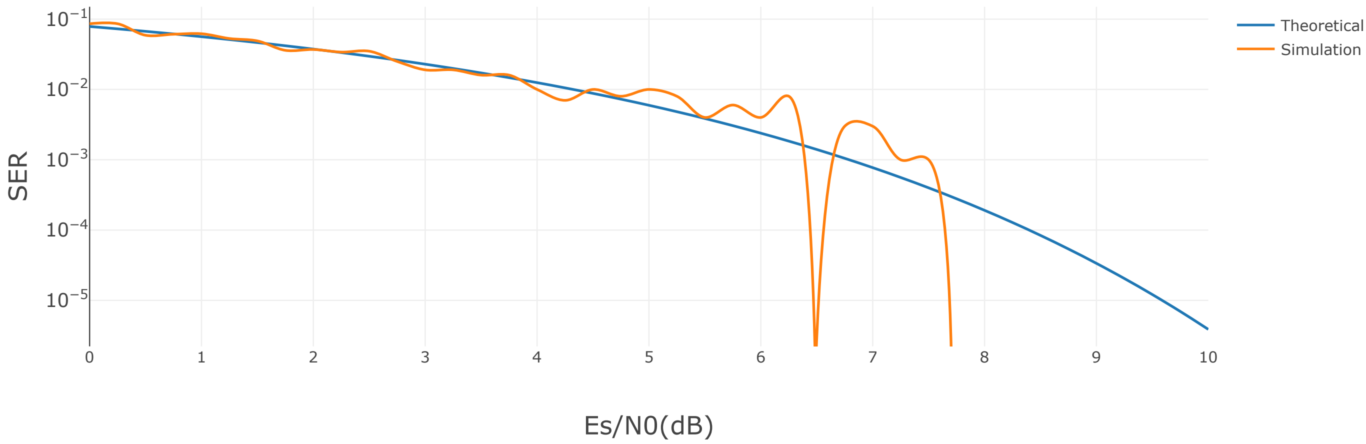


Fig.2 Simulation result with 10^3 bits

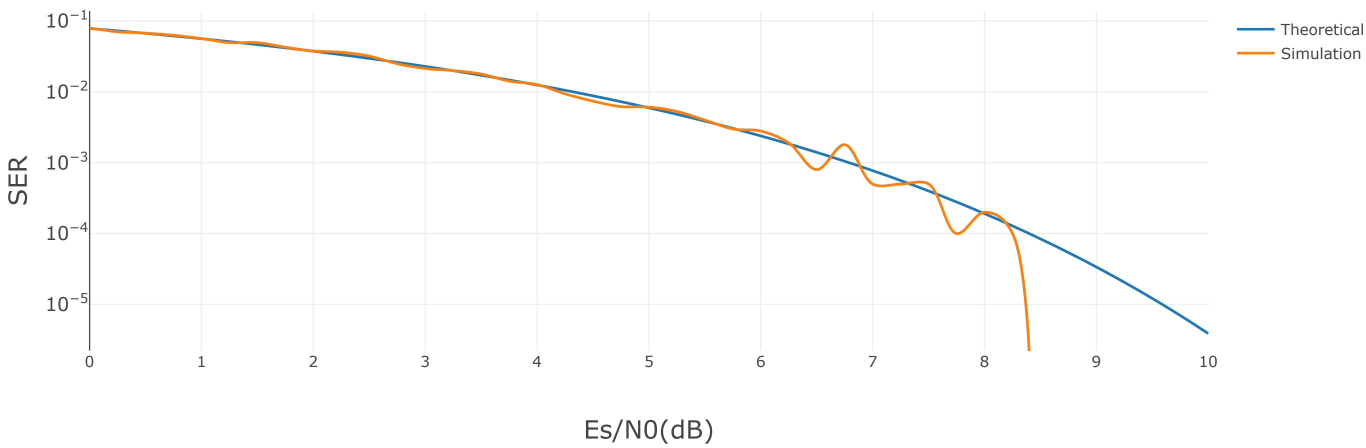
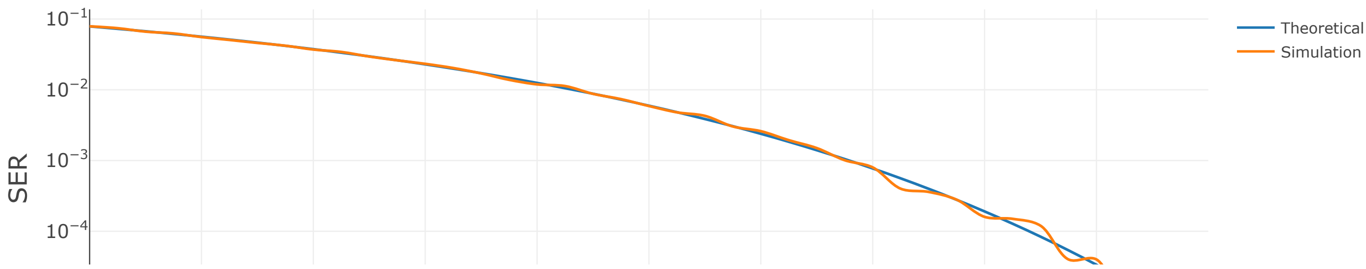


Fig.3 Simulation result with 10^4 bits



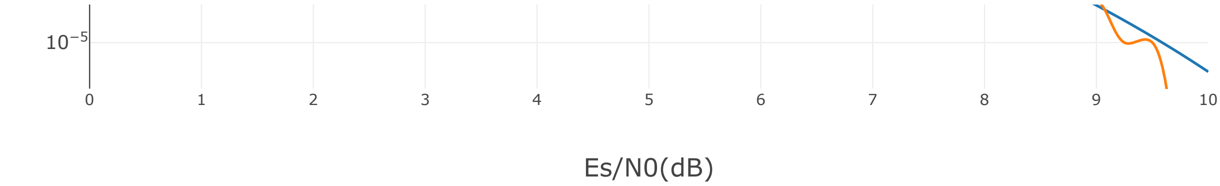


Fig.4 Simulation result with 10^5 bits

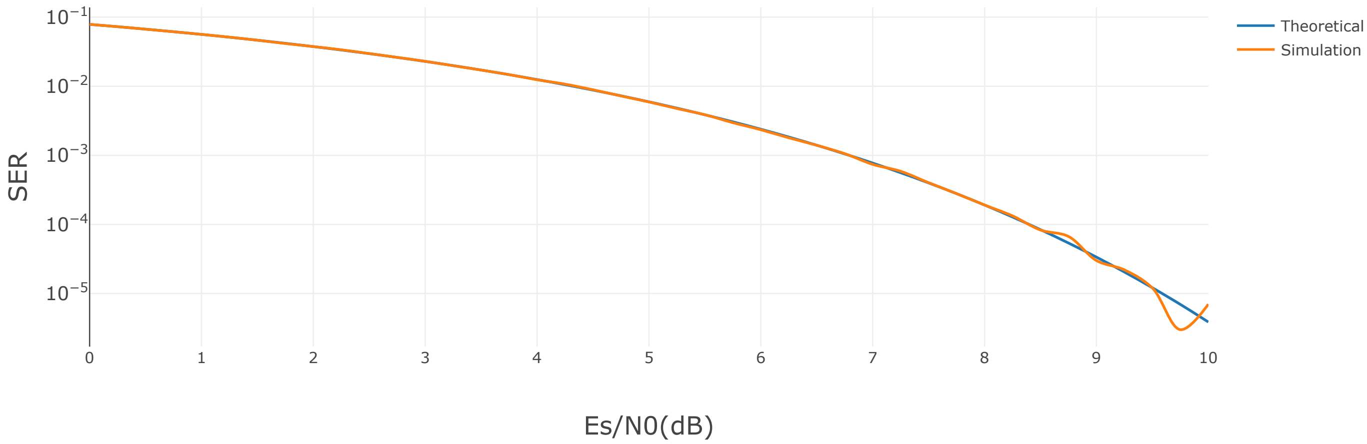


Fig.5 Simulation result with 10^6 bits

From the above figures, it can be seen that 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 in figure (5), to get a bit error rate of 10^{-5}, one needs to send at least 10^6 bits. Similar conclusions can be drawn from the other figures shown above. Also, it can be seen that the P_e reduces gradually with increase in $\frac{E_b}{N_0}$, the SNR per bit. This is due to the fact that as the SNR per bit 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.